

pgpool-II 4.1.2 Documentation

The Pgpool Global Development Group

Copyright © 2003-2020 The Pgpool Global Development Group

Legal Notice

Pgpool and Pgpool-II are Copyright © 2003-2020 by the Pgpool Global Development Group.

PostgreSQL are Copyright © 1996-2020 by the PostgreSQL Global Development Group.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS-IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Table of Contents

[Preface](#)

[What is Pgpool-II?](#)

[A Brief History of Pgpool-II](#)

[Conventions](#)

[Further Information](#)

[Restrictions](#)

[Bug Reporting Guidelines](#)

I. [Tutorial](#)

1. [Getting Started](#)

II. [Server Administration](#)

2. [Installation of Pgpool-II](#)

3. [Server Setup and Operation](#)

4. [Watchdog](#)

5. [Server Configuration](#)

6. [Client Authentication](#)

7. [Performance Considerations](#)

III. [Examples](#)

8. [Configuration Examples](#)

IV. [Reference](#)

I. [Server commands](#)

II. [PCP commands](#)

III. [Other commands](#)

IV. [SQL type commands](#)

V. [pgpool_adm extension](#)

V. [Appendixes](#)

A. [Release Notes](#)

[Index](#)

Preface

This book is the official documentation of **Pgpool-II**. It has been written by the **Pgpool-II** developers and other volunteers in parallel to the development of the **Pgpool-II** software. It describes all the functionality that the current version of **Pgpool-II** officially supports.

To make the large amount of information about **Pgpool-II** manageable, this book has been organized in several parts. Each part is targeted at a different class of users, or at users in different stages of their **Pgpool-II** experience:

- [Part I](#) is an informal introduction for new users.
 - [Part II](#) describes the installation and administration of the server. Everyone who runs a **Pgpool-II** server, be it for private use or for others, should read this part.
 - [Part III](#) explains several configuration examples so that users can choose the starting point of their actual systems.
 - [Part IV](#) contains reference information about SQL commands, client and server programs. This part supports the other parts with structured information sorted by command or program.
 - [Part V](#) is an appendix information such as release notes.
-

What is Pgpool-II?

Pgpool-II is a proxy software that sits between **PostgreSQL** servers and a **PostgreSQL** database client. It provides the following features:

Connection Pooling

Pgpool-II maintains established connections to the **PostgreSQL** servers, and reuses them whenever a new connection with the same properties (i.e. user name, database, protocol version, and other connection parameters if any) comes in. It reduces the connection overhead, and improves system's overall throughput.

Load Balancing

If a database is replicated (because running in either replication mode or master/slave mode), performing a **SELECT** query on any server will return the same result. **Pgpool-II** takes advantage of the replication feature in order to reduce the load on each **PostgreSQL** server. It does that by distributing **SELECT** queries among available servers, improving the system's overall throughput. In an ideal scenario, read performance could improve proportionally to the number of **PostgreSQL** servers. Load balancing works best in a scenario where there are a lot of users executing many read-only queries at the same time.

Automated fail over

If one of the database servers goes down or becomes unreachable, **Pgpool-II** will detach it and will continue operations by using the rest of database servers. There are some sophisticated features that help the automated failover including timeouts and retries.

Online Recovery

Pgpool-II can perform online recovery of database node by executing one command. When the online recovery is used with the automated fail over, a detached node by fail over is possible to attach as standby node automatically. It is possible to synchronize and attach new **PostgreSQL** server too.

Replication

Pgpool-II can manage multiple **PostgreSQL** servers. Activating the replication feature makes it possible

Pgpool-II can manage multiple PostgreSQL servers. Activating the replication feature makes it possible to create a real time backup on 2 or more PostgreSQL clusters, so that the service can continue without interruption if one of those clusters fails. Pgpool-II has built-in replication (native replication). However user can use external replication features including streaming replication of PostgreSQL.

Limiting Exceeding Connections

There is a limit on the maximum number of concurrent connections with PostgreSQL, and new connections are rejected when this number is reached. Raising this maximum number of connections, however, increases resource consumption and has a negative impact on overall system performance. Pgpool-II also has a limit on the maximum number of connections, but extra connections will be queued instead of returning an error immediately. However, you can configure to return an error when the connection limit is exceeded (4.1 or later).

Watchdog

Watchdog can coordinate multiple Pgpool-II, create a robust cluster system and avoid the single point of failure or split brain. To avoid the split brain, you need at least 3 Pgpool-II nodes. Watchdog can perform lifecheck against other pgpool-II nodes, to detect a fault of Pgpool-II. If active Pgpool-II goes down, standby Pgpool-II can be promoted to active, and take over Virtual IP.

In Memory Query Cache

In memory query cache allows to save a pair of SELECT statement and its result. If an identical SELECTs comes in, Pgpool-II returns the value from cache. Since no SQL parsing nor access to PostgreSQL are involved, using in memory cache is extremely fast. On the other hand, it might be slower than the normal path in some cases, because it adds some overhead of storing cache data.

Pgpool-II speaks PostgreSQL's backend and frontend protocol, and relays messages between a backend and a frontend. Therefore, a database application (frontend) thinks that Pgpool-II is the actual PostgreSQL server, and the server (backend) sees Pgpool-II as one of its clients. Because Pgpool-II is transparent to both the server and the client, an existing database application can be used with Pgpool-II almost without a change to its source code.

Pgpool-II works on Linux, Solaris, FreeBSD, and most of the UNIX-like architectures. Windows is not supported. Supported PostgreSQL server's versions are 6.4 and higher. If you are using PostgreSQL 7.3 or older, some features of Pgpool-II won't be available. But you shouldn't use such an old release anyway. You must also make sure that all of your PostgreSQL servers are using the same major version. In addition to this, we do not recommend mixing different PostgreSQL installation with different build options: including supporting SSL or not, to use --disable-integer-datetimes or not, different block size. These might affect part of functionality of Pgpool-II. The difference of PostgreSQL minor versions is not usually a problem. However we do not test every occurrence of minor versions and we recommend to use exact same minor version of PostgreSQL.

There are some restrictions to using SQL via Pgpool-II. See [Restrictions](#) for more details.

A Brief History of Pgpool-II

Pgpool-II started its life as a personal project by Tatsuo Ishii. In the project it was just a simple connection pooling software. So the name Pgpool came from the fact. The first version was in public in 2003.

In 2004, Pgpool 1.0 was released with the native replication feature (SQL statement based replication). In the same year 2.0 was released with load balancing, and support for version 3 frontend/backend protocol. In 2005, automated fail over and master slave mode support were added.

In 2006, Pgpool became Pgpool-II. The first release 1.0 eliminated many of restrictions in Pgpool, for example the number of PostgreSQL servers was up to 2 in Pgpool. Also many new features such as parallel query mode and PCP commands (PCP stands for "Pgpool Control Protocol") were added. Probably the most important change made between Pgpool and Pgpool-II was that the project was changed from a personal project to a group project owned by the Pgpool Development Group.

Conventions

The following conventions are used in the synopsis of a command: brackets ([and]) indicate optional parts. (In the synopsis of a Tcl command, question marks (?) are used instead, as is usual in Tcl.) Braces ({ and })

and vertical lines (|) indicate that you must choose one alternative. Dots (...) mean that the preceding element can be repeated.

Where it enhances the clarity, SQL commands are preceded by the prompt =>, and shell commands are preceded by the prompt \$. Normally, prompts are not shown, though.

An **administrator** is generally a person who is in charge of installing and running the server. A **user** could be anyone who is using, or wants to use, any part of the Pgpool-II system. These terms should not be interpreted too narrowly; this book does not have fixed presumptions about system administration procedures.

Further Information

Besides the documentation, that is, this book, there are other resources about Pgpool-II:

Web Site

The Pgpool-II [web site](#) is a central place providing official information regarding Pgpool-II: downloads, documentation, FAQ, mailing list archives and more.

Mailing Lists

The mailing lists are a good place to have your questions answered, to share experiences with other users, and to contact the developers. Consult the Pgpool-II web site for details.

Yourself!

pgpool-II is an open-source project. As such, it depends on the user community for ongoing support. As you begin to use Pgpool-II, you will rely on others for help, either through the documentation or through the mailing lists. Consider contributing your knowledge back. Read the mailing lists and answer questions. If you learn something which is not in the documentation, write it up and contribute it. If you add features to the code, contribute them.

Restrictions

This section describes current restrictions of Pgpool-II.

Functionality of PostgreSQL

If you use `pg_terminate_backend()` to stop a backend, this will trigger a failover. The reason why this happens is that PostgreSQL sends exactly the same message for a terminated backend as for a full postmaster shutdown. There is no workaround prior of version 3.6. From version 3.6, this limitation has been mitigated. If the argument to the function (that is a process id) is a constant, you can safely use the function. In extended protocol mode, you cannot use the function though.

Load Balancing

Multi-statement queries (multiple SQL commands on single line) are always sent to primary node (in streaming replication mode) or master node (in other modes). Usually Pgpool-II dispatch query to appropriate node, but it's not applied to multi-statement queries.

Authentication/Access Controls

In the replication mode or master/slave mode, trust and pam methods are supported. md5 is also supported since Pgpool-II 3.0. md5 is supported by using an authentication file `pool_passwd`. `scram-sha-256`, `cert`, and clear text password is also supported since Pgpool-II 4.0. `pool_passwd` is default name of the authentication file. Here are the steps to enable md5 authentication:

1. Login as the database's operating system user and type:

```
pg_md5 --md5auth --username=your_username your_passwd
```

user name and md5 encrypted password are registered into `pool_passwd`. If `pool_passwd` does not

exist yet, `pg_md5` command will automatically create it for you. The format of `pool_passwd` is `username:encrypted_passwd`.

2. You also need to add an appropriate `md5` entry to `pool_hba.conf`. See [Section 6.1](#) for more details.
3. Please note that the user name and password must be identical to those registered in PostgreSQL.
4. After changing `md5` password (in both `pool_passwd` and PostgreSQL of course), you need to execute `pgpool reload`.

Large objects

In streaming replication mode, **Pgpool-II** supports large objects.

In native replication mode, **Pgpool-II** supports large objects if the backend is PostgreSQL 8.1 or later. For this, you need to enable `lobj_lock_table` directive in `pgpool.conf`. Large object replication using backend function `lo_import` is not supported, however.

In other mode, including Slony mode, large objects are not supported.

Temporary tables

Creating/inserting/updating/deleting temporary tables are always executed on the master (primary) in master slave mode. `SELECT` on these tables is executed on master as well. However if the temporary table name is used as a literal in `SELECT`, there's no way to detect it, and the `SELECT` will be load balanced. That will trigger a "not found the table" error or will find another table having same name. To avoid the problem, use `/*NO LOAD BALANCE*/` SQL comment.

Note that such literal table names used in queries to access system catalogs do cause problems described above. `psql's \d` command produces such that query:

```
SELECT 't1'::regclass::oid;
```

In such that case **Pgpool-II** always sends the query to master and will not cause the problem.

Tables created by `CREATE TEMP TABLE` will be deleted at the end of the session by specifying `DISCARD ALL` in `reset_query_list` if you are using PostgreSQL 8.3 or later.

For 8.2.x or earlier, tables created by `CREATE TEMP TABLE` will not be deleted after exiting a session. It is because of the connection pooling which, from PostgreSQL's backend point of view, keeps the session alive. To avoid this, you must explicitly drop the temporary tables by issuing `DROP TABLE`, or use `CREATE TEMP TABLE ... ON COMMIT DROP` inside the transaction block.

Functions, etc. In Native Replication mode

There is no guarantee that any data provided using a context-dependent mechanism (e.g. random number, transaction ID, OID, SERIAL, sequence), will be replicated correctly on multiple backends. For SERIAL, enabling `insert_lock` will help replicating data. `insert_lock` also helps `SELECT setval()` and `SELECT nextval()`.

`INSERT/UPDATE` using `CURRENT_TIMESTAMP`, `CURRENT_DATE`, `now()` will be replicated correctly. `INSERT/UPDATE` for tables using `CURRENT_TIMESTAMP`, `CURRENT_DATE`, `now()` as their `DEFAULT` values will also be replicated correctly. This is done by replacing those functions by constants fetched from master at query execution time. There are a few limitations however:

In **Pgpool-II** 3.0 or before, the calculation of temporal data in table default value is not accurate in some cases. For example, the following table definition:

```
CREATE TABLE rel1(  
  d1 date DEFAULT CURRENT_DATE + 1  
)
```

is treated the same as:

```
CREATE TABLE rel1(  
d1 date DEFAULT CURRENT_DATE  
)
```

Pgpool-II 3.1 or later handles these cases correctly. Thus the column "d1" will have tomorrow as the default value. However this enhancement does not apply if extended protocols (used in JDBC, PHP PDO for example) or PREPARE are used.

Please note that if the column type is not a temporal one, rewriting is not performed. Such example:

```
foo bigint default (date_part('epoch'::text,('now'::text)::timestamp(3) with time zone) * (1000)::double precision)
```

Suppose we have the following table:

```
CREATE TABLE rel1(  
c1 int,  
c2 timestamp default now()  
)
```

We can replicate

```
INSERT INTO rel1(c1) VALUES(1)
```

since this turn into

```
INSERT INTO rel1(c1, c2) VALUES(1, '2009-01-01 23:59:59.123456+09')
```

However,

```
INSERT INTO rel1(c1) SELECT 1
```

cannot to be transformed, thus cannot be properly replicated in the current implementation. Values will still be inserted, with no transformation at all.

SQL type commands

[SQL type commands](#) cannot be used in extended query mode.

Multi-byte Characters

Pgpool-II does not perform encoding conversion between client and PostgreSQL for multi-byte characters. The encoding for the client and backends must be the same.

Multi-statement Query

Pgpool-II cannot process multi-statement queries. However, when Pgpool-II is connected by `psql`, It is no problem. `psql` parse multi-statement, send a statement one by one.

libpq

libpq is linked while building Pgpool-II. libpq version must be 3.0 or later. Building Pgpool-II with libpq version 2.0 will fail.

Bug Reporting Guidelines

When you find a bug in Pgpool-II, please register to our [bug tracking system](#).

I. Tutorial

This chapter explains how to get start with Pgpool-II.

Table of Contents

1. [Getting Started](#)
 - 1.1. [Installation](#)
 - 1.2. [Your First Replication](#)
 - 1.3. [Testing Replication](#)
 - 1.4. [Testing Load Balance](#)
 - 1.5. [Testing Fail Over](#)
 - 1.6. [Testing Online Recovery](#)
 - 1.7. [Architectural Fundamentals](#)

Chapter 1. Getting Started

1.1. Installation

In this section we assume that you have already installed Pgpool-II following an instruction described in [Part II](#). Alternatively you can use [pgpool_setup](#) to create a temporary installation of Pgpool-II and PostgreSQL.

1.2. Your First Replication

In this section we are going to explain how to manage a PostgreSQL cluster with streaming replication using Pgpool-II, which is one of most common setup.

Before going further, you should properly set up `pgpool.conf` with streaming replication mode. Sample configurations are provided with Pgpool-II, there configuration file are located at `/usr/local/etc` with default installation from source code. you can copy `pgpool.conf.sample-stream` as `pgpool.conf`.

```
cp /usr/local/etc/pgpool.conf.sample-stream pgpool.conf
```

If you plan to use `pgpool_setup`, type:

```
pgpool_setup
```

This will create a Pgpool-II with streaming replication mode installation, primary PostgreSQL installation, and a async standby PostgreSQL installation.

From now on, we assume that you use `pgpool_setup` to create the installation under current directory. Please note that the current directory must be empty before executing `pgpool_setup`.

To start the whole system, type:

```
./startall
```

Once the system starts, you can check the cluster status by issuing a pseudo SQL command called "show pool_nodes" to any of databases. `pgpool_setup` automatically creates "test" database. We use the database. Note that the port number is 11000, which is the default port number assigned to Pgpool-II by `pgpool_setup`.

```
$ psql -p 11000 -c "show pool_nodes" test
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay | last_status_change
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | /tmp | 11002 | up | 0.500000 | primary | 0 | false | 0 | 2019-01-31 10:23:09
1 | /tmp | 11003 | up | 0.500000 | standby | 0 | true | 0 | 2019-01-31 10:23:09
(2 rows)
```

The result shows that the "status" column is "up", which means the PostgreSQL is up and running, which is good.

1.3. Testing Replication

Let's test the replication functionality using a benchmark tool `pgbench`, which comes with the standard PostgreSQL installation. Type following to create the benchmark tables.

```
$ pgbench -i -p 11000 test
```

To see if the replication works correctly, directly connect to the primary and the standby server to see if they return identical results.

```
$ psql -p 11002 test
\dt
List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | pgbench_accounts | table | t-ishii
public | pgbench_branches | table | t-ishii
public | pgbench_history | table | t-ishii
public | pgbench_tellers | table | t-ishii
(4 rows)
\q
$ psql -p 11003 test
\dt
List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | pgbench_accounts | table | t-ishii
public | pgbench_branches | table | t-ishii
public | pgbench_history | table | t-ishii
public | pgbench_tellers | table | t-ishii
(4 rows)
```

The primary server (port 11002) and the standby server (port 11003) return identical results. Next, let's run `pgbench` for a while and check to results.


```

$ pgbench -p 11000 -T 10 test
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
duration: 10 s
number of transactions actually processed: 4276
latency average = 2.339 ms
tps = 427.492167 (including connections establishing)
tps = 427.739078 (excluding connections establishing)

$ psql -p 11002 -c "SELECT sum(abalance) FROM pgbench_accounts" test
sum
-----
216117
(1 row)

$ psql -p 11003 -c "SELECT sum(abalance) FROM pgbench_accounts" test
sum
-----
216117
(1 row)

```

Again, the results are identical.

1.4. Testing Load Balance

Pgpool-II allows read query load balancing. It is enabled by default. To see the effect, let's use `pgbench -S` command.

```

$ ./shutdownall
$ ./startall
$ pgbench -p 11000 -c 10 -j 10 -S -T 60 test
starting vacuum...end.
transaction type: <builtin: select only>
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 10
duration: 60 s
number of transactions actually processed: 1086766
latency average = 0.552 ms
tps = 18112.487043 (including connections establishing)
tps = 18125.572952 (excluding connections establishing)

$ psql -p 11000 -c "show pool_nodes" test
node_id | hostname | status | lb_weight | role | select_cnt | load_balance_node | replication_delay | last_status_change
-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | /tmp | 11002 | up | 0.500000 | primary | 537644 | false | 0 | 2019-01-31 11:51:58
1 | /tmp | 11003 | up | 0.500000 | standby | 548582 | true | 0 | 2019-01-31 11:51:58
(2 rows)

```

"select_cnt" column shows how many SELECT are dispatched to each node. Since with the default configuration, Pgpool-II tries to dispatch equal number of SELECT, the column shows almost same numbers.

Pgpool-II offers more sophisticated strategy for load balancing. See [Section 5.7](#) for more details.

1.5. Testing Fail Over

Pgpool-II allows an automatic fail over when PostgreSQL server goes down. In this case Pgpool-II sets the status of the server to "down" and continue the database operation using remaining servers.

```

$ pg_ctl -D data1 stop
waiting for server to shut down.... done
server stopped
$ psql -p 11000 -c "show pool_nodes" test
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay | last_status_change
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | /tmp | 11002 | up | 0.500000 | primary | 4276 | true | 0 | 2019-01-31 12:00:09
1 | /tmp | 11003 | down | 0.500000 | standby | 1 | false | 0 | 2019-01-31 12:03:07
(2 rows)

```

The standby node was shut down by `pg_ctl` command. **Pgpool-II** detects it and detaches the standby node. "show pool_nodes" command shows that the standby node is in down status. You can continue to use the cluster without the standby node:

```

$ psql -p 11000 -c "SELECT sum(abalance) FROM pgbench_accounts" test
sum
-----
216117
(1 row)

```

What happens if the primary server goes down? In this case, one of remaining standby server is promoted to new primary server. For this testing, we start from the state in which both nodes are up.

```

$ psql -p 11000 -c "show pool_nodes" test
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay | last_status_change
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | /tmp | 11002 | up | 0.500000 | primary | 0 | false | 0 | 2019-01-31 12:04:58
1 | /tmp | 11003 | up | 0.500000 | standby | 0 | true | 0 | 2019-01-31 12:04:58
(2 rows)

$ pg_ctl -D data0 stop
waiting for server to shut down.... done
server stopped
$ psql -p 11000 -c "show pool_nodes" test
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay | last_status_change
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | /tmp | 11002 | down | 0.500000 | standby | 0 | false | 0 | 2019-01-31 12:05:20
1 | /tmp | 11003 | up | 0.500000 | primary | 0 | true | 0 | 2019-01-31 12:05:20
(2 rows)

```

Now the primary node is changed from 0 to 1. What happens inside? When the node 0 goes down, **Pgpool-II** detects it and executes `failover_command` defined in `pgpool.conf`. Here is the content of the file.

```

#!/bin/sh
# Execute command by failover.
# special values: %d = node id
#                %h = host name
#                %p = port number
#                %D = database cluster path
#                %m = new master node id
#                %M = old master node id
#                %H = new master node host name
#                %P = old primary node id
#                %R = new master database cluster path
#                %r = new master port number
#                %% = '%' character
failed_node_id=$1
failed_host_name=$2
failed_port=$3
failed_db_cluster=$4
new_master_id=$5
old_master_id=$6
new_master_host_name=$7
old_primary_node_id=$8
new_master_port_number=$9
new_master_db_cluster=${10}
mydir=/home/t-ishii/tmp/Tutorial
log=$mydir/log/failover.log
pg_ctl=/usr/local/pgsql/bin/pg_ctl
cluster0=$mydir/data0
cluster1=$mydir/data1

date >> $log
echo "failed_node_id $failed_node_id failed_host_name $failed_host_name failed_port $failed_port failed_db_cluster $failed_db_c

if [ a"$failed_node_id" = a"$old_primary_node_id" ];then # master failed
! new_primary_db_cluster=${mydir}/data"$new_master_id"
echo $pg_ctl -D $new_primary_db_cluster promote >>$log # let standby take over
$pg_ctl -D $new_primary_db_cluster promote >>$log # let standby take over
sleep 2
fi

```

The script receives necessary information as parameters from **Pgpool-II**. If the primary server goes down, it executes "`pg_ctl -D data1 promote`", which should promote the standby server to a new primary server.

1.6. Testing Online Recovery

Pgpool-II allows to recover a downed node by technique called "Online Recovery". This copies data from the primary node to a standby node so that it sync with the primary. This may take long time and database may be updated during the process. That's no problem because in the streaming configuration, the standby will receive WAL log and applies it to catch up the primary. To test online recovery, let's start with previous cluster, where node 0 is in down state.

```

$ pcp_recovery_node -p 11001 -n 0
Password:
pcp_recovery_node -- Command Successful

$ psql -p 11000 -c "show pool_nodes" test
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay | last_status_change
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
0       | /tmp    | 11002 | up     | 0.500000 | standby | 0           | 0                 | 2019-01-31 12:06:48
1       | /tmp    | 11003 | up     | 0.500000 | primary | 0           | 0                 | 2019-01-31 12:05:20
(2 rows)

```

`pcp_recovery_node` is one of control commands coming with **Pgpool-II** installation. The argument `-p` is to specify the port number assigned to the command, which is 11001 set by `pgpool_setup`. The argument `-n` is to specify the node id to be recovered. After executing the command, node 0 returned to "up" status.

The script executed by `pcp_recovery_node` is specified as "recovery_1st_stage_command" in `pgpool.conf`. Here is the file installed by `pgpool_setup`.

```
#!/bin/sh
psql=/usr/local/pgsql/bin/psql
DATADIR_BASE=/home/t-ishii/tmp/Tutorial
PGSUPERUSER=t-ishii

master_db_cluster=$1
recovery_node_host_name=$2
DEST_CLUSTER=$3
PORT=$4
recovery_node=$5

pg_rewind_failed="true"

log=$DATADIR_BASE/log/recovery.log
echo >> $log
date >> $log
if [ $pg_rewind_failed = "true" ];then

$psql -p $PORT -c "SELECT pg_start_backup('Streaming Replication', true)" postgres

echo "source: $master_db_cluster dest: $DEST_CLUSTER" >> $log

rsync -C -a -c --delete --exclude postgresql.conf --exclude postmaster.pid \
--exclude postmaster.opts --exclude pg_log \
--exclude recovery.conf --exclude recovery.done \
--exclude pg_xlog \
$master_db_cluster/ $DEST_CLUSTER/

rm -fr $DEST_CLUSTER/pg_xlog
mkdir $DEST_CLUSTER/pg_xlog
chmod 700 $DEST_CLUSTER/pg_xlog
rm $DEST_CLUSTER/recovery.done
fi
cat > $DEST_CLUSTER/recovery.conf $!<:&lt;:EOF
standby_mode      = 'on'
primary_conninfo  = 'port=$PORT user=$PGSUPERUSER'
recovery_target_timeline='latest'
restore_command = 'cp $DATADIR_BASE/archivedir/%f "%p" 2> /dev/null'
EOF

if [ $pg_rewind_failed = "true" ];then
$psql -p $PORT -c "SELECT pg_stop_backup()" postgres
fi

if [ $pg_rewind_failed = "false" ];then
cp /tmp/postgresql.conf $DEST_CLUSTER/
fi
```

1.7. Architectural Fundamentals

`Pgpool-II` is a proxy server sitting between clients and PostgreSQL. `Pgpool-II` understands the wire level protocol used by PostgreSQL called "frontend and backend protocol". For more details of the protocol, see the PostgreSQL manual. No modified PostgreSQL is required to use `Pgpool-II` (more precisely, you will need a few extensions to use full functions of `Pgpool-II`). So `Pgpool-II` can cope with variety of PostgreSQL versions. In theory, even the earliest version of PostgreSQL can be used with `Pgpool-II`. Same thing can be said to client side. As long as it follows the protocol, `Pgpool-II` happily accept connections from it, no matter what kind of languages or drivers it uses.

`Pgpool-II` consists of multiple process. There is a main process, which is the parent process of all other process. It is responsible for forking child process each of which accepts connections from clients. There are some worker process those are forked from the main process as well, which is responsible for detecting streaming replication delay. There is also a special process called "pcp process", which is solely used for management of `Pgpool-II` itself. `Pgpool-II` has a built-in high availability function called "watchdog". Watchdog consists of some process. For more details of watchdog, see [Chapter 4](#).

Figure 1-1. Process architecture of `Pgpool-II`

II. Server Administration

This part covers topics that are of interest to Pgpool-II administrators.

Table of Contents

2. [Installation of Pgpool-II](#)

- 2.1. [Planning](#)
- 2.2. [Installation of Pgpool-II](#)
- 2.3. [Requirements](#)
- 2.4. [Getting The Source](#)
- 2.5. [Installing Pgpool-II](#)
- 2.6. [Installing pgpool_recovery](#)
- 2.7. [Installing pgpool-regclass](#)
- 2.8. [Creating insert_lock table](#)
- 2.9. [Compiling and installing documents](#)
- 2.10. [Installation from RPM](#)
- 2.11. [Tips for Installation](#)

3. [Server Setup and Operation](#)

- 3.1. [The Pgpool-II User Account](#)
- 3.2. [Configuring pcp.conf](#)
- 3.3. [Configuring Pgpool-II](#)
- 3.4. [Configuring backend information](#)

4. [Watchdog](#)

- 4.1. [Introduction](#)
- 4.2. [Integrating external lifecheck with watchdog](#)
- 4.3. [Restrictions on watchdog](#)
- 4.4. [Architecture of the watchdog](#)

5. [Server Configuration](#)

- 5.1. [Setting Parameters](#)
- 5.2. [Connections and Authentication](#)
- 5.3. [Running mode](#)
- 5.4. [Backend Settings](#)
- 5.5. [Connection Pooling](#)
- 5.6. [Error Reporting and Logging](#)
- 5.7. [Load Balancing](#)
- 5.8. [Health Check](#)
- 5.9. [Failover and Failback](#)
- 5.10. [Online Recovery](#)
- 5.11. [Streaming Replication Check](#)
- 5.12. [In Memory Query Cache](#)
- 5.13. [Secure Socket Layer \(SSL\)](#)
- 5.14. [Watchdog](#)
- 5.15. [Misc Configuration Parameters](#)

6. [Client Authentication](#)

- 6.1. [The pool_hba.conf File](#)
- 6.2. [Authentication Methods](#)
- 6.3. [Using different methods for frontend and backend authentication](#)
- 6.4. [Using AES256 encrypted passwords in pool_passwd](#)

7. [Performance Considerations](#)

- 7.1. [Resource Requirement](#)
 - 7.2. [Managing Client Connections](#)
 - 7.3. [Read Query Load Balancing](#)
 - 7.4. [In Memory Query Caching](#)
 - 7.5. [Relation Cache](#)
 - 7.6. [Other Performance Considerations](#)
-

Chapter 2. Installation of Pgpool-II

2.1. Planning

Since Pgpool-II is a tool to manage PostgreSQL, we need to decide how to deploy them first. In addition, it is possible to have multiple number of Pgpool-II installations to enhance the availability of Pgpool-II itself. We need to plan how many installations of Pgpool-II is required before hand. In this chapter we first discuss the running mode of PostgreSQL then the deployment of Pgpool-II.

2.1.1. Running mode of PostgreSQL

It is possible to have more than or equal to one installation of PostgreSQL, it is common to have more than 2 installations of it because if there's only one installation, the whole database system goes down if the PostgreSQL is not available. When we use two or more PostgreSQL servers, it is necessary to sync the databases in some way. We call the methods of syncing databases as "running mode". The most popular mode ever used is "streaming replication mode". Unless there's necessity to have special consideration, it is recommended to use the streaming replication mode. See [Section 3.3.2](#) for more details of running mode.

The next thing we need to consider is how many PostgreSQL installations we want. If there are two, we can continue to operate the database system. However it is not uncommon to use more than two PostgreSQL if you want to employ read query load balancing by running multiple read quires on multiple servers. Pgpool-II provides rich variety of options to tune load balancing. See [Section 5.7](#) for more details.

Since it is possible to add PostgreSQL servers later on in Pgpool-II, two PostgreSQL can be a good starter for you.

2.1.2. Deployment of Pgpool-II

Although it is possible to use only one Pgpool-II, we recommend to use more than 1 Pgpool-II to avoid whole database unavailability due to the Pgpool-II being down. Multiple Pgpool-II work together and monitor each other. One of them is called "master" and it has a virtual IP. Clients do not need to aware that there are multiple Pgpool-II because they always access the same VIP. (See [Section 4.1](#) for watchdog). If one of Pgpool-II goes down, other Pgpool-II takes over the master role.

Since it is not allowed to have multiple master, watchdog votes to decide a new master. If there are even number of Pgpool-II, it is impossible to decide the new master by voting. Thus we recommend to deploy Pgpool-II in more than 3 odd numbers.

Please note that it is possible to have Pgpool-II and PostgreSQL on a same server. For example you can have only three servers to run both Pgpool-II and PostgreSQL on each of it.

You can find a production level detailed example using three Pgpool-II and two PostgreSQL in streaming replication mode in [Section 8.3](#) for those who want to have a production level Pgpool-II installation today.

2.2. Installation of Pgpool-II

This chapter describes the installation of Pgpool-II. First, installation from source code distribution is explained. Then installation from RPM packages is explained.

2.3. Requirements

In general, a modern Unix-compatible platform should be able to run Pgpool-II. Windows is not supported.

The following software packages are required for building Pgpool-II:

- GNU make version 3.80 or newer is required; other make programs or older GNU make versions will **not** work. (GNU make is sometimes installed under the name `gmake`.) To test for GNU make enter:

```
make --version
```

- You need an ISO/ANSI C compiler (at least C89-compliant). Recent versions of GCC are recommended, but Pgpool-II is known to build using a wide variety of compilers from different vendors.
- `tar` is required to unpack the source distribution, in addition to `gzip`.
- Several packages of PostgreSQL are required to install Pgpool-II. You install `postgresql-libs` and `postgresql-devel` packages from rpm.

If you are building from a Git tree instead of using a released source package, or if you want to do server development, you also need the following packages:

- Flex and Bison are needed to build from a Git checkout, or if you changed the actual scanner and parser definition files. If you need them, be sure to get Flex 2.5.31 or later and Bison 1.875 or later. Other `lex` and `yacc` programs cannot be used.

If you need to get a GNU package, you can find it at your local GNU mirror site (see <http://www.gnu.org/order/ftp.html> for a list) or at <ftp://ftp.gnu.org/gnu/>.

Also check that you have sufficient disk space. You will need about 40 MB for the source tree during compilation and about 20 MB for the installation directory. If you are going to run the regression tests you will temporarily need up to an extra 4 GB. Use the `df` command to check free disk space.

2.4. Getting The Source

The Pgpool-II 4.1.2 sources can be obtained from the download section of our website: <http://www.pgpool.net>. You should get a file named `pgpool-II-4.1.2.tar.gz`. After you have obtained the file, unpack it:

```
tar xf pgpool-II-4.1.2.tar.gz
```

This will create a directory `pgpool-II-4.1.2` under the current directory with the Pgpool-II sources. Change into that directory for the rest of the installation procedure.

2.5. Installing Pgpool-II

After extracting the source tarball, execute the `configure` script.

```
./configure
```

You can customize the build and installation process by supplying one or more of the following command line options to `configure`:

`--prefix=path`

Specifies the top directory where Pgpool-II binaries and related files like docs will be installed in. Default value is `/usr/local`.

`--with-pgsql=path`

Specifies the top directory where PostgreSQL's client libraries are installed. Default value is the path

provided by `pg_config` command.

`--with-openssl`

Pgpool-II binaries will be built with OpenSSL support. OpenSSL support is disabled by default.

`--enable-sequence-lock`

Use `insert_lock` compatible with Pgpool-II 3.0 series (until 3.0.4). Pgpool-II locks against a row in the sequence table. PostgreSQL 8.2 or later which was released after June 2011 cannot use this lock method.

`--enable-table-lock`

Use `insert_lock` compatible with Pgpool-II 2.2 and 2.3 series. Pgpool-II locks against the insert target table. This lock method is deprecated because it causes a lock conflict with `VACUUM`.

`--with-memcached=path`

Pgpool-II binaries will use `memcached` for in memory query cache. You have to install [libmemcached](#).

`--with-pam`

Pgpool-II binaries will be built with PAM authentication support. PAM authentication support is disabled by default.

```
make
make install
```

This will install Pgpool-II. (If you use Solaris or FreeBSD, replace `make` with `gmake`)

2.6. Installing `pgpool_recovery`

Pgpool-II need function of `pgpool_recovery`, `pgpool_remote_start` and `pgpool_switch_xlog`, when you use the online recovery that describes latter. Also `pgpoolAdmin` of management tool, stop, restart or reload a PostgreSQL on the screen by use `pgpool_pgctl`. It is enough, if these function installed in `template1` first. These function do not needed that install in all databases.

This is required in all Pgpool-II installation.

```
$ cd pgpool-II-4.1.2/src/sql/pgpool-recovery
$ make
$ make install
```

After this:

```
$ psql template1
=# CREATE EXTENSION pgpool_recovery;
```

or

```
$ psql -f pgpool-recovery.sql template1
```

With Pgpool-II 3.3 or later, you need to tweak `postgresql.conf`. Suppose the path to `pg_ctl` is `/usr/local/pgsql/bin/pg_ctl`. Then you add following to `postgresql.conf`.

```
pgpool.pg_ctl = '/usr/local/pgsql/bin/pg_ctl'
```


Probably you want to execute following after this:

```
$ pg_ctl reload -D /usr/local/pgsql/data
```

2.7. Installing pgpool-regclass

If you are using PostgreSQL 9.4 or later, you can skip this section.

If you are using PostgreSQL 8.0 to PostgreSQL 9.3, installing `pgpool_regclass` function on all PostgreSQL to be accessed by Pgpool-II is strongly recommended, as it is used internally by Pgpool-II. Without this, handling of duplicate table names in different schema might cause trouble (temporary tables aren't a problem). If you are using PostgreSQL 9.4 or later, installing `pgpool_regclass` is not necessary since an equivalent (`to_regclass`) is included in the PostgreSQL core.

```
$ cd pgpool-II-4.1.2/src/sql/pgpool-regclass
$ make
$ make install
```

After this:

```
$ psql template1
=# CREATE EXTENSION pgpool_regclass;
```

or

```
$ psql -f pgpool-regclass.sql template1
```

Executing `CREATE EXTENSION` or `pgpool-regclass.sql` should be performed on every databases accessed via Pgpool-II. However, you do not need to do this for a database created after the execution of `CREATE EXTENSION` or `psql -f pgpool-regclass.sql template1`, as this template database will be cloned to create new databases.

2.8. Creating insert_lock table

If you are not going to use the native replication mode, you can skip this section.

If you plan to use native replication mode and `insert_lock`, creating `pgpool_catalog.insert_lock` table for mutual exclusion is strongly recommended. Without this, `insert_lock` works so far. However in that case Pgpool-II locks against the insert target table. This behavior is same table lock conflicts with `VACUUM`, so `INSERT` processing may be thereby kept waiting for a long time.

```
$ cd pgpool-II-4.1.2/src/sql
$ psql -f insert_lock.sql template1
```

Executing `insert_lock.sql` should be performed on every databases accessed via Pgpool-II. You do not need to do this for a database created after the execution of `psql -f insert_lock.sql template1`, as this template database will be cloned to create new databases.

2.9. Compiling and installing documents

2.9.1. Tool Sets

Pgpool-II documents are written in SGML (more precisely, DocBook, which is a language implemented using

SGML). To generate readable HTML documents, you need to compile them using docbook tools. To install Docbook tools on RHEL or similar systems, use:

```
yum install docbook-dtds docbook-style-dsssl docbook-style-xsl libxslt openjade
```

2.9.2. Compiling docs

Once the tool sets are installed on the system, you can compile the docs:

```
$ cd doc
$ make
$ cd ..
$ cd doc.ja
$ make
```

You will see English HTML docs under `doc/src/sgml/html`, and online docs under `sgml/man[1-8]`. Japanese docs can be found under `doc.ja/src/sgml/html`, and online docs under `sgml/man[1-8]`.

2.10. Installation from RPM

This chapter describes the installation of Pgpool-II from RPM. If you are going to install from the source code, please check [Section 2.2](#).

2.10.1. Installing RPM

Pgpool-II official RPMs can be obtained from <http://www.pgpool.net/yum>.

For RHEL and its derivatives do following once:

```
yum install http://www.pgpool.net/yum/rpms/4.0/redhat/rhel-7-x86_64/pgpool-II-release-4.0-1.noarch.rpm
```

Then:

```
yum install pgpool-II-pg11
```

pg11 means PostgreSQL 11. Pgpool-II needs PostgreSQL's library and extensions directory. Since the directory paths are different in the particular PostgreSQL versions, You must choose appropriate RPM for your PostgreSQL rpm installation. We also assume you are using [PostgreSQL community rpms](#). Optionally you can install:

```
yum install pgpool-II-pg11-debuginfo
```

which makes it easier to retrieve debugging symbols from the core or the backtrace. We recommend to install it. There is an optional package for developers.

```
yum install pgpool-II-pg11-devel
```

This installs header files which developers are interested in

On all the PostgreSQL servers you need to install:

```
yum install pgpool-II-pg11-extensions
```

2.10.2. Configuration with RPM

All the Pgpool-II configuration files live in `/etc/pgpool-II`. Please refer to [Section 3.3](#) to see how to set up configuration files.

2.10.3. Starting/stopping Pgpool-II

On RHEL7/CentOS 7, do this once. Do this once, if set the automatic startup of Pgpool-II.

```
systemctl enable pgpool.service
```

After this, restart the whole system or do this. Please note that PostgreSQL servers must have been started before this.

```
systemctl start pgpool.service
```

To stop Pgpool-II, do this once. Pgpool-II must need to stop, before PostgreSQL is stopped.

```
systemctl stop pgpool.service
```

After this, you can stop PostgreSQL servers.

On RHEL6/CentOS 6, do this once.

```
chkconfig pgpool on
```

After this, restart the whole system or:

```
service start pgpool
```

Please note that PostgreSQL servers must have been started before this. To stop Pgpool-II:

```
service stop pgpool
```

After this, you can stop PostgreSQL servers.

2.11. Tips for Installation

This chapter gathers random tips for installing Pgpool-II.

2.11.1. Firewalls

When Pgpool-II connects to other Pgpool-II servers or PostgreSQL servers, the target port must be accessible by enabling firewall management softwares.

First, allow to access port that Pgpool-II use.

```
firewall-cmd --permanent --zone=public --add-port=9999/tcp --add-port=9898/tcp --add-port=9000/tcp --add-port=9694/tcp
firewall-cmd --permanent --zone=public --add-port=9999/udp --add-port=9898/udp --add-port=9000/udp --add-port=9694/udp
firewall-cmd --reload
```

Here is an example for CentOS/RHEL7 when access to PostgreSQL is required.

```
firewall-cmd --permanent --zone=public --add-service=postgresql
firewall-cmd --reload
```

"postgresql" is the service name assigned to PostgreSQL. The list of service names can be obtained by:

```
firewall-cmd --get-services
```

Note that you can define your own service name in `/usr/lib/firewalld/services/`.

If PostgreSQL is listening on 11002 port, rather than the standard 5432 port, you can do:

```
firewall-cmd --zone=public --remove-service=postgresql --permanent
firewall-cmd --zone=public --add-port=11002/tcp --permanent
firewall-cmd --reload
```

Chapter 3. Server Setup and Operation

This chapter discusses how to set up and run the Pgpool-II server and its interactions with the operating system.

3.1. The Pgpool-II User Account

As with any server daemon that is accessible to the outside world, it is advisable to run Pgpool-II under a separate user account. This user account should only own the data that is managed by the server, and should not be shared with other daemons. (For example, using the user `nobody` is a bad idea.) It is not advisable to install executables owned by this user because compromised systems could then modify their own binaries.

To add a Unix user account to your system, look for a command `useradd` or `adduser`. The user name `pgpool` is often used, and is assumed throughout this book, but you can use another name if you like.

3.2. Configuring `pcp.conf`

Pgpool-II provides a interface for administrators to perform management operation, such as getting Pgpool-II status or terminating Pgpool-II processes remotely. `pcp.conf` is the user/password file used for authentication by this interface. All operation modes require the `pcp.conf` file to be set. A `$prefix/etc/pcp.conf.sample` file is created during the installation of Pgpool-II. Copy the file as `$prefix/etc/pcp.conf` and add your user name and password to it.

```
$ cp $prefix/etc/pcp.conf.sample $prefix/etc/pcp.conf
```

An empty line or a line starting with `#` is treated as a comment and will be ignored. A user name and its associated password must be written as one line using the following format:

```
username:[md5 encrypted password]
```

[md5 encrypted password] can be produced with the `$prefix/bin/pg_md5` command.

```
$ pg_md5 your_password
1060b7b46a3bd36b3a0d66e0127d0517
```

If you don't want pass the password as the argument, execute `pg_md5 -p`.

```
$ pg_md5 -p
password: your_password
```

The `pcp.conf` file must be readable by the user who executes Pgpool-II.

3.3. Configuring Pgpool-II

3.3.1. Configuring pgpool.conf

`pgpool.conf` is the main configuration file of Pgpool-II. You need to specify the path to the file when starting Pgpool-II using `-f` option. `pgpool.conf` is located at `$prefix/etc/pgpool.conf` by default, if it installed from source code.

For each Pgpool-II operation mode, there are sample configurations.

Table 3-1. pgpool.conf samples

Operation mode	Configuration file name
Streaming replication mode	
Replication mode	<code>pgpool.conf.sample-replication</code>
Master slave mode	
Raw mode	<code>pgpool.conf.sample</code>
Logical replication mode	<code>pgpool.conf.sample-logical</code>

These configuration files are located at `/usr/local/etc` with default installation from source code. You can copy one of them as `pgpool.conf`. (probably you need root privilege for this)

```
# cd /usr/local/etc
# cp pgpool.conf.sample-stream pgpool.conf
```

3.3.2. Running mode of Pgpool-II

There are four different running modes in Pgpool-II: streaming replication mode, logical replication mode, master slave mode (slony mode), native replication mode and raw mode. In any mode, Pgpool-II provides connection pooling, automatic fail over and online recovery.

Those modes are exclusive each other and cannot be changed after starting the server. You should make a decision which to use in the early stage of designing the system. If you are not sure, it is recommended to use the streaming replication mode.

The **streaming replication mode** can be used with PostgreSQL servers operating streaming replication. In this mode, PostgreSQL is responsible for synchronizing databases. This mode is widely used and most recommended way to use Pgpool-II. Load balancing is possible in the mode.

The **logical replication mode** can be used with PostgreSQL servers operating logical replication. In this mode, PostgreSQL is responsible for synchronizing tables. Load balancing is possible in the mode. Since logical replication does not replicate all tables, it's user's responsibility to replicate the table which could be load balanced. Pgpool-II load balances all tables. This means that if a table is not replicated, Pgpool-II may lookup outdated tables in the subscriber side.

The **master slave mode** mode (slony mode) can be used with PostgreSQL servers operating Slony. In this mode, Slony/PostgreSQL is responsible for synchronizing databases. Since Slony-I is being obsoleted by streaming replication, we do not recommend to use this mode unless you have specific reason to use Slony. Load balancing is possible in the mode.

In the **native replication mode**, Pgpool-II is responsible for synchronizing databases. The advantage for the mode is the synchronization is done in synchronous way: writing to the database does not return until all of PostgreSQL servers finish the write operation. However, you could get a similar effect using PostgreSQL 9.6 or later with `synchronous_commit = remote_apply` being set in streaming replication. If you could use the setting, we strongly recommend to use it instead of native replication mode because you can avoid some [restrictions](#) in the native replication mode. Since PostgreSQL does not provide cross node snapshot control, it is possible that session X can see data on node A committed by session Y before session Y commits data on node B. If session X tries to update data on node B based on the data seen on node A, then data consistency between node A and B might be lost. To avoid the problem, user need to issue an explicit lock on the data. This is another reason why we recommend to use streaming replication mode with `synchronous_commit = remote_apply`.

Load balancing is possible in the mode.

In the **raw mode**, Pgpool-II does not care about the database synchronization. It's user's responsibility to make the whole system does a meaningful thing. Load balancing is **not** possible in the mode.

3.4. Configuring backend information

For Pgpool-II to recognize PostgreSQL backend servers, you need to configure `backend*` in `pgpool.conf`. For starters, at least `backend_hostname` and `backend_port` parameters are required to be set up to start Pgpool-II server.

3.4.1. Backend Settings

Backend PostgreSQL used by Pgpool-II must be specified in `pgpool.conf`. See [Section 5.4](#)

Chapter 4. Watchdog

4.1. Introduction

Watchdog is a sub process of Pgpool-II to add high availability. Watchdog is used to resolve the single point of failure by coordinating multiple Pgpool-II nodes. The watchdog was first introduced in Pgpool-II **V3.2** and is significantly enhanced in Pgpool-II **V3.5**, to ensure the presence of a quorum at all time. This new addition to watchdog makes it more fault tolerant and robust in handling and guarding against the split-brain syndrome and network partitioning. In addition, **V3.7** introduced quorum failover (see [Section 5.14.6](#)) to reduce the false positives of PostgreSQL server failures. **To ensure the quorum mechanism properly works, the number of Pgpool-II nodes must be odd in number and greater than or equal to 3.**

4.1.1. Coordinating multiple Pgpool-II nodes

Watchdog coordinates multiple Pgpool-II nodes by exchanging information with each other.

At the startup, if the watchdog is enabled, Pgpool-II node sync the status of all configured backend nodes from the master watchdog node. And if the node goes on to become a master node itself it initializes the backend status locally. When a backend node status changes by failover etc., watchdog notifies the information to other Pgpool-II nodes and synchronizes them. When online recovery occurs, watchdog restricts client connections to other Pgpool-II nodes for avoiding inconsistency between backends.

Watchdog also coordinates with all connected Pgpool-II nodes to ensure that failback, failover and

follow_master commands must be executed only on one Pgpool-II node.

4.1.2. Life checking of other Pgpool-II nodes

Watchdog lifecheck is the sub-component of watchdog to monitor the health of Pgpool-II nodes participating in the watchdog cluster to provide the high availability. Traditionally Pgpool-II watchdog provides two methods of remote node health checking. "heartbeat" and "query" mode. The watchdog in Pgpool-II **V3.5** adds a new "external" to [wd_lifecheck_method](#), which enables to hook an external third party health checking system with Pgpool-II watchdog.

Apart from remote node health checking watchdog lifecheck can also check the health of node it is installed on by monitoring the connection to upstream servers. If the monitoring fails, watchdog treats it as the local Pgpool-II node failure.

In *heartbeat* mode, watchdog monitors other Pgpool-II processes by using *heartbeat* signal. Watchdog receives heartbeat signals sent by other Pgpool-II periodically. If there is no signal for a certain period, watchdog regards this as the failure of the Pgpool-II. For redundancy you can use multiple network connections for heartbeat exchange between Pgpool-II nodes. This is the default and recommended mode to be used for health checking.

In *query* mode, watchdog monitors Pgpool-II service rather than process. In this mode watchdog sends queries to other Pgpool-II and checks the response.

Note: Note that this method requires connections from other Pgpool-II, so it would fail monitoring if the [num_init_children](#) parameter isn't large enough. This mode is deprecated and left for backward compatibility.

external mode is introduced by Pgpool-II **V3.5**. This mode basically disables the built in lifecheck of Pgpool-II watchdog and expects that the external system will inform the watchdog about health of local and all remote nodes participating in the watchdog cluster.

4.1.3. Consistency of configuration parameters on all Pgpool-II nodes

At startup watchdog verifies the Pgpool-II configuration of the local node for the consistency with the configurations on the master watchdog node and warns the user of any differences. This eliminates the likelihood of undesired behavior that can happen because of different configuration on different Pgpool-II nodes.

4.1.4. Changing active/standby state when certain fault is detected

When a fault of Pgpool-II is detected, watchdog notifies the other watchdogs of it. If this is the active Pgpool-II, watchdogs decide the new active Pgpool-II by voting and change active/standby state.

4.1.5. Automatic virtual IP switching

When a standby Pgpool-II server promotes to active, the new active server brings up virtual IP interface. Meanwhile, the previous active server brings down the virtual IP interface. This enables the active Pgpool-II to work using the same IP address even when servers are switched.

4.1.6. Automatic registration of a server as a standby in recovery

When the broken server recovers or new server is attached, the watchdog process notifies this to the other watchdogs in the cluster along with the information of the new server, and the watchdog process receives information on the active server and other servers. Then, the attached server is registered as a standby.

4.1.7. Starting/stopping watchdog

The watchdog process starts and stops automatically as sub-processes of the Pgpool-II, therefore there is no dedicated command to start and stop watchdog.

Watchdog controls the virtual IP interface, the commands executed by the watchdog for bringing up and bringing down the VIP require the root privileges. Pgpool-II requires the user running Pgpool-II to have root privileges when the watchdog is enabled along with virtual IP. This is however not good security practice to run the Pgpool-II as root user, the alternative and preferred way is to run the Pgpool-II as normal user and use either the custom commands for [if_up_cmd](#), [if_down_cmd](#), and [arping_cmd](#) using `sudo` or use `setuid` ("set user ID upon execution") on `if_*` commands

Lifeclock process is a sub-component of watchdog, its job is to monitor the health of Pgpool-II nodes participating in the watchdog cluster. The Lifeclock process is started automatically when the watchdog is configured to use the built-in life-checking, it starts after the watchdog main process initialization is complete. However lifeclock process only kicks in when all configured watchdog nodes join the cluster and becomes active. If some remote node fails before the Lifeclock become active that failure will not get caught by the lifeclock.

4.2. Integrating external lifeclock with watchdog

Pgpool-II watchdog process uses the BSD sockets for communicating with all the Pgpool-II processes and the same BSD socket can also be used by any third party system to provide the lifeclock function for local and remote Pgpool-II watchdog nodes. The BSD socket file name for IPC is constructed by appending Pgpool-II `wd_port` after "`s.PGPOOLWD_CMD.`" string and the socket file is placed in the [wd_ipc_socket_dir](#) directory.

4.2.1. Watchdog IPC command packet format

The watchdog IPC command packet consists of three fields. Below table details the message fields and description.

Table 4-1. Watchdog IPC command packet format

Field	Type	Description
TYPE	BYTE1	Command Type
LENGTH	INT32 in network byte order	The length of data to follow
DATA	DATA in JSON format	Command data in JSON format

4.2.2. Watchdog IPC result packet format

The watchdog IPC command result packet consists of three fields. Below table details the message fields and description.

Table 4-2. Watchdog IPC result packet format

Field	Type	Description
TYPE	BYTE1	Command Type
LENGTH	INT32 in network byte order	The length of data to follow
DATA	DATA in JSON format	Command result data in JSON format

4.2.3. Watchdog IPC command packet types

The first byte of the IPC command packet sent to watchdog process and the result returned by watchdog process is identified as the command or command result type. The below table lists all valid types and their meanings

Table 4-3. Watchdog IPC command packet types

Name	Byte Value	Type	Description
REGISTER FOR NOTIFICATIONS	'0'	Command packet	Command to register the current connection to receive watchdog notifications
NODE STATUS CHANGE	'2'	Command packet	Command to inform watchdog about node status change of watchdog node
GET NODES LIST	'3'	Command packet	Command to get the list of all configured watchdog nodes
NODES LIST DATA	'4'	Result packet	The JSON data in packet contains the list of all configured watchdog nodes
CLUSTER IN TRANSITION	'7'	Result packet	Watchdog returns this packet type when it is not possible to process the command because the cluster is transitioning.
RESULT BAD	'8'	Result packet	Watchdog returns this packet type when the IPC command fails
RESULT OK	'9'	Result packet	Watchdog returns this packet type when IPC command succeeds

4.2.4. External lifecheck IPC packets and data

"GET NODES LIST" , "NODES LIST DATA" and "NODE STATUS CHANGE" IPC messages of watchdog can be used to integration an external lifecheck systems. Note that the built-in lifecheck of pgpool also uses the same channel and technique.

4.2.4.1. Getting list of configured watchdog nodes

Any third party lifecheck system can send the "GET NODES LIST" packet on watchdog IPC socket with a JSON data containing the authorization key and value if [wd_authkey](#) is set or empty packet data when [wd_authkey](#) is not configured to get the "NODES LIST DATA" result packet.

The result packet returned by watchdog for the "GET NODES LIST" will contains the list of all configured watchdog nodes to do health check on in the JSON format. The JSON of the watchdog nodes contains the "WatchdogNodes" Array of all watchdog nodes. Each watchdog JSON node contains the "ID", "NodeName", "HostName", "DelegateIP", "WdPort" and "PgpoolPort" for each node.

-- The example JSON data contained in "NODES LIST DATA"

```
{
  "NodeCount":3,
  "WatchdogNodes":
  [
    {
      "ID":0,
      "State":1,
      "NodeName":"Linux_ubuntu_9999",
      "HostName":"watchdog-host1",
      "DelegateIP":"172.16.5.133",
      "WdPort":9000,
      "PgpoolPort":9999
    },
    {
      "ID":1,
      "State":1,
      "NodeName":"Linux_ubuntu_9991",
      "HostName":"watchdog-host2",
      "DelegateIP":"172.16.5.133",
      "WdPort":9000,
      "PgpoolPort":9991
    },
    {
      "ID":2,
      "State":1,
      "NodeName":"Linux_ubuntu_9992",
      "HostName":"watchdog-host3",
      "DelegateIP":"172.16.5.133",
      "WdPort":9000,
      "PgpoolPort":9992
    }
  ]
}
```

-- Note that ID 0 is always reserved for local watchdog node

After getting the configured watchdog nodes information from the watchdog the external lifecycle system can proceed with the health checking of watchdog nodes, and when it detects some status change of any node it can inform that to watchdog using the "NODE STATUS CHANGE" IPC messages of watchdog. The data in the message should contain the JSON with the node ID of the node whose status is changed (The node ID must be same as returned by watchdog for that node in WatchdogNodes list) and the new status of node.

-- The example JSON to inform pgpool-II watchdog about health check failed on node with ID 1 will look like

```
{
  "NodeID":1,
  "NodeStatus":1,
  "Message":"optional message string to log by watchdog for this event"
  "IPCAuthKey":"wd_authkey configuration parameter value"
}
```

-- NodeStatus values meanings are as follows

```
NODE STATUS DEAD = 1
NODE STATUS ALIVE = 2
```

4.3. Restrictions on watchdog

4.3.1. Watchdog restriction with query mode lifecycle

In query mode, when all the DB nodes are detached from a Pgpool-II due to PostgreSQL server failure or

pcp_detach_node issued, watchdog regards that the Pgpool-II service is in the down status and brings the virtual IP assigned to watchdog down. Thus clients of Pgpool-II cannot connect to Pgpool-II using the virtual IP any more. This is necessary to avoid split-brain, that is, situations where there are multiple active Pgpool-II.

4.3.2. Connecting to Pgpool-II whose watchdog status is down

Don't connect to Pgpool-II in down status using the real IP. Because a Pgpool-II in down status can't receive information from other Pgpool-II watchdogs so it's backend status may be different from other the Pgpool-II.

4.3.3. Pgpool-II whose watchdog status is down requires restart

Pgpool-II in down status can't become active nor the standby Pgpool-II. Recovery from down status requires the restart of Pgpool-II.

4.3.4. Watchdog promotion to active takes few seconds

After the active Pgpool-II stops, it will take a few seconds until the standby Pgpool-II promote to new active, to make sure that the former virtual IP is brought down before a down notification packet is sent to other Pgpool-II.

4.4. Architecture of the watchdog

Watchdog is a sub process of Pgpool-II, which adds the high availability and resolves the single point of failure by coordinating multiple Pgpool-II. The watchdog process automatically starts (if enabled) when the Pgpool-II starts up and consists of two main components, Watchdog core and the lifecheck system.

4.4.1. Watchdog Core

Watchdog core referred as a "watchdog" is a Pgpool-II child process that manages all the watchdog related communications with the Pgpool-II nodes present in the cluster and also communicates with the Pgpool-II parent and lifecheck processes.

The heart of a watchdog process is a state machine that starts from its initial state (`WD_LOADING`) and transit towards either standby (`WD_STANDBY`) or master/coordinator (`WD_COORDINATOR`) state. Both standby and master/coordinator states are stable states of the watchdog state machine and the node stays in standby or master/coordinator state until some problem in local Pgpool-II node is detected or a remote Pgpool-II disconnects from the cluster.

The watchdog process performs the following tasks:

- Manages and coordinates the local node watchdog state.
- Interacts with built-in or external lifecheck system for the of local and remote Pgpool-II node health checking.
- Interacts with Pgpool-II main process and provides the mechanism to Pgpool-II parent process for executing the cluster commands over the watchdog channel.
- Communicates with all the participating Pgpool-II nodes to coordinate the selection of master/coordinator node and to ensure the quorum in the cluster.
- Manages the Virtual-IP on the active/coordinator node and allow the users to provide custom scripts for escalation and de-escalation.
- Verifies the consistency of Pgpool-II configurations across the participating Pgpool-II nodes in the watchdog cluster.
- Synchronize the status of all PostgreSQL backends at startup.
- Provides the distributed locking facility to Pgpool-II main process for synchronizing the different failover

commands.

4.4.1.1. Communication with other nodes in the Cluster

Watchdog uses TCP/IP sockets for all the communication with other nodes. Each watchdog node can have two sockets opened with each node. One is the outgoing (client) socket which this node creates and initiate the connection to the remote node and the second socket is the one which is listening socket for inbound connection initiated by remote watchdog node. As soon as the socket connection to remote node succeeds watchdog sends the ADD NODE (`WD_ADD_NODE_MESSAGE`) message on that socket. And upon receiving the ADD NODE message the watchdog node verifies the node information encapsulated in the message with the Pgpool-II configurations for that node, and if the node passes the verification test it is added to the cluster otherwise the connection is dropped.

4.4.1.2. IPC and data format

Watchdog process exposes a UNIX domain socket for IPC communications, which accepts and provides the data in JSON format. All the internal Pgpool-II processes, including Pgpool-II's built-in lifecheck and Pgpool-II main process uses this IPC socket interface to interact with the watchdog. This IPC socket can also be used by any external/3rd party system to interact with watchdog.

See [Section 4.2](#) for details on how to use watchdog IPC interface for integrating external/3rd party systems.

4.4.2. Watchdog Lifecheck

Watchdog lifecheck is the sub-component of watchdog that monitors the health of Pgpool-II nodes participating in the watchdog cluster. Pgpool-II watchdog provides three built-in methods of remote node health checking, "heartbeat", "query" and "external" mode.

In "heartbeat" mode, The lifecheck process sends and receives the data over UDP socket to check the availability of remote nodes and for each node the parent lifecheck process spawns two child process one for sending the heartbeat signal and another for receiving the heartbeat. While in "query" mode, The lifecheck process uses the PostgreSQL libpq interface for querying the remote Pgpool-II. And in this mode the lifecheck process creates a new thread for each health check query which gets destroyed as soon as the query finishes. While in "external" mode, this mode disables the built in lifecheck of Pgpool-II, and expects that the external system will monitor local and remote node instead.

Apart from remote node health checking watchdog lifecheck can also check the health of node it is installed on by monitoring the connection to upstream servers. For monitoring the connectivity to the upstream server Pgpool-II lifecheck uses `execv()` function to executes 'ping -q -c3 hostname' command. So a new child process gets spawned for executing each ping command. This means for each health check cycle a child process gets created and destroyed for each configured upstream server. For example, if two upstream servers are configured in the lifecheck and it is asked to health check at ten second intervals, then after each ten second lifecheck will spawn two child processes, one for each upstream server, and each process will live until the ping command is finished.

Chapter 5. Server Configuration

There are many configuration parameters that affect the behavior of Pgpool-II. In the first section of this chapter we describe how to interact with configuration parameters. The subsequent sections discuss each parameter in detail.

5.1. Setting Parameters

5.1.1. Parameter Names and Values

All parameter names are case-insensitive. Every parameter takes a value of one of five types: boolean, string, integer, floating point, or enumerated (enum). The type determines the syntax for setting the parameter:

- **Boolean:** Values can be written as `on`, `off`, `true`, `false`, `yes`, `no`, `1`, `0` (all case-insensitive) or any unambiguous

prefix of one of these.

- **String:** In general, enclose the value in single quotes, doubling any single quotes within the value. Quotes can usually be omitted if the value is a simple number or identifier, however.
- **Numeric (integer and floating point):** A decimal point is permitted only for floating-point parameters. Do not use thousands separators. Quotes are not required.
- **Enumerated:** Enumerated-type parameters are written in the same way as string parameters, but are restricted to have one of a limited set of values. Enum parameter values are case-insensitive.

5.1.2. Parameter Interaction via the Configuration File

The most fundamental way to set these parameters is to edit the file `pgpool.conf`, which is located in `$prefix/etc/pgpool.conf`, if it installed from source code. An example of what this file might look like is:

```
# This is a comment
listen_addresses = 'localhost'
port = 9999
serialize_accept = off
reset_query_list = 'ABORT; DISCARD ALL'
```

One parameter is specified per line. The equal sign between name and value is optional. Whitespace is insignificant (except within a quoted parameter value) and blank lines are ignored. Hash marks (`#`) designate the remainder of the line as a comment. Parameter values that are not simple identifiers or numbers must be single-quoted. To embed a single quote in a parameter value, write either two quotes (preferred) or backslash-quote.

Parameters set in this way provide default values for the cluster. The settings seen by active sessions will be these values unless they are overridden. The following sections describe ways in which the administrator or user can override these defaults.

The configuration file is reread whenever the main server process receives a `SIGHUP` signal; this signal is most easily sent by running `pgpool reload` from the command line. The main `pgpool` process also propagates this signal to all its child processes, so that existing sessions also adopt the new values. Some parameters can only be set at server start; any changes to their entries in the configuration file will be ignored until the server is restarted. Invalid parameter settings in the configuration file are likewise ignored (but logged) during `SIGHUP` processing.

5.1.3. Parameter Interaction via SQL Clients

Pgpool-II also provides two SQL style commands to interact with session-local configuration settings.

- The [PGPOOL SHOW](#) command allows inspection of the current value of all parameters.
- The [PGPOOL SET](#) command allows modification of the current value of those parameters that can be set locally to a session; it has no effect on other sessions.

5.2. Connections and Authentication

5.2.1. Connection Settings

`listen_addresses` (string)

Specifies the hostname or IP address, on which Pgpool-II will accept TCP/IP connections. `*` accepts all incoming connections. `"` disables TCP/IP connections. Default is `'localhost'`. Connections via UNIX domain socket are always accepted.

This parameter can only be set at server start.

`port` (integer)

The port number used by Pgpool-II to listen for connections. Default is 9999.

This parameter can only be set at server start.

socket_dir (string)

The directory where the UNIX domain socket accepting connections for Pgpool-II will be created. Default is `/tmp`. Be aware that this socket might be deleted by a cron job. We recommend to set this value to `/var/run` or such directory.

This parameter can only be set at server start.

pcp_listen_addresses (string)

Specifies the hostname or IP address, on which pcp process will accept TCP/IP connections. `*` accepts all incoming connections. `""` disables TCP/IP connections. Default is `*`. Connections via UNIX domain socket are always accepted.

This parameter can only be set at server start.

pcp_port (integer)

The port number used by PCP process to listen for connections. Default is 9898.

This parameter can only be set at server start.

pcp_socket_dir (string)

The directory where the UNIX domain socket accepting connections for PCP process will be created. Default is `/tmp`. Be aware that this socket might be deleted by a cron job. We recommend to set this value to `/var/run` or such directory.

This parameter can only be set at server start.

num_init_children (integer)

The number of preforked Pgpool-II server processes. Default is 32. `num_init_children` is also the concurrent connections limit to Pgpool-II from clients. If more than `num_init_children` clients try to connect to Pgpool-II, **they are blocked (not rejected with an error, like PostgreSQL) until a connection to any Pgpool-II process is closed** unless [reserved_connections](#) is set to 1 or more. Up to [listen_backlog_multiplier](#)* `num_init_children` can be queued.

The queue is inside the kernel called "listen queue". The length of the listen queue is called "backlog". There is an upper limit of the backlog in some systems, and if `num_init_children`*[listen_backlog_multiplier](#) exceeds the number, you need to set the backlog higher. Otherwise, following problems may occur in heavy loaded systems:

- connecting to Pgpool-II fails
- connecting to Pgpool-II is getting slow because of retries in the kernel.

You can check if the listen queue is actually overflowed by using "netstat -s" command. If you find something like:

```
535 times the listen queue of a socket overflowed
```

then the listen queue is definitely overflowed. You should increase the backlog in this case (you will be required a super user privilege).

```
# sysctl net.core.somaxconn
net.core.somaxconn = 128
# sysctl -w net.core.somaxconn = 256
```

You could add following to `/etc/sysctl.conf` instead.

```
net.core.somaxconn = 256
```

Number of connections to each PostgreSQL is roughly $\text{max_pool} * \text{num_init_children}$.

However, canceling a query creates another connection to the backend; thus, a query cannot be canceled if all the connections are in use. If you want to ensure that queries can be canceled, set this value to twice the expected connections.

In addition, PostgreSQL allows concurrent connections for non superusers up to $\text{max_connections} - \text{superuser_reserved_connections}$.

In summary, max_pool , num_init_children , max_connections , $\text{superuser_reserved_connections}$ must satisfy the following formula:

```
max_pool*num_init_children <= (max_connections - superuser_reserved_connections) (no query canceling needed)
max_pool*num_init_children*2 <= (max_connections - superuser_reserved_connections) (query canceling needed)
```

This parameter can only be set at server start.

`reserved_connections` (integer)

When this parameter is set to 1 or greater, incoming connections from clients are not accepted with error message "Sorry, too many clients already", rather than blocked if the number of current connections from clients is more than $(\text{num_init_children} - \text{reserved_connections})$. For example, if $\text{reserved_connections} = 1$ and $\text{num_init_children} = 32$, then the 32th connection from a client will be refused. This behavior is similar to PostgreSQL and good for systems on which the number of connections from clients is large and each session may take long time. In this case length of the listen queue could be very long and may cause the system unstable. In this situation setting this parameter to non 0 is a good idea to prevent the listen queue becomes very long.

If this parameter is set to 0, no connection from clients will be refused. The default value is 0. This parameter can only be set at server start.

5.2.2. Authentication Settings

`enable_pool_hba` (boolean)

If `true`, Pgpool-II will use the `pool_hba.conf` for the client authentication. See [Section 6.1](#) for details on how to configure `pool_hba.conf` for client authentication. Default is `false`.

This parameter can be changed by reloading the Pgpool-II configurations.

`pool_passwd` (string)

Specify the path (absolute or relative) to password file for authentication. Default value is `"pool_passwd"`. A relative path will be interpreted with respect to the directory where configuration file is placed. Specifying `"` (empty) disables the use of password file.

Passwords can be stored in the `pool_passwd` file using three formats. AES256 encrypted format, plain text format and md5 format. Pgpool-II identifies the password format type by it's prefix, so each password entry in the `pool_passwd` must be prefixed as per the password format.

To store the password in the plain text format use `TEXT` prefix. For example. to store clear text password string `"mypassword"` in the `pool_passwd`, prepend the password string with `TEXT` prefix. e.g. `TEXTmypassword`

similarly md5 hashed passwords must be prefixed with `md5` and AES256 encrypted password types can be stored using `AES` prefix. see [Section 6.4](#) for more details on using AES256 encrypted passwords.

In the absence of a valid prefix, Pgpool-II will be considered the string as a plain text password.

This parameter can only be set at server start.

`allow_clear_text_frontend_auth` (boolean)

If PostgreSQL backend servers require `md5` or `SCRAM` authentication for some user's authentication but the password for that user is not present in the `"pool_passwd"` file, then enabling `allow_clear_text_frontend_auth` will allow the Pgpool-II to use clear-text-password authentication with frontend clients to get the password in plain text form from the client and use it for backend authentication.

Default is `false`.

This parameter can be changed by reloading the Pgpool-II configurations.

Note: `allow_clear_text_frontend_auth` only works when `enable_pool_hba` is not enabled

`authentication_timeout` (integer)

Specify the timeout in seconds for Pgpool-II authentication. Specifying 0 disables the time out. Default value is 60.

This parameter can be changed by reloading the Pgpool-II configurations.

5.3. Running mode

5.3.1. Master slave mode

This mode is used to couple Pgpool-II with another master/slave replication software (like Slony-I and Streaming replication), that is responsible for doing the actual data replication.

Note: The number of slave nodes are not limited to 1 and Pgpool-II can have up to 127 slave nodes. master/slave mode can also work just master node without any slave nodes.

Load balancing (see [Section 5.7](#)) can also be used with master/slave mode to distribute the read load on the standby backend nodes.

Following options are required to be specified for master/slave mode.

`master_slave_mode` (boolean)

Setting to on enables the master/slave mode. Default is off.

Note: `master_slave_mode` and `replication_mode` are mutually exclusive and only one can be enabled at a time.

This parameter can only be set at server start.

`master_slave_sub_mode` (enum)

Specifies the external replication system used for data replication between PostgreSQL nodes. Below table contains the list of valid values for the parameter.

Table 5-1. master slave sub mode options

Value	Description
-------	-------------

Value	Description
'stream'	Suitable for PostgreSQL's built-in replication system (Streaming Replication)
'slony'	Suitable for Slony-I
'logical'	Suitable for PostgreSQL's built-in replication system (Logical Replication)

Default is 'stream'.

This parameter can only be set at server start.

5.3.2. Replication mode

This mode makes the Pgpool-II to replicate data between PostgreSQL backends.

Load balancing (see [Section 5.7](#)) can also be used with replication mode to distribute the load to the attached backend nodes.

Following options affect the behavior of Pgpool-II in the replication mode.

replication_mode (boolean)

Setting to on enables the replication mode. Default is off.

Note: [replication_mode](#) and [master_slave_mode](#) are mutually exclusive and only one can be enabled at a time.

This parameter can only be set at server start.

replication_stop_on_mismatch (boolean)

When set to on, and all nodes do not reply with the same packet kind to the query that was sent to all PostgreSQL backend nodes, then the backend node whose reply differs from the majority is degenerated by the Pgpool-II. If [replication_stop_on_mismatch](#) is set to off and a similar situation happens then the Pgpool-II only terminates the current user session but does not degenerate a backend node.

Note: Pgpool-II does not examine the data returned by the backends and takes the decision only by comparing the result packet types.

A typical use case of enabling the [replication_stop_on_mismatch](#) is to guard against the data inconsistency among the backend nodes. For example, you may want to degenerate a backend node if an UPDATE statement fails on one backend node while passes on others.

Default is off.

This parameter can be changed by reloading the Pgpool-II configurations.

failover_if_affected_tuples_mismatch (boolean)

When set to on, and all nodes do not reply with the same number of affected tuples to the INSERT/UPDATE/DELETE query, then the backend node whose reply differs from the majority is degenerated by the Pgpool-II. If [failover_if_affected_tuples_mismatch](#) is set to off and a similar situation happens then the Pgpool-II only terminates the current user session but does not degenerate a backend node.

Note: In case of a tie, when two or more groups have the same number of nodes, then the group containing the master node (backend node having the youngest node id) gets the precedence.

Default is off.

This parameter can be changed by reloading the Pgpool-II configurations.

replicate_select (boolean)

When set to on, Pgpool-II enables the SELECT query replication mode. i.e. The SELECT queries are sent to all backend nodes.

Table 5-2. replicate_select with load_balance_mode affects on SELECT routing

replicate_select is true	Y	N			
load_balance_mode is true	ANY	Y		N	
SELECT is inside a transaction block	ANY	Y	N	ANY	ANY
Transaction isolation level is SERIALIZABLE and the transaction has issued a write query	ANY	Y	N	ANY	ANY
results(R:replication, M: send only to master, L: load balance)	R	M	L	L	M

Default is off.

This parameter can be changed by reloading the Pgpool-II configurations.

insert_lock (boolean)

When set to on, Pgpool-II will automatically lock the table on PostgreSQL before an INSERT statement is issued for that.

When replicating a table with SERIAL data type, the SERIAL column value may get different values on the different backends. The workaround to this problem is to explicitly lock the table before issuing the INSERT.

So for automatically locking the table Pgpool-II do the following transformation:

```
INSERT INTO ...
```

to

```
BEGIN;
LOCK TABLE ...
INSERT INTO ...
COMMIT;
```

Caution

This approach severely degrades the transactions' parallelism

Pgpool-II **V2.2** or later, automatically detects whether the table has a SERIAL columns or not, so it never locks the table if it doesn't have the SERIAL columns.

Pgpool-II **V3.0** until Pgpool-II **V3.0.4** uses a row lock against the sequence relation, rather than table lock. This is intended to minimize lock conflict with VACUUM (including autovacuum). However this can lead to another problem. After transaction wraparound happens, row locking against the sequence

relation causes PostgreSQL internal error (more precisely, access error on `pg_clog`, which keeps transaction status). To prevent this, PostgreSQL core developers decided to disallow row locking against sequences and this broke the Pgpool-II, of course (the "fixed" version of PostgreSQL was released as 9.0.5, 8.4.9, 8.3.16 and 8.2.22).

Pgpool-II **V3.0.5** or later uses a row lock against `pgpool_catalog.insert_lock` table because new PostgreSQL disallows a row lock against the sequence relation. So creating `insert_lock` table in all databases which are accessed via Pgpool-II beforehand is required. See [Section 2.8](#) for more details. If does not exist `insert_lock` table, Pgpool-II locks the insert target table. This behavior is same as Pgpool-II **V2.2** and **V2.3** series.

If you want to use `insert_lock` which is compatible with older releases, you can specify lock method by configure script. See [Section 2.5](#) for more details.

For fine (per statement) control:

- set `insert_lock` to true, and add `/*NO INSERT LOCK*/` at the beginning of an INSERT statement for which you do not want to acquire the table lock.
- set `insert_lock` to false, and add `/*INSERT LOCK*/` at the beginning of an INSERT statement for which you want to acquire the table lock.

Note: If `insert_lock` is enabled, the regression tests for PostgreSQL 8.0 gets fail in transactions, privileges, rules, and `alter_table`.

The reason for this is that Pgpool-II tries to LOCK the VIEW for the rule test, and it produces the below error message:

```
! ERROR: current transaction is aborted, commands ignored until
end of transaction block
```

For example, the transactions test tries an INSERT into a table which does not exist, and Pgpool-II causes PostgreSQL to acquire the lock for the table. Of cause this results in an error. The transaction will be aborted, and the following INSERT statement produces the above error message.

Default is off.

This parameter can be changed by reloading the Pgpool-II configurations.

`lobj_lock_table` (string)

Specifies a table name used for large object replication control. If it is specified, Pgpool-II will lock the table specified by `lobj_lock_table` and generate a large object id by looking into `pg_largeobject` system catalog and then call `lo_create` to create the large object. This procedure guarantees that Pgpool-II will get the same large object id in all DB nodes in replication mode.

Note: PostgreSQL 8.0 and older does not have `lo_create`, so this feature does not work with PostgreSQL 8.0 and older versions.

A call to the `libpq` function `lo_creat()` triggers this feature. Also large object creation through Java API (JDBC driver), PHP API (`pg_lo_create`, or similar API in PHP library such as PDO), and this same API in various programming languages are known to use a similar protocol, and thus should work.

This feature does not works with following operations on large objects.

- All APIs using `lo_create`, `lo_import_with_oid`.
- `lo_import` function in backend called in SELECT.
- `lo_create` function in backend called in SELECT.

Note: All PostgreSQL users must have a write access on `lobj_lock_table` and it can be created in any schema.

Example to create a large object lock table:

```
CREATE TABLE public.my_lock_table ();
GRANT ALL ON public.my_lock_table TO PUBLIC;
```

Default is `''`(empty), which disables the feature.

5.4. Backend Settings

5.4.1. Backend Connection Settings

`backend_hostname` (string)

`backend_hostname` specifies the PostgreSQL backend to be connected to. It is used by Pgpool-II to communicate with the server.

For TCP/IP communication, this parameter can take a hostname or an IP address. If this begins with a slash(/), it specifies Unix-domain communication rather than TCP/IP; the value is the name of the directory in which the socket file is stored. The default behavior when `backend_hostname` is empty (`''`) is to connect to a Unix-domain socket in `/tmp`.

Multiple backends can be specified by adding a number at the end of the parameter name (e.g. `backend_hostname0`). This number is referred to as "DB node ID", and it starts from 0. The backend which was given the DB node ID of 0 will be called "master node". When multiple backends are defined, the service can be continued even if the master node is down (not true in some modes). In this case, the youngest DB node ID alive will be the new master node.

Please note that the DB node which has id 0 has no special meaning if operated in streaming replication mode. Rather, you should care about if the DB node is the "primary node" or not. See [Section 5.7](#), [Section 5.9](#), [Section 5.11](#) for more details.

If you plan to use only one PostgreSQL server, specify it by `backend_hostname0`.

New nodes can be added by adding parameter rows and reloading a configuration file. However, the existing values cannot be updated, so you must restart Pgpool-II in that case.

`backend_port` (integer)

`backend_port` specifies the port number of the backends. Multiple backends can be specified by adding a number at the end of the parameter name (e.g. `backend_port0`). If you plan to use only one PostgreSQL server, specify it by `backend_port0`.

New backend ports can be added by adding parameter rows and reloading a configuration file. However, the existing values cannot be updated, so you must restart Pgpool-II in that case.

`backend_weight` (floating point)

`backend_weight` specifies the load balance ratio of the backends. It may be set to any integer or floating point value greater than or equal to zero. Multiple backends can be specified by adding a number at the end of the parameter name (e.g. `backend_weight0`). If you plan to use only one PostgreSQL server, specify it by `backend_weight0`.

New `backend_weight` can be added in this parameter by reloading a configuration file. However, this will take effect only for new established client sessions. Pgpool-II **V2.2.6, V2.3** or later allows updating the values by reloading a configuration file. This is useful if you want to prevent any query sent to slaves to perform some administrative work in master/slave mode.

5.4.2. Backend Data Settings

`backend_data_directory` (string)

`backend_data_directory` specifies the database cluster directory of the backend. Multiple backends can be specified by adding a number at the end of the parameter name (e.g. `backend_data_directory0`). If you plan to use only one PostgreSQL server, specify it by `backend_data_directory0`.

New `backend_data_directory` can be added by adding parameter rows and reloading a configuration file. However, the existing values cannot be updated, so you must restart Pgpool-II in that case.

`backend_flag` (string)

`backend_flag` controls various backend behavior. Multiple backends can be specified by adding a number at the end of the parameter name (e.g. `backend_flag0`). If you plan to use only one PostgreSQL server, specify it by `backend_flag0`.

New backend flags can be added by adding parameter rows and reloading a configuration file. Currently followings are allowed. Multiple flags can be specified by using "|".

Table 5-3. Backend flags

Flag	Description
ALLOW_TO_FAILOVER	Allow to failover or detaching backend. This is the default. You cannot specify with DISALLOW_TO_FAILOVER at a same time.
DISALLOW_TO_FAILOVER	Disallow to failover or detaching backend This is useful when you protect backend by using HA (High Availability) softwares such as Heartbeat or Pacemaker. You cannot specify with ALLOW_TO_FAILOVER at a same time.
ALWAYS_MASTER	This is only useful in streaming replication mode. See Section 3.3.2 about streaming replication mode. If this flag is set to one of backends, Pgpool-II will not find the primary node by inspecting backend. Instead, always regard the node which the flag is set as the primary node. This is useful for systems including Amazon Aurora for PostgreSQL Compatibility which has fixed master server name. See Section 8.5 for an example settings.

This parameter can be changed by reloading the Pgpool-II configurations.

`backend_application_name` (string)

`backend_application_name` specifies the application name of walsender, which receives WAL log from primary node. Thus in other than streaming replication mode, this parameter does not need to be set. Also this parameter is required to if you want to show "replication_state" and "replication_sync_state" column in [SHOW POOL NODES](#) command.

This parameter can be changed by reloading the Pgpool-II configurations.

5.5. Connection Pooling

Pgpool-II maintains established connections to the PostgreSQL servers, and reuses them whenever a new connection with the same properties (i.e. user name, database, protocol version) comes in. It reduces the connection overhead, and improves system's overall throughput.

5.5.1. Connection Pooling Settings

`connection_cache` (boolean)

Caches connections to backends when set to on. Default is on. However, connections to ...

Caches connections to backends when set to on. Default is on. **however, connections to template0, template1, postgres and regression databases are not cached even if connection_cache is on.**

You need to restart Pgpool-II if you change this value.

max_pool (integer)

The maximum number of cached connections in each Pgpool-II child process. Pgpool-II reuses the cached connection if an incoming connection is connecting to the same database with the same user name and the same run-time parameters. If not, Pgpool-II creates a new connection to the backend. If the number of cached connections exceeds max_pool, the oldest connection will be discarded, and uses that slot for the new connection.

Default value is 4. Please be aware that the number of connections from Pgpool-II processes to the backends may reach num_init_children * max_pool in total.

This parameter can only be set at server start.

listen_backlog_multiplier (integer)

Specifies the length of connection queue from frontend to Pgpool-II. The queue length (actually "backlog" parameter of listen() system call) is defined as listen_backlog_multiplier * [num_init_children](#).

Note: Some systems have the upper limit of the backlog parameter of listen() system call. See [num_init_children](#) for more details.

Default is 2.

This parameter can only be set at server start.

serialize_accept (boolean)

When set to on, Pgpool-II enables the serialization on incoming client connections. Without serialization the OS kernel wakes up all of the Pgpool-II children processes to execute accept() and one of them actually gets the incoming connection. The problem here is, because so many child processes wake up at a same time, heavy context switching occurs and the performance is affected.

This phenomena is a well known classic problem called "the thundering herd problem". This can be solved by the serialization of the accept() calls, so that only one Pgpool-II process gets woken up for incoming connection to execute the accept() .

But serialization has its own overheads, and it is recommended to be used only with the larger values of [num_init_children](#). For the small number of [num_init_children](#), the serialize accept can degrade the performance because of serializing overhead.

Note: It is recommended to do a benchmark before deciding whether to use serialize_accept or not, because the correlation of [num_init_children](#) and serialize_accept can be different on different environments.

Example 5-1. Using pgbench to decide if serialize_accept should be used

To run the pgbench use the following command.

```
pgbench -n -S -p 9999 -c 32 -C -S -T 300 test
```

Here, -C tells pgbench to connect to database each time a transaction gets executed. -c 32 specifies the number of the concurrent sessions to Pgpool-II. You should change this according to your system's

requirement. After `pgbench` finishes, check the number from "including connections establishing".

Note: When `child_life_time` is enabled, `serialize_accept` has no effect. Make sure that you set `child_life_time` to 0 if you intend to turn on the `serialize_accept`. And if you are worried about Pgpool-II process memory leaks or whatever potential issue, you could use `child_max_connections` instead. This is purely an implementation limitation and may be removed in the future.

Default is off.

This parameter can only be set at server start.

`child_life_time` (integer)

Specifies the time in seconds to terminate a Pgpool-II child process if it remains idle. The new child process is immediately spawned by Pgpool-II when it is terminated because of `child_life_time`. `child_life_time` is a measure to prevent the memory leaks and other unexpected errors in Pgpool-II children.

Note: `child_life_time` does not apply to processes that have not accepted any connection yet.

Note: `serialize_accept` becomes ineffective when `child_life_time` is enabled.

Default is 300 (5 minutes) and setting it to 0 disables the feature.

This parameter can only be set at server start.

`client_idle_limit` (integer)

Specifies the time in seconds to disconnect a client if it remains idle since the last query. This is useful for preventing the Pgpool-II children from being occupied by a lazy clients or broken TCP/IP connection between client and Pgpool-II.

Note: `client_idle_limit` is ignored in the second stage of online recovery.

The default is 0, which turns off the feature.

This parameter can be changed by reloading the Pgpool-II configurations. You can also use `PGPOOL SET` command to alter the value of this parameter for a current session.

`child_max_connections` (integer)

Specifies the lifetime of a Pgpool-II child process in terms of the number of client connections it can receive. Pgpool-II will terminate the child process after it has served `child_max_connections` client connections and will immediately spawn a new child process to take its place.

`child_max_connections` is useful on a very busy server, where `child_life_time` and `connection_life_time` never gets triggered. It is also useful to prevent the PostgreSQL servers from getting too big.

The default is 0, which turns off the feature.

This parameter can only be set at server start.

connection_life_time (integer)

Specifies the time in seconds to terminate the cached connections to the PostgreSQL backend. This serves as the cached connection expiration time.

The default is 0, which means the cached connections will not be disconnected.

This parameter can only be set at server start.

reset_query_list (string)

Specifies the SQL commands to be sent to reset the backend connection when exiting the user session. Multiple commands can be specified by delimiting each by ";".

The available commands differ among PostgreSQL versions. Below are some recommended settings for `reset_query_list` on different PostgreSQL versions. Note, however, that `ABORT` command should be always included.

Table 5-4. Recommended setting for `reset_query_list` on different PostgreSQL versions

PostgreSQL version	reset_query_list
7.1 or earlier	'ABORT'
7.2 to 8.2	'ABORT; RESET ALL; SET SESSION AUTHORIZATION DEFAULT'
8.3 or later	'ABORT; DISCARD ALL'

Note: "ABORT" is not issued when not in a transaction block for 7.4 or later PostgreSQL versions.

Default is 'ABORT; DISCARD ALL'.

This parameter can be changed by reloading the Pgpool-II configurations.

5.6. Error Reporting and Logging

5.6.1. Where To Log

log_destination (string)

Pgpool-II supports two destinations for logging the Pgpool-II messages. The supported log destinations are `stderr` and `syslog`. You can also set this parameter to a list of desired log destinations separated by commas if you want the log messages on the multiple destinations.

```
#for example to log on both syslog and stderr  
log_destination = 'syslog,stderr'
```

The default is to log to `stderr` only.

Note: On some systems you will need to alter the configuration of your system's `syslog` daemon in order to make use of the `syslog` option for `log_destination`. Pgpool-II can log to `syslog` facilities LOCAL0 through LOCAL7 (see [syslog_facility](#)), but the default `syslog`

configuration on most platforms will discard all such messages. You will need to add something like:

```
local0.* /var/log/pgpool.log
```

to the syslog daemon's configuration file to make it work.

This parameter can be changed by reloading the Pgpool-II configurations.

syslog_facility (enum)

See also the documentation of your system's syslog daemon. When logging to syslog is enabled, this parameter determines the syslog "facility" to be used. You can choose from LOCAL0, LOCAL1, LOCAL2, LOCAL3, LOCAL4, LOCAL5, LOCAL6, LOCAL7; the default is LOCAL0. See also the documentation of your system's syslog daemon.

This parameter can be changed by reloading the Pgpool-II configurations.

syslog_ident (string)

When logging to syslog is enabled, this parameter determines the program name used to identify Pgpool-II messages in syslog logs. The default is pgpool.

This parameter can be changed by reloading the Pgpool-II configurations.

5.6.2. When To Log

client_min_messages (enum)

Controls which minimum message levels are sent to the client. Valid values are DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, LOG, NOTICE, WARNING and ERROR. Each level includes all the levels that follow it. The default is NOTICE.

This parameter can be changed by reloading the Pgpool-II configurations. You can also use [PGPOOL SET](#) command to alter the value of this parameter for a current session.

log_min_messages (enum)

The default is WARNING. Controls which minimum message levels are emitted to log. Valid values are DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL, and PANIC. Each level includes all the levels that follow it. The default is WARNING.

This parameter can be changed by reloading the Pgpool-II configurations. You can also use [PGPOOL SET](#) command to alter the value of this parameter for a current session.

5.6.3. What To Log

log_statement (boolean)

Setting to on, prints all SQL statements to the log.

This parameter can be changed by reloading the Pgpool-II configurations. You can also use [PGPOOL SET](#) command to alter the value of this parameter for a current session.

log_per_node_statement (boolean)

Similar to [log_statement](#), except that it print the logs for each DB node separately. It can be useful to make sure that replication or load-balancing is working.

This parameter can be changed by reloading the Pgpool-II configurations. You can also use [PGPOOL SET](#) command to alter the value of this parameter for a current session.

log_client_messages (boolean)

Setting to on, prints client messages to the log.

This parameter can be changed by reloading the Pgpool-II configurations. You can also use [PGPOOL SET](#) command to alter the value of this parameter for a current session.

log_hostname (boolean)

Setting to on, prints the hostname instead of IP address in the `ps` command result, and connection logs (when [log_connections](#) is on).

This parameter can be changed by reloading the Pgpool-II configurations.

log_connections (boolean)

Setting to on, prints all client connections from to the log.

This parameter can be changed by reloading the Pgpool-II configurations.

log_error_verbosity (enum)

Controls the amount of detail emitted for each message that is logged. Valid values are `TERSE`, `DEFAULT`, and `VERBOSE`, each adding more fields to displayed messages. `TERSE` excludes the logging of `DETAIL`, `HINT`, and `CONTEXT` error information.

This parameter can be changed by reloading the Pgpool-II configurations. You can also use [PGPOOL SET](#) command to alter the value of this parameter for a current session.

log_line_prefix (string)

This is a `printf`-style string that is output at the beginning of each log line. `%` characters begin "escape sequences" that are replaced with information outlined below. All unrecognized escapes are ignored. Other characters are copied straight to the log line. Default is `'%t: pid %p: '`, which prints timestamp and process id, which keeps backward compatibility with prePgpool-II **V3.4**.

Table 5-5. log_line_prefix escape options

Escape	Effect
<code>%a</code>	Client application name
<code>%p</code>	Process ID (PID)
<code>%P</code>	Process name
<code>%t</code>	Time stamp
<code>%d</code>	Database name
<code>%u</code>	User name
<code>%l</code>	Log line number for each process
<code>%%</code>	'%' character

This parameter can be changed by reloading the Pgpool-II configurations.

5.7. Load Balancing

Pgpool-II load balancing of `SELECT` queries works with Master Slave mode ([Section 5.3.1](#)) and Replication mode ([Section 5.3.2](#)). When enabled Pgpool-II sends the writing queries to the primary node in Master Slave mode, all of the backend nodes in Replication mode, and other queries get load balanced among all backend nodes. To which node the load balancing mechanism sends read queries is decided at the session start time and will not be changed until the session ends. However there are some exceptions. See below for more details.

Note: Queries which are sent to primary node or replicated because they cannot be

balanced are also accounted for in the load balancing algorithm.

Note: If you don't want a query that qualifies for the load balancing to be load balanced by Pgpool-II, you can put arbitrary comment(***/*NO LOAD BALANCE**** is usually used) before the `SELECT` statement. This will disable the load balance of the particular query and Pgpool-II will send it to the master node (the primary node in Master Slave mode).

Note: You can check which DB node is assigned as the load balancing node by using [SHOW POOL NODES](#).

5.7.1. Condition for Load Balancing

For a query to be load balanced, all the following requirements must be met:

- PostgreSQL version 7.4 or later
- either in replication mode or master slave mode
- the query must not be in an explicitly declared transaction (i.e. not in a `BEGIN ~ END` block)
 - However, if following conditions are met, load balance is possible even if in an explicit transaction
 - transaction isolation level is not `SERIALIZABLE`
 - transaction has not issued a write query yet (until a write query is issued, load balance is possible. Here "write query" means non `SELECT` DML or DDL. **Before Pgpool-II 4.1**, `SELECT`s having write functions as specified in `black` or `white` function list is not regarded as a write query.)
 - If `black` and `white` function list is empty, `SELECT`s having functions is regarded as a read only query.
- it's not `SELECT INTO`
- it's not `SELECT FOR UPDATE` nor `FOR SHARE`
- it starts with "`SELECT`" or one of `COPY TO STDOUT`, `EXPLAIN`, `EXPLAIN ANALYZE SELECT...`
[ignore_leading_white_space = true](#) will ignore leading white space. (Except for `SELECT`s using writing functions specified in [black_function_list](#) or [white_function_list](#))
- in master slave mode, in addition to above, following conditions must be met:
 - does not use temporary tables
 - does not use unlogged tables
 - does not use system catalogs

Note: You could suppress load balancing by inserting arbitrary comments just in front of the `SELECT` query:

```
/*REPLICATION*/ SELECT ...
```

If you want to use comments without suppressing load balancing, you can set [allow_sql_comments](#) to on. Please refer to [replicate_select](#) as well.

Note: The JDBC driver has an autocommit option. If the autocommit is false, the JDBC driver sends "BEGIN" and "COMMIT" by itself. In this case the same restriction above regarding load balancing will be applied.

5.7.2. Writing queries may affect Load Balancing

In general, read queries are load balanced if certain conditions are met. However, writing queries may affect the load balancing. Here "writing queries" mean all the queries except below:

- SELECT/WITH without FOR UPDATE/SHARE
- COPY TO STDOUT
- EXPLAIN
- EXPLAIN ANALYZE and the query is SELECT not including writing functions
- SHOW

If writing queries appear, succeeding read queries may not be load balanced. i.e. sent to primary node (in streaming replication mode) or master node (in other mode) depending on the setting of [disable_load_balance_on_write](#).

5.7.3. Load Balancing in Streaming Replication

While using Streaming replication and Hot Standby, it is important to determine which query can be sent to the primary or the standby, and which one should not be sent to the standby. Pgpool-II's Streaming Replication mode carefully takes care of this.

We distinguish which query should be sent to which node by looking at the query itself.

- These queries should be sent to the primary node only
 - INSERT, UPDATE, DELETE, COPY FROM, TRUNCATE, CREATE, DROP, ALTER, COMMENT
 - SELECT ... FOR SHARE | UPDATE
 - SELECT in transaction isolation level SERIALIZABLE
 - LOCK command more strict than ROW EXCLUSIVE MODE
 - DECLARE, FETCH, CLOSE
 - SHOW
 - Some transactional commands:
 - BEGIN READ WRITE, START TRANSACTION READ WRITE
 - SET TRANSACTION READ WRITE, SET SESSION CHARACTERISTICS AS TRANSACTION READ WRITE
 - SET transaction_read_only = off
 - Two phase commit commands: PREPARE TRANSACTION, COMMIT PREPARED, ROLLBACK PREPARED
 - LISTEN, UNLISTEN, NOTIFY

- VACUUM
- Some sequence functions (nextval and setval)
- Large objects creation commands
- Multi-statement queries (multiple SQL commands on single line)
- These queries can be sent to both the primary node and the standby node. If load balancing is enabled, these types of queries can be sent to the standby node. However, if `delay_threshold` is set and the replication delay is higher than [delay_threshold](#), queries are sent to the primary node.
 - SELECT not listed above
 - COPY TO STDOUT
 - EXPLAIN
 - EXPLAIN ANALYZE and the query is SELECT not including writing functions
 - SHOW
- These queries are sent to both the primary node and the standby node
 - SET
 - DISCARD
 - DEALLOCATE ALL

In an explicit transaction:

- Transaction starting commands such as BEGIN are sent to both the primary node and the standby node.
- Following SELECT and some other queries that can be sent to both primary or standby are executed in the transaction or on the standby node.
- Commands which cannot be executed on the standby such as INSERT are sent to the primary. After one of these commands, even SELECTs are sent to the primary node, This is because these SELECTs might want to see the result of an INSERT immediately. This behavior continues until the transaction closes or aborts.

In the extended protocol, it is possible to determine if the query can be sent to standby or not in load balance mode while parsing the query. The rules are the same as for the non extended protocol. For example, INSERTs are sent to the primary node. Following bind, describe and execute will be sent to the primary node as well.

Note: If the parse of a SELECT statement is sent to the standby node due to load balancing, and then a DML statement, such as an INSERT, is sent to Pgpool-II, then the parsed SELECT will have to be executed on the primary node. Therefore, we re-parse the SELECT on the primary node.

Lastly, queries that Pgpool-II's parser thinks to be an error are sent to the primary node.

5.7.4. Load Balancing Settings

`load_balance_mode` (boolean)

When set to on, Pgpool-II enables the load balancing on incoming SELECT queries. i.e. SELECT queries from the clients gets distributed to the configured PostgreSQL backends. Default is off.

This parameter can only be set at server start.

ignore_leading_white_space (boolean)

When set to on, **Pgpool-II** ignores the white spaces at the beginning of SQL queries in load balancing. It is useful if used with APIs like DBI/DBD:Pg which adds white spaces against the user's intention.

This parameter can be changed by reloading the **Pgpool-II** configurations.

white_function_list (string)

Specifies a comma separated list of function names that **DO NOT** update the database. SELECTs including functions **not specified** in this list are not load balanced. These are replicated among all the DB nodes in Replication mode, sent to the primary node only in Master Slave mode.

You can use regular expression to match function names, to which **^** and **\$** are automatically added.

Example 5-2. Using regular expression

If you have prefixed all your read only function with 'get_' or 'select_', You can set the [white_function_list](#) like below:

```
white_function_list = 'get_*,select_*
```

Note: Schema qualifications can not be used in [white_function_list](#) because **Pgpool-II** silently ignores a schema qualification in function names appearing in an input SQL while comparing the list and the input SQL. As a result, a schema qualified function name in the list never matches function names appearing in the input SQL.

This parameter can be changed by reloading the **Pgpool-II** configurations.

black_function_list (string)

Specifies a comma separated list of function names that **DO** update the database. SELECTs including functions **specified** in this list are not load balanced. These are replicated among all the DB nodes in Replication mode, sent to the primary node only in Master Slave mode.

You can use regular expression to match function names, to which **^** and **\$** are automatically added.

Example 5-3. Using regular expression

If you have prefixed all your updating functions with 'set_', 'update_', 'delete_' or 'insert_', You can set the [black_function_list](#) like below:

```
black_function_list = 'nextval,setval,set_*,update_*,delete_*,insert_*
```

Note: Schema qualifications can not be used in [black_function_list](#) because **Pgpool-II** silently ignores a schema qualification in function names appearing in an input SQL while comparing the list and the input SQL. As a result, a schema qualified function name in the list never matches function names appearing in the input SQL.

Note: [black_function_list](#) and [white_function_list](#) are mutually exclusive and only one of the two lists can be set in the configuration.

Example 5-4. Configuring using `nextval()` and `setval()` to land on proper backend

Prior to Pgpool-II V3.0, `nextval()` and `setval()` were known as functions writing to the database. You can configure this by setting `black_function_list` and `white_function_list` as follows

```
white_function_list = ""
black_function_list = 'nextval,setval,lastval,currval'
```

Note: PostgreSQL also contains `lastval()` and `currval()` in addition to `nextval()` and `setval()`. Though `lastval()` and `currval()` are not writing function type, but it is advised to treat `lastval()` and `currval()` as writing functions to avoid errors which occur when these functions are accidentally load balanced.

This parameter can be changed by reloading the Pgpool-II configurations.

`black_query_pattern_list` (string)

Specifies a semicolon separated list of SQL patterns that should be sent to primary node. SQL that matched patterns specified in this list are not load balanced. Only Master Slave mode is supported.

You can use regular expression to match SQL patterns, to which `^` and `$` are automatically added. When using special characters in regular expressions (such as `"", ";", "*", "(", ")", "|", "+", ".", "\\", "?", "^", "$", "{", "}", "{", " or "}"`, etc.) in SQL patterns, you need to escape them by using `"\"`. SQL pattern specified in this parameter is case-insensitive.

Example 5-5. Using regular expression

If the following SQL should be sent to the primary node only, You can set the `black_query_pattern_list` like below:

- `SELECT * FROM table_name1;`
- `SELECT col1, col2 FROM table_name2 WHERE col1 LIKE '%a%';`
- SQL including `table_name3`

```
black_query_pattern_list = 'SELECT \* FROM table_name1\;;SELECT col1, col2 FROM table_name2 WHERE col1 LIKE \'%a'
```

Note: If SQL matches both `black_function_list` and `white_function_list`, `white_function_list` setting is ignored and the SQL should be sent only to the primary node.

Depending on the SQL patterns, performance might be 1-2% lower when using this feature.

This parameter can be changed by reloading the Pgpool-II configurations.

`database_redirect_preference_list` (string)

Specifies the list of **"database-name:node id(ratio)"** pairs to send `SELECT` queries to a particular backend node for a particular database connection at a specified load balance ratio. The load balance ratio specifies a value between 0 and 1. The default is 1.0.

For example, by specifying `"test:1(0.5)"`, Pgpool-II will redirect 50% `SELECT` queries to the backend node of ID 1 for the connection to `"test"` database. You can specify multiple **"database name:node id"** pair by separating them using comma (,).

Regular expressions are also accepted for database name. You can use special keywords as **node id**. If "**primary**" is specified, queries are sent to the primary node, and if "**standby**" is specified, one of the standby nodes are selected randomly based on weights ([backend_weight](#)).

Example 5-6. Using `database_redirect_preference_list`

If you want to configure the following `SELECT` query routing rules:

- Route all `SELECT` queries on `postgres` database to the primary backend node.
- Route 30% `SELECT` queries on `mydb0` or on `mydb1` databases to backend node of ID. The other 70% `SELECT` queries will be sent to other backend nodes.
- Route all `SELECT` queries on `mydb2` database to standby backend nodes.

then the [database_redirect_preference_list](#) will be configured as follows:

```
database_redirect_preference_list = 'postgres:primary,mydb[01]:1(0.3),mydb2:standby'
```

This parameter can be changed by reloading the Pgpool-II configurations.

`app_name_redirect_preference_list` (string)

Specifies the list of "**application-name:node id(ratio)**" pairs to send `SELECT` queries to a particular backend node for a particular client application connection at a specified load balance ratio.

Note: In PostgreSQL **V9.0** or later the "Application name" is a name specified by a client when it connects to database.

For example, application name of `psql` command is "`psql`".

Note: Pgpool-II recognizes the application name only specified in the start-up packet. Although a client can provide the application name later in the session, but that does not get considered by the Pgpool-II for query routing.

The notion of [app_name_redirect_preference_list](#) is same as the [database_redirect_preference_list](#) thus you can also use the regular expressions for application names. Similarly special keyword "**primary**" indicates the primary node and "**standby**" indicates one of standby nodes. The load balance weight specifies a value between 0 and 1. The default is 1.0.

Example 5-7. Using `app-name_redirect_preference_list`

If you want to configure the following `SELECT` query routing rules:

- Route all `SELECT` from `psql` client to the primary backend node.
- Route 30% `SELECT` queries from `myapp1` client to backend node of ID 1. The other 70% `SELECT` queries will be sent to other backend nodes.
- Route all `SELECT` queries from `myapp2` client to standby backend nodes.

then the [app_name_redirect_preference_list](#) will be configured as follows:

```
app_name_redirect_preference_list = 'psql:primary,myapp1:1(0.3),myapp2:standby'
```


Note: [app_name_redirect_preference_list](#) takes precedence over the [database_redirect_preference_list](#).

For example, if you set `database_redirect_preference_list = 'postgres:standby(1.0)'` and `app_name_redirect_preference_list = 'myapp1:primary(1.0)'`, all `SELECT` from application `myapp1` on postgres database will be sent to primary backend node.

Note: By specifying of [app_name_redirect_preference_list](#) and [database_redirect_preference_list](#), when multiple database names and application names are matched, the first setting will be used.

For example, if you set `database_redirect_preference_list = 'postgres:primary,postgres:standby',` "postgres: primary" will be used.

Caution

JDBC driver PostgreSQL-9.3 and earlier versions does not send the application name in the startup packet even if the application name is specified using the JDBC driver option "ApplicationName" and "assumeMinServerVersion=9.0". So if you want to use the [app_name_redirect_preference_list](#) feature through JDBC, Use PostgreSQL-9.4 or later version of the driver.

This parameter can be changed by reloading the Pgpool-II configurations.

`allow_sql_comments` (boolean)

When set to on, Pgpool-II ignore the SQL comments when identifying if the load balance or query cache is possible on the query. When this parameter is set to off, the SQL comments on the query could effectively prevent the query from being load balanced or cached (pre Pgpool-II **V3.4** behavior).

This parameter can be changed by reloading the Pgpool-II configurations. You can also use [PGPOOL SET](#) command to alter the value of this parameter for a current session.

`disable_load_balance_on_write` (string)

Specify load balance behavior after write queries appear. This parameter is especially useful in streaming replication mode. When write queries are sent to primary server, the changes are applied to standby servers but there's a time lag. So if a client read the same row right after the write query, the client may not see the latest value of the row. If that's the problem, clients should always read data from the primary server. However this effectively disables load balancing, which leads to lesser performance. This parameter allows a fine tuning for the trade off between not-clustering-aware applications compatibility and performance.

If this parameter is set to `off`, read queries are load balanced even if write queries appear. This gives the best load balance performance but clients may see older data. This is useful for an environment where PostgreSQL parameter `synchronous_commit = 'remote_apply'`, or in the native replication mode, since there's no replication delay in such environments.

If this parameter is set to `transaction` and write queries appear in an explicit transaction, subsequent read queries are not load balanced until the transaction ends. Please note that read queries not in an explicit transaction are not affected by the parameter. This setting gives the best balance in most cases and you should start from this. This is the default and same behavior in Pgpool-II 3.7 or before.

If this parameter is set to `trans_transaction` and write queries appear in an explicit transaction, subsequent read queries are not load balanced in the transaction and subsequent explicit transaction until the session ends. So this parameter is safer for older applications but give lesser performance than `transaction`. Please note that read queries not in an explicit transaction are not affected by the parameter.

If this parameter is set to *always* and write queries appear, subsequent read queries are not load balanced until the session ends regardless they are in explicit transactions or not. This gives the highest compatibility with not-clustering-aware applications and the lowest performance.

statement_level_load_balance (boolean)

When set to on, the load balancing node is decided for each read query. When set to off, load balancing node is decided at the session start time and will not be changed until the session ends. For example, in applications that use connection pooling remain connections open to the backend server, because the session may be held for a long time, the load balancing node does not change until the session ends. In such applications, When `statement_level_load_balance` is enabled, it is possible to decide load balancing node per query, not per session. The default is off.

This parameter can be changed by reloading the Pgpool-II configurations.

5.8. Health Check

Pgpool-II periodically connects to the configured PostgreSQL backends to detect any error on the servers or networks. This error check procedure is called "health check". If an error is detected, Pgpool-II performs failover or degeneration depending on the configurations.

Caution

Health check requires one extra connection to each backend node, so `max_connections` in the `postgresql.conf` needs to be adjusted accordingly.

Following parameter names can also have numeric suffix at the end of each name. The suffix corresponds to backend id, which is defined in backend information, such as `backend_hostname`. example, `health_check_timeout0` is applied to backend 0's `health_check_timeout` value.

If there's no parameter with suffix, the value for the backend is taken from the parameter name which does not have a suffix. In this sense, parameter names without suffix work like "global variables".

health_check_timeout (integer)

Specifies the timeout in seconds to give up connecting to the backend PostgreSQL if the TCP connect does not succeed within this time.

This parameter serves to prevent the health check from waiting for a long time when the network cable is unplugged. Default value is 20. Setting it to 0, disables the timeout (waits until TCP/IP timeout).

This parameter can be changed by reloading the Pgpool-II configurations.

health_check_period (integer)

Specifies the interval between the health checks in seconds. Default is 0, which means health check is disabled.

This parameter can be changed by reloading the Pgpool-II configurations.

health_check_user (string)

Specifies the PostgreSQL user name to perform health check. The same user must exist in all the PostgreSQL backends. Otherwise, health check causes an error.

This parameter can be changed by reloading the Pgpool-II configurations.

health_check_password (string)

Specifies the password for the PostgreSQL user name configured in `health_check_user` to perform health check. The user and password must be same in all the PostgreSQL backends. Otherwise, health check results in an error.

If `health_check_password` is left blank Pgpool-II will first try to get the password for `health_check_user` from `pool_passwd` file before using the empty password.

Pgpool-II accepts following forms of password in either `health_check_password` or `pool_passwd` file:

AES256-CBC encrypted password

Most secure and recommended way to store password. The password string must be prefixed with `AES`. You can use `pg_enc` utility to create the correctly formatted `AES` encrypted password strings. Pgpool-II will require a valid decryption key at the startup to use the encrypted passwords. see [Section 6.4.2](#) for more details on providing the decryption key to Pgpool-II

MD5 hashed password

Not so secure as AES256, but still better than clear text password. The password string must be prefixed with `MD5`. Note that the backend must set up MD5 authentication as well. You can use `pg_md5` utility to create the correctly formatted `MD5` hashed password strings.

Plain text password

Not encrypted, clear text password. You should avoid to use this if possible. The password string must be prefixed with `TEXT`. For example if you want to set `mypass` as a password, you should specify `TEXTmypass` in the password field. In the absence of a valid prefix, Pgpool-II will considered the string as a plain text password.

This parameter can be changed by reloading the Pgpool-II configurations.

`health_check_database` (string)

Specifies the PostgreSQL database name to perform health check. The default is "(empty)", which tries "postgres" database first, then "template1" database until it succeeds

`health_check_database` was introduced in Pgpool-II **V3.5**.

This parameter can be changed by reloading the Pgpool-II configurations.

`health_check_max_retries` (integer)

Specifies the maximum number of retries to do before giving up and initiating failover when health check fails.

Tip: This setting can be useful in spotty networks, when it is expected that health checks will fail occasionally even when the master node is fine.

Tip: It is advised that [failover_on_backend_error](#) must be disabled, if you want to enable `health_check_max_retries`.

Default is 0, which means do not retry.

This parameter can be changed by reloading the Pgpool-II configurations.

`health_check_retry_delay` (integer)

Specifies the amount of time in seconds to sleep between failed health check retries (not used unless [health_check_max_retries](#) is > 0). If 0, then retries are immediate without delay.

This parameter can be changed by reloading the Pgpool-II configurations.

`connect_timeout` (integer)

Specifies the amount of time in milliseconds before giving up connecting to backend using `connect()` system call. Default is 10000 ms (10 second). The flaky network user may want to increase the value. 0 means no timeout.

Note: `connect_timeout` value is not only used for a health check, but also for creating ordinary connection pools.

This parameter can be changed by reloading the Pgpool-II configurations.

5.9. Failover and Failback

Failover means automatically detaching PostgreSQL backend node which is not accessible by Pgpool-II. This happens automatically regardless the configuration parameter settings and is so called **automatic failover** process. Pgpool-II confirms the inaccessibility of PostgreSQL backend node by using following methods:

- Regular health check process (see [Section 5.8](#) for more details). The health check process tries to connect from Pgpool-II to PostgreSQL node to confirm its healthiness. If it fails to connect, it is possible that there's something wrong with network connection between Pgpool-II and PostgreSQL, and/or PostgreSQL does not work properly. Pgpool-II does not distinguish each case and just decides that the particular PostgreSQL node is not available if health check fails.
- An error occurs while connecting to PostgreSQL, or network level errors occur while communicating with it. Pgpool-II will just disconnect the session to client if [failover_on_backend_error](#) is off in that case. However.
- In the case When clients already connect to Pgpool-II and PostgreSQL is shutdown (please note that if no client connects to Pgpool-II at all, shutting down of PostgreSQL does not trigger a failover).

If [failover_command](#) is configured and a failover happens, [failover_command](#) gets executed. [failover_command](#) should be provided by user. From 4.1 an example script for failover command is provided as `failover.sh.sample` which can be a good starting point for you.

The major role of failover command is choosing new primary server from existing standby servers and promoting it for example. Another example would be let the administrator know that a failover happens by sending a mail.

While a failover could happen when a failure occurs, it is possible to trigger it by hand. This is called a **switch over**. For instance, switching over a PostgreSQL to take its backup would be possible. Note that switching over just sets the status to down and never bringing PostgreSQL down. A switch over can be triggered by using [pcp_detach_node](#) command.

A PostgreSQL node detached by failover or switch over will never return to the previous state (attached state) automatically in the default setting. Restarting Pgpool-II with `-D` option or running [pcp_attach_node](#) makes it to the attached state again. It is recommended to confirm the replication state of [SHOW POOL NODES](#) is "streaming" before doing that. The state indicates that the standby server is properly connected to the primary server through streaming replication and both databases are in sync.

From 4.1 a new parameter [auto_failback](#) can be used to do above automatically. See [auto_failback](#) for more details.

5.9.1. Failover and Failback Settings

`failover_command` (string)

Specifies a user command to run when a PostgreSQL backend node gets detached. Pgpool-II replaces the following special characters with the backend specific information.

Table 5-6. failover command options

Special character	Description
%d	DB node ID of the detached node
%h	Hostname of the detached node
%p	Port number of the detached node

Special character	Description
%D	Database cluster directory of the detached node
%M	Old master node ID
%m	New master node ID
%H	Hostname of the new master node
%P	Old primary node ID
%r	Port number of the new master node
%R	Database cluster directory of the new master node
%N	Hostname of the old primary node (Pgpool-II 4.1 or after)
%S	Port number of the old primary node (Pgpool-II 4.1 or after)
%%	'%' character

Note: The "master node" refers to a node which has the "youngest (or the smallest) node id" among live the database nodes. In [streaming replication mode](#), this may be different from primary node. In [Table 5-6](#), %m is the new master node chosen by Pgpool-II. It is the node being assigned the youngest (smallest) node id which is alive. For example if you have 3 nodes, namely node 0, 1, 2. Suppose node 1 the primary and all of them are healthy (no down node). If node 1 fails, failover_command is called with %m = 0. And, if all standby nodes are down and primary node failover happens, failover_command is called with %m = -1 and %H,%R,\$r = "".

Note: When a failover is performed, basically Pgpool-II kills all its child processes, which will in turn terminate all the active sessions to Pgpool-II. After that Pgpool-II invokes the failover_command and after the command completion Pgpool-II starts new child processes which makes it ready again to accept client connections.

However from Pgpool-II 3.6, in the steaming replication mode, client sessions will not be disconnected any more when a failover occurs if the session does not use the failed standby server. Please note that if a query is sent while failover is processing, the session will be disconnected. If the primary server goes down, still all sessions will be disconnected. Health check timeout case will also cause the full session disconnection. Other health check error, including retry over case does not trigger full session disconnection.

Note: You can run `psql` (or whatever command) against backend to retrieve some information in the script, but you cannot run `psql` against Pgpool-II itself, since the script is called from Pgpool-II and it needs to run while Pgpool-II is working on failover.

A complete failover_command example can be found in [Section 8.3](#).

This parameter can be changed by reloading the Pgpool-II configurations.

failback_command (string)

Specifies a user command to run when a PostgreSQL backend node gets attached to Pgpool-II. Pgpool-II replaces the following special characters with the backend specific information. before executing the command.

Table 5-7. failback command options

Special character	Description
%d	DB node ID of the attached node
%h	Hostname of the attached node
%p	Port number of the attached node
%D	Database cluster directory of the attached node
%M	Old master node ID
%m	New master node ID
%H	Hostname of the new master node
%P	Current primary node ID
%r	Port number of the new master node
%R	Database cluster directory of the new master node
%N	Hostname of the old primary node (Pgpool-II 4.1 or after)
%S	Port number of the old primary node (Pgpool-II 4.1 or after)
%%	'%' character

Note: You can run `psql` (or whatever command) against backend to retrieve some information in the script, but you cannot run `psql` against Pgpool-II itself, since the script is called from Pgpool-II and it needs to run while Pgpool-II is working on failover.

This parameter can be changed by reloading the Pgpool-II configurations.

`follow_master_command` (string)

Specifies a user command to run after failover on the primary node failover. In case of standby node failover, the command will not be executed. This command also runs if a node promote request is issued by `pcp_promote_node` command. This works only in Master Replication mode with streaming replication.

Since the command is executed within a child process forked off by Pgpool-II after failover is completed, execution of follow master command does not block the service of Pgpool-II. Here is a pseudocode to illustrate how the command is executed:

```

for each backend node
{
if (the node is not the new primary)
set down node status to shared memory status
memorize that follow master command is needed to execute
}
if (we need to executed follow master command)
fork a child process
(within the child process)

for each backend node
if (the node status in shared memory is down)
execute follow master command

```

Pgpool-II replaces the following special characters with the backend specific information before executing the command.

Table 5-8. follow master command options

Special character	Description
-------------------	-------------

Special character	Description
%d	DB node ID of the detached node
%h	Hostname of the detached node
%p	Port number of the detached node
%D	Database cluster directory of the detached node
%M	Old master node ID
%m	New primary node ID
%H	Hostname of the new primary node
%P	Old primary node ID
%r	Port number of the new primary node
%R	Database cluster directory of the new primary node
%N	Hostname of the old primary node (Pgpool-II 4.1 or after)
%S	Port number of the old primary node (Pgpool-II 4.1 or after)
%%	'%' character

Note: If `follow_master_command` is not empty, then after failover on the primary node gets completed in Master Slave mode with streaming replication, Pgpool-II degenerates all nodes except the new primary and starts new child processes to be ready again to accept connections from the clients. After this, Pgpool-II executes the command configured in the `follow_master_command` for each degenerated backend nodes.

Typically `follow_master_command` command is used to recover the slave from the new primary by calling the `pcp_recovery_node` command. In the `follow_master_command`, it is recommended to check whether target PostgreSQL node is running or not using `pg_ctl` since already stopped node usually has a reason to be stopped: for example, it's broken by hardware problems or administrator is maintaining the node. If the node is stopped, skip the node. If the node is running, stop the node first and recovery it. A complete `follow_master_command` example can be found in [Section 8.3](#).

This parameter can be changed by reloading the Pgpool-II configurations.

`failover_on_backend_error` (boolean)

When set to on, Pgpool-II considers the reading/writing errors on the PostgreSQL backend connection as the backend node failure and trigger the failover on that node after disconnecting the current session. When this is set to off, Pgpool-II only report an error and disconnect the session in case of such errors.

Note: It is recommended to turn on the backend health checking (see [Section 5.8](#)) when `failover_on_backend_error` is set to off. Note, however, that Pgpool-II still triggers the failover when it detects the administrative shutdown of PostgreSQL backend server. If you want to avoid a fail over even in this case, you need to specify `DISALLOW_TO_FAILOVER` on [backend_flag](#).

This parameter can be changed by reloading the Pgpool-II configurations.

Note: Prior to Pgpool-II **V4.0**, this configuration parameter name was `fail_over_on_backend_error`.

search_primary_node_timeout (integer)

Specifies the maximum amount of time in seconds to search for the primary node when a failover scenario occurs. Pgpool-II will give up looking for the primary node if it is not found within this configured time. Default is 300 and setting this parameter to 0 means keep trying forever.

This parameter is only applicable in the streaming replication mode.

This parameter can be changed by reloading the Pgpool-II configurations.

detach_false_primary (boolean)

If set to on, detach false primary node. The default is off. This parameter is only valid in streaming replication mode and for PostgreSQL 9.6 or after since this feature uses pg_stat_wal_receiver. If PostgreSQL 9.5.x or older version is used, no error is raised, just the feature is ignored.

If there's no primary node, no checking will be performed.

If there's no standby node, and there's only one primary node, no checking will be performed.

If there's no standby node, and there's multiple primary nodes, leave the primary node which has the youngest node id and detach rest of primary nodes.

If there are one or more primaries and one or more standbys, check the connectivity between primary and standby nodes by using pg_stat_wal_receiver if PostgreSQL 9.6 or after. In this case if a primary node connects to all standby nodes, the primary is regarded as "true" primary. Other primaries are regarded as "false" primary and the false primaries will be detached if detach_false_primary is true. If no "true" primary is found, nothing will happen.

When Pgpool-II starts, the checking of false primaries are performed only once in the Pgpool-II main process. If sr_check_period is greater than 0, the false primaries checking will be performed at the same timing of streaming replication delay checking.

Note: [sr_check_user](#) must be PostgreSQL super user or in "pg_monitor" group to use this feature. To make [sr_check_user](#) in pg_monitor group, execute following SQL command by PostgreSQL super user (replace "sr_check_user" with the setting of [sr_check_user](#)):

```
GRANT pg_monitor TO sr_check_user;
```

For PostgreSQL 9.6, there's no pg_monitor group and [sr_check_user](#) must be PostgreSQL super user.

This parameter is only applicable in the streaming replication mode.

This parameter can be changed by reloading the Pgpool-II configurations.

Figure 5-1. Detecting false primaries

□

auto_failback (boolean)

When set to on, standby node be automatically failback, if the node status is down but streaming replication works normally. This is useful when standby node is degenerated by pgpool because of the temporary network failure.

To use this feature, [Section 5.11](#) must be enabled, and PostgreSQL 9.1 or later is required as backend nodes. This feature use pg_stat_replicatoin on primary node, the automatic failback is performed to standby node only. Note that failback_command will be executed as well if failback_command is not empty. If you plan to detach standby node for maintenance, set this parameter to off beforehand. Otherwise it's possible that standby node is reattached against your intention.

The default is off. This parameter can be changed by reloading the Pgpool-II configurations.

auto_failback_interval (integer)

Specifies the minimum amount of time in seconds for execution interval of auto failback. Next auto failback won't execute until that specified time have passed after previous auto failback. When Pgpool-II frequently detect backend down because of network error for example, you may avoid repeating failover and failback by setting this parameter to large enough value. The default is 60. Setting this parameter to 0 means that auto failback don't wait.

5.9.2. Failover in the raw Mode

Failover can be performed in raw mode if multiple backend servers are defined. Pgpool-II usually accesses the backend specified by `backend_hostname0` during normal operation. If the `backend_hostname0` fails for some reason, Pgpool-II tries to access the backend specified by `backend_hostname1`. If that fails, Pgpool-II tries the `backend_hostname2`, 3 and so on.

5.10. Online Recovery

Pgpool-II can synchronize database nodes and attach a node without stopping the service. This feature is called "online recovery". Online recovery can be executed by using `pcp_recovery_node` command.

For online recovery, the recovery target node must be in detached state. This means the node must be either manually detached by `pcp_detach_node` or automatically detached by Pgpool-II as a consequence of failover.

If you wish to add a PostgreSQL server node dynamically, reload the `pgpool.conf` after adding the `backend_hostname` and its associated parameters. This will register the new server to Pgpool-II as a detached backend node, after that you execute `pcp_recovery_node` command, the server is add.

Note: The recovery target PostgreSQL server must not be running for performing the online recovery. If the target PostgreSQL server has already started, you must shut it down before starting the online recovery.

Online recovery is performed in two phases. The first phase is called "first stage" and the second phase is called "second stage". You need to provide scripts for each stage. Only `replication_mode` requires the second stage. For other modes including streaming replication mode the second stage is not performed and you don't need to provide a script for the stage in `recovery_2nd_stage_command`. i.e. you can safely leave it as an empty string. Those scripts example can be found in [Section 8.3.5.3](#).

Connections from clients are not allowed only in the second stage while the data can be updated or retrieved during the first stage.

Pgpool-II performs the follows steps in online recovery:

- CHECKPOINT.
- Execute first stage of online recovery.
- Wait until all client connections have disconnected (only in `replication_mode`).
- CHECKPOINT (only in `replication_mode`). specified).
- Execute second stage of online recovery (only in `replication_mode`).
- Start up postmaster (perform `pgpool_remote_start`)

The `pgpool_remote_start` is script to start up the PostgreSQL node of recovery target. `pgpool_remote_start` receives following 2 parameters:

- Hostname of the backend node to be recovered.
- Path to the database cluster of the master(primary) node.

The script example can be found in [Section 8.3.5.3](#).

Note: The script path and filename are hard coded, `$PGDATA/pgpool_remote_start` is executed on master(primary) node.

- Node attach

Note: There is a restriction in the online recovery in [replication mode](#). If Pgpool-II itself is installed on multiple hosts, online recovery does not work correctly, because Pgpool-II has to stop all the clients during the 2nd stage of online recovery. If there are several Pgpool-II hosts, only one of them will have received the online recovery command and will block the connections from clients.

recovery_user (string)

Specifies the PostgreSQL user name to perform online recovery.

This parameter can be changed by reloading the Pgpool-II configurations.

recovery_password (string)

Specifies the password for the PostgreSQL user name configured in [recovery_user](#) to perform online recovery.

If `recovery_password` is left blank Pgpool-II will first try to get the password for [recovery_user](#) from [pool_passwd](#) file before using the empty password.

You can also specify AES256-CBC encrypted password in `recovery_password` field. To specify the AES encrypted password, password string must be prefixed with `AES` after encrypting (using `aes-256-cbc` algorithm) and encoding to `base64`.

To specify the unencrypted clear text password, prefix the password string with `TEXT`. For example if you want to set `mypass` as a password, you should specify `TEXTmypass` in the password field. In the absence of a valid prefix, Pgpool-II will considered the string as a plain text password.

You can also use [pg_enc](#) utility to create the correctly formatted AES encrypted password strings.

Note: Pgpool-II will require a valid decryption key at the startup to use the encrypted passwords. see [Section 6.4.2](#) for more details on providing the decryption key to Pgpool-II

This parameter can be changed by reloading the Pgpool-II configurations.

recovery_1st_stage_command (string)

Specifies a command to be run by master (primary) node at the first stage of online recovery. The command file must be placed in the database cluster directory for security reasons. For example, if `recovery_1st_stage_command = 'sync-command'`, then Pgpool-II will look for the command script in `$PGDATA` directory and will try to execute `$PGDATA/sync-command`.

`recovery_1st_stage_command` receives following 6 parameters:

- Path to the database cluster of the master (primary) node.
- Hostname of the backend node to be recovered.
- Path to the database cluster of the node to be recovered.
- Port number of the master (primary) node (Pgpool-II 3.4 or after).
- Node number to be recovered (Pgpool-II 4.0 or after)
- Port number to be recovered (Pgpool-II 4.1 or after)

Note: Pgpool-II accept connections and queries while `recovery_1st_stage` command is executed, so you can retrieve and update data.

Caution

`recovery_1st_stage` command runs as a SQL command from PostgreSQL's point of view. So `recovery_1st_stage` command can get prematurely killed by PostgreSQL if the PostgreSQL's `statement_time_out` is configured with the value that is smaller than the time `recovery_1st_stage_command` takes for completion.

Typical error in such case is

```
rsync used in the command is killed by signal 2 for example.
```

This parameter can be changed by reloading the Pgpool-II configurations.

`recovery_2nd_stage_command` (string)

Specifies a command to be run by master node at the second stage of online recovery. This command is required only [replication_mode](#), so for other modes don't need to provide a command file. The command file must be placed in the database cluster directory for security reasons. For example, if `recovery_2nd_stage_command = 'sync-command'`, then Pgpool-II will look for the command script in `$PGDATA` directory and will try to execute `$PGDATA/sync-command`.

`recovery_2nd_stage_command` receives following 6 parameters:

- Path to the database cluster of the master(primary) node.
- Hostname of the backend node to be recovered.
- Path to the database cluster of the node to be recovered.
- Port number of the master (primary) node (Pgpool-II 3.4 or after).
- Node number to be recovered (Pgpool-II 4.0 or after)
- Port number to be recovered (Pgpool-II 4.1 or after)

Note: Pgpool-II **does not** accept client connections and queries during the execution of `recovery_2nd_stage_command` command, and waits for the existing clients to close their connections before executing the command. Therefore, the `recovery_2nd_stage_command` may not execute if the client stays connected for a long time.

Caution

`recovery_2nd_stage` command runs as a SQL command from PostgreSQL's point of view. Therefore, `recovery_2nd_stage` command can get prematurely killed by PostgreSQL if the PostgreSQL's `statement_time_out` is configured with the value that is smaller than the time `recovery_2nd_stage_command` takes for completion.

This parameter can be changed by reloading the Pgpool-II configurations.

`recovery_timeout` (integer)

Specifies the timeout in seconds to cancel the online recovery if it does not complete within this time. Since Pgpool-II does not accept the connections during the second stage of online recovery, this parameter can be used to cancel the online recovery to manage the service down time during the online recovery.

This parameter can be changed by reloading the Pgpool-II configurations.

`client_idle_limit_in_recovery` (integer)

Specifies the time in seconds to disconnect a client if it remains idle since the last query during the online recovery. `client_idle_limit_in_recovery` is similar to the [client_idle_limit](#) but only takes effect during the second stage of online recovery.

This is useful for preventing the Pgpool-II recovery from being disturbed by the lazy clients or if the TCP/IP connection between the client and Pgpool-II is accidentally down (a cut cable for instance).

Note: `client_idle_limit_in_recovery` must be smaller than [recovery_timeout](#). Otherwise, [recovery_timeout](#) comes first and you will see following error while executing online recovery:

```
ERROR: node recovery failed, waiting connection closed in the other pgpools timeout
```

If set to -1, all clients get immediately disconnected when the second stage of online recovery starts. The default is 0, which turns off the feature.

This parameter can be changed by reloading the Pgpool-II configurations. You can also use [PGPOOL SET](#) command to alter the value of this parameter for a current session.

5.11. Streaming Replication Check

Pgpool-II can work with PostgreSQL native Streaming Replication, that is available since PostgreSQL 9.0. To configure Pgpool-II with streaming replication, enable [master_slave_mode](#) and set [master_slave_sub_mode](#) to 'stream'.

Pgpool-II assumes that Streaming Replication is configured with Hot Standby on PostgreSQL, which means that the standby database can handle read-only queries.

`sr_check_period` (integer)

Specifies the time interval in seconds to check the streaming replication delay. Default is 0, which means the check is disabled.

This parameter can be changed by reloading the Pgpool-II configurations.

`sr_check_user` (string)

Specifies the PostgreSQL user name to perform streaming replication check. The user must have LOGIN

privilege and exist on all the PostgreSQL backends.

Note: [sr_check_user](#) and [sr_check_password](#) are used even when [sr_check_period](#) is set to 0 (disabled) for the identification of the primary server.

This parameter can be changed by reloading the Pgpool-II configurations.

`sr_check_password` (string)

Specifies the password of the [sr_check_user](#) PostgreSQL user to perform the streaming replication checks. Use "" (empty string) if the user does not requires a password.

If `sr_check_password` is left blank Pgpool-II will first try to get the password for [sr_check_user](#) from [pool_passwd](#) file before using the empty password.

Pgpool-II accepts following forms of password in either `sr_check_password` or [pool_passwd](#) file:

AES256-CBC encrypted password

Most secure and recommended way to store password. The password string must be prefixed with AES. You can use [pg_enc](#) utility to create the correctly formatted AES encrypted password strings. Pgpool-II will require a valid decryption key at the startup to use the encrypted passwords. see [Section 6.4.2](#) for more details on providing the decryption key to Pgpool-II

MD5 hashed password

Not so secure as AES256, but still better than clear text password. The password string must be prefixed with MD5. Note that the backend must set up MD5 authentication as well. You can use [pg_md5](#) utility to create the correctly formatted MD5 hashed password strings.

Plain text password

Not encrypted, clear text password. You should avoid to use this if possible. The password string must be prefixed with TEXT. For example if you want to set `mypass` as a password, you should specify `TEXTmypass` in the password field. In the absence of a valid prefix, Pgpool-II will considered the string as a plain text password.

This parameter can be changed by reloading the Pgpool-II configurations.

`sr_check_database` (string)

Specifies the database to perform streaming replication delay checks. The default is "postgres".

This parameter can be changed by reloading the Pgpool-II configurations.

`delay_threshold` (integer)

Specifies the maximum tolerance level of replication delay in WAL bytes on the standby server against the primary server. If the delay exceeds this configured level, Pgpool-II stops sending the SELECT queries to the standby server and starts routing everything to the primary server even if [load_balance_mode](#) is enabled, until the standby catches-up with the primary. Setting this parameter to 0 disables the delay checking. This delay threshold check is performed every [sr_check_period](#). Default is 0.

This parameter can be changed by reloading the Pgpool-II configurations.

`log_standby_delay` (string)

Specifies when to log the replication delay. Below table contains the list of all valid values for the parameter.

Table 5-9. Log standby delay options

Value	Description
-------	-------------

Value	Description
'none'	Never log the standby delay
'always'	Log the standby delay, every time the replication delay is checked
'if_over_threshold'	Only log the standby delay, when it exceeds delay_threshold value

This parameter can be changed by reloading the `Pgpool-II` configurations.

5.12. In Memory Query Cache

In memory query cache can be used with all modes of `Pgpool-II`. `Pgpool-II` does not need a restart when the cache gets outdated because of the underlying table updates.

In memory cache saves the pair of `SELECT` statement and its result (along with the `Bind` parameters, if the `SELECT` is an extended query). If the same `SELECT`s comes in, `Pgpool-II` returns the value from cache. Since no `SQL` parsing nor access to `PostgreSQL` are involved, the serving of results from the in memory cache is extremely fast.

Note: Basically following `SELECT`s will not be cached:

- SELECTs including non immutable functions
- SELECTs including temp tables, unlogged tables
- SELECT result is too large (`memqcache_maxcache`)
- SELECT FOR SHARE/UPDATE
- SELECT starting with `"/*NO QUERY CACHE*/"` comment
- SELECT including system catalogs
- SELECT uses `TABLESAMPLE`

However, `VIEWS` and `SELECT`s accessing unlogged tables can be cached by specifying in the [white_memqcache_table_list](#).

On the other hand, it might be slower than the normal path in some cases, because it adds some overhead to store cache. Moreover when a table is updated, `Pgpool-II` automatically deletes all the caches related to the table. Therefore, the performance will be degraded by a system with a lot of updates. If the query cache hit ratio (it can be checked by using [SHOW POOL_CACHE](#)) is lower than 70%, you might want to disable in memory cache.

5.12.1. Enabling in memory query cache

`memory_cache_enabled` (boolean)

Setting to on enables the memory cache. Default is off.

This parameter can only be set at server start.

Note: The query cache will also be used by shared relation cache if [enable_shared_relcache](#) is set to on. Moreover the query cache is used even if [memory_cache_enabled](#) parameter is set to off. See [Section 5.15](#) for more details to relation cache.

5.12.2. Choosing cache storage

memqcache_method (string)

Specifies the storage type to be used for the cache. Below table contains the list of all valid values for the parameter.

Table 5-10. Memcache method options

Value	Description
'shmem'	Use shared memory
'memcached'	Use memcached

Default is 'shmem'.

This parameter can only be set at server start.

5.12.3. Common configurations

These below parameter are valid for both shmem and memcached type query cache.

memqcache_expire (integer)

Specifies the life time of query cache in seconds. Default is 0. which means no cache expiration and cache remains valid until the table is updated.

This parameter can be changed by reloading the Pgpool-II configurations.

Note: memqcache_expire and [memqcache_auto_cache_invalidation](#) are orthogonal to each other.

memqcache_auto_cache_invalidation (boolean)

Setting to on, automatically deletes the cache related to the updated tables. When off, cache is not deleted.

Default is on.

Note: This parameters [memqcache_auto_cache_invalidation](#) and [memqcache_expire](#) are orthogonal to each other.

This parameter can be changed by reloading the Pgpool-II configurations.

memqcache_maxcache (integer)

Specifies the maximum size in bytes of the SELECT query result to be cached. The result with data size larger than this value will not be cached by Pgpool-II. When the caching of data is rejected because of the size constraint the following message is shown.

```
LOG: pid 13756: pool_add_temp_query_cache: data size exceeds memqcache_maxcache. current:4095 requested:111
```

Note: For the shared memory query('shmem') cache the `memqcache_maxcache` must be set lower than [memqcache_cache_block_size](#) and for 'memcached' it must be lower than the size of slab (default is 1 MB).

This parameter can be changed by reloading the Pgpool-II configurations.

`white_memqcache_table_list` (string)

Specifies a comma separated list of table names whose SELECT results should be cached by Pgpool-II. This parameter only applies to VIEWS and SELECTs accessing unlogged tables. Regular tables can be cached unless specified by [black_memqcache_table_list](#).

You can use regular expression into the list to match table name (to which `^` and `$` are automatically added).

Note: If the queries can refer the table with and without the schema qualification then you must add both entries(with and without schema name) in the list.

```
#For example:  
#If the queries sometime use "table1" and other times "public.table1"  
#to refer the table1 then the white_memqcache_table_list  
#would be configured as follows.  
  
white_memqcache_table_list = "table1,public.table1"
```

This parameter can be changed by reloading the Pgpool-II configurations.

`black_memqcache_table_list` (string)

Specifies a comma separated list of table names whose SELECT results should **NOT** be cached by the Pgpool-II.

You can use regular expression into the list to match table name (to which `^` and `$` are automatically added),

Note: If the queries can refer the table with and without the schema qualification then you must add both entries(with and without schema name) in the list.

```
#For example:  
#If the queries sometime use "table1" and other times "public.table1"  
#to refer the table1 then the black_memqcache_table_list  
#would be configured as follows.  
  
black_function_list = "table1,public.table1"
```

This parameter can be changed by reloading the Pgpool-II configurations.

Note: `black_memqcache_table_list` precedence over [white_memqcache_table_list](#)

memqcache_oiddir (string)

Specifies the full path to the directory for storing the `oids` of tables used by `SELECT` queries.

`memqcache_oiddir` directory contains the sub directories for the databases. The directory name is the OID of the database. In addition, each database directory contains the files for each table used by `SELECT` statement. Again the name of the file is the OID of the table. These files contains the pointers to query cache which are used as key for deleting the caches.

Note: Normal restart of Pgpool-II does not clear the contents of `memqcache_oiddir`.

This parameter can be changed by reloading the Pgpool-II configurations.

5.12.4. Configurations to use shared memory

These are the parameters used with shared memory as the cache storage.

memqcache_total_size (integer)

Specifies the shared memory cache size in bytes.

This parameter can only be set at server start.

memqcache_max_num_cache (integer)

Specifies the number of cache entries. This is used to define the size of cache management space.

Note: The management space size can be calculated by: `memqcache_max_num_cache * 48` bytes. Too small number will cause an error while registering cache. On the other hand too large number will just waste space.

This parameter can only be set at server start.

memqcache_cache_block_size (integer)

Specifies the cache block size. Pgpool-II uses the cache memory arranged in `memqcache_cache_block_size` blocks. `SELECT` result is packed into the block and must fit in a single block. And the results larger than `memqcache_cache_block_size` are not cached.

`memqcache_cache_block_size` must be set to at least 512.

This parameter can only be set at server start.

5.12.5. Configurations to use memcached

These are the parameters used with memcached as the cache storage.

memqcache_memcached_host (string)

Specifies the host name or the IP address on which `memcached` works. You can use `'localhost'` if `memcached` and `Pgpool-II` resides on same server.

This parameter can only be set at server start.

memqcache_memcached_port (integer)

Specifies the port number of memcached. Default is 11211.

This parameter can only be set at server start.

5.13. Secure Socket Layer (SSL)

5.13.1. SSL Settings

ssl (boolean)

When set to on, Pgpool-II enables the SSL for both the frontend and backend communications. Default is off.

Note: [ssl_key](#) and [ssl_cert](#) must also be configured in order for SSL to work with frontend connections.

Note: For SSL to work Pgpool-II must be build with OpenSSL support. See [Section 2.5](#) for details on building the Pgpool-II.

This parameter can only be set at server start.

ssl_key (string)

Specifies the path to the private key file to be used for incoming frontend connections. If specifies relative path, it is based on the directory where pgpool is starting up. There is no default value for this option, and if left unset SSL will be disabled for incoming frontend connections.

This parameter can only be set at server start.

ssl_cert (string)

Specifies the path to the public x509 certificate file to be used for the incoming frontend connections. If specifies relative path, based path is Pgpool-II is run directory. There is no default value for this option, and if left unset SSL will be disabled for incoming frontend connections.

This parameter can only be set at server start.

ssl_ca_cert (string)

Specifies the path to a PEM format CA certificate files, which can be used to verify the backend server certificates. This is analogous to the `-CApath` option of the `OpenSSL verify(1)` command.

This parameter can only be set at server start.

ssl_ca_cert_dir (string)

Specifies the path to a directory containing PEM format CA certificate files, which can be used to verify the backend server certificates. This is analogous to the `-CApath` option of the `OpenSSL verify(1)` command.

The default value for this option is unset, which means no verification takes place. Verification will still happen if this option is not set but a value is provided for [ssl_ca_cert](#).

This parameter can only be set at server start.

ssl_ciphers (string)

Specifies a list of SSL cipher suites that are allowed to be used on secure connections. See the ciphers

manual page in the [OpenSSL](#) package for the syntax of this setting and a list of supported values. The default value is `HIGH:MEDIUM:+3DES:!aNULL`, which is same as [PostgreSQL](#). See [PostgreSQL](#) manual to know why the value is chosen.

This parameter can only be set at server start.

`ssl_prefer_server_ciphers` (boolean)

Specifies whether to use the server's SSL cipher preferences, rather than the client's. The default value is false.

This parameter can only be set at server start.

`ssl_ecdh_curve` (string)

Specifies the name of the curve to use in [ECDH](#) key exchange. It needs to be supported by all clients that connect. It does not need to be the same curve used by the server's Elliptic Curve key. The default value is `prime256v1`.

OpenSSL names for the most common curves are: `prime256v1` (NIST P-256), `secp384r1` (NIST P-384), `secp521r1` (NIST P-521). The full list of available curves can be shown with the command `openssl ecparam -list_curves`. Not all of them are usable in [TLS](#) though.

This parameter can only be set at server start.

`ssl_dh_params_file` (string)

Specifies the name of the file containing Diffie-Hellman parameters used for so-called ephemeral DH family of SSL ciphers. The default is empty. In which case compiled-in default DH parameters used. Using Custom DH parameters reduces the exposure if an attacker manages to crack the well-known compiled-in DH parameters. You can create your own DH parameters file with the command `openssl -out dhparams.pem 2048`.

This parameter can only be set at server start.

5.13.2. Generating SSL certificates

Certificate handling is outside the scope of this document. The [Secure TCP/IP Connections with SSL](#) page at [postgresql.org](#) has pointers with sample commands for how to generate self-signed certificates.

5.14. Watchdog

Watchdog configuration parameters are described in `pgpool.conf`. There is sample configuration in the `WATCHDOG` section of `pgpool.conf.sample` file. All following options are required to be specified in `watchdog` process.

5.14.1. Enable watchdog

`use_watchdog` (boolean)

If on, activates the watchdog. Default is off

This parameter can only be set at server start.

5.14.2. Watchdog communication

`wd_hostname` (string)

Specifies the hostname or IP address of `Pgpool-II` server. This is used for sending/receiving queries and packets, and also as an identifier of the watchdog node.

This parameter can only be set at server start.

wd_port (integer)

Specifies the port number to be used by watchdog process to listen for connections. Default is 9000.

This parameter can only be set at server start.

wd_authkey (string)

Specifies the authentication key used for all watchdog communications. All Pgpool-II must have the same key. Packets from watchdog having different key will get rejected. This authentication is also applied to the heartbeat signals when the `heartbeat` mode is used as a lifecheck method.

Since in Pgpool-II **V3.5** or beyond `wd_authkey` is also used to authenticate the watchdog IPC clients, all clients communicating with Pgpool-II watchdog process needs to provide this `wd_authkey` value for "IPCAuthKey" key in the JSON data of the command.

Default is "" (empty) which means disables the watchdog authentication.

This parameter can only be set at server start.

5.14.3. Upstream server connection

trusted_servers (string)

Specifies the list of trusted servers to check the up stream connections. Each server in the list is required to respond to ping. Specify a comma separated list of servers such as "hostA,hostB,hostC". If none of the server are reachable, watchdog will regard it as failure of the Pgpool-II. Therefore, it is recommended to specify multiple servers. Please note that you should not assign PostgreSQL servers to this parameter.

This parameter can only be set at server start.

ping_path (string)

Specifies the path of a ping command for monitoring connection to the upper servers. Set the only path of the directory containing the ping utility, such as "/bin" or such directory.

This parameter can only be set at server start.

5.14.4. Virtual IP control

delegate_IP (string)

Specifies the virtual IP address (VIP) of Pgpool-II that is connected from client servers (application servers etc.). When a Pgpool-II is switched from standby to active, the Pgpool-II takes over this VIP. **VIP will not be brought up in case the quorum does not exist.** Default is ""(empty): which means virtual IP will never be brought up.

This parameter can only be set at server start.

if_cmd_path (string)

Specifies the path to the command that Pgpool-II will use to switch the virtual IP on the system. Set only the path of the directory containing the binary, such as "/sbin" or such directory. If [if_up_cmd](#) or [if_down_cmd](#) starts with "/", this parameter will be ignored.

This parameter can only be set at server start.

if_up_cmd (string)

Specifies the command to bring up the virtual IP. Set the command and parameters such as "ip addr add \$_IP_\$/24 dev eth0 label eth0:0". Since root privilege is required to execute this command, use `setuid` on `ip` command or allow Pgpool-II startup user (postgres user by default) to run `sudo` command without a password, and specify it such as "/usr/bin/sudo /sbin/ip addr add \$_IP_\$/24 dev eth0 label eth0:0". `$_IP_` will get replaced by the IP address specified in the [delegate_IP](#).

This parameter can only be set at server start.

if_down_cmd (string)

Specifies the command to bring down the virtual IP. Set the command and parameters such as "ip addr del \$_IP_\$ /24 dev eth0". Since root privilege is required to execute this command, use `setuid` on `ip` command or allow Pgpool-II startup user (`postgres` user by default) to run `sudo` command without a password, and specify it such as `"/usr/bin/sudo /sbin/ip addr del $_IP_$ /24 dev eth0"`. `$_IP_$` will get replaced by the IP address specified in the [delegate_IP](#).

This parameter can only be set at server start.

arping_path (string)

Specifies the path to the command that Pgpool-II will use to send the ARP requests after the virtual IP switch. Set only the path of the directory containing the binary, such as `"/usr/sbin"` or such directory. If [arping_cmd](#) starts with `"/"`, this parameter will be ignored.

This parameter can only be set at server start.

arping_cmd (string)

Specifies the command to use for sending the ARP requests after the virtual IP switch. Set the command and parameters such as `"arping -U $_IP_$ -w 1 -I eth0"`. Since root privilege is required to execute this command, use `setuid` on `ip` command or allow Pgpool-II startup user (`postgres` user by default) to run `sudo` command without a password, and specify it such as `"/usr/bin/sudo /usr/sbin/arping -U $_IP_$ -w 1 -I eth0"`. `$_IP_$` will get replaced by the IP address specified in the [delegate_IP](#).

This parameter can only be set at server start.

5.14.5. Behaviour on escalation and de-escalation

Configuration about behavior when Pgpool-II escalates to active (virtual IP holder)

clear_memqcache_on_escalation (boolean)

When set to on, watchdog clears all the query cache in the shared memory when pgpool-II escalates to active. This prevents the new active Pgpool-II from using old query caches inconsistent to the old active.

Default is on.

This works only if [memqcache_method](#) is `'shmem'`.

This parameter can only be set at server start.

wd_escalation_command (string)

Watchdog executes this command on the node that is escalated to the master watchdog.

This command is executed just before bringing up the virtual IP if that is configured on the node.

This parameter can only be set at server start.

wd_de_escalation_command (string)

Watchdog executes this command on the master Pgpool-II watchdog node when that node resigns from the master node responsibilities. A master watchdog node can resign from being a master node, when the master node Pgpool-II shuts down, detects a network blackout or detects the lost of **quorum**.

This command is executed before bringing down the virtual/floating IP address if it is configured on the watchdog node.

`wd_de_escalation_command` is not available prior to Pgpool-II **V3.5**.

This parameter can only be set at server start.

5.14.6. Controlling the Failover behavior

These settings are used to control the behavior of backend node failover when the watchdog is enabled. The

effect of these configurations is limited to the failover/degenerate requests initiated by Pgpool-II internally, while the user initiated detach backend requests (using PCP command) by-pass these configuration settings.

failover_when_quorum_exists (boolean)

When enabled, Pgpool-II will consider quorum when it performs the degenerate/failover on backend node.

We can say that "quorum exists" if the number of live watchdog nodes (that is number of Pgpool-II nodes) can be a majority against the total number of watchdog nodes. For example, suppose number of watchdog nodes is 5. If number of live nodes is greater than or equal to 3, then quorum exists. On the other hand if number of live nodes is 2 or lower, quorum does not exist since it never be majority.

If the quorum exists, Pgpool-II could work better on failure detection because even if a watchdog node mistakenly detects a failure of backend node, it would be denied by other major watchdog nodes. Pgpool-II works that way when [failover_require_consensus](#) is on (the default), but you can change it so that immediate failover happens when a failure is detected. A Pgpool-II node which mistakenly detects failure of backend node will quarantine the backend node.

The existence of quorum can be shown by invoking [pcp_watchdog_info](#) command with `--verbose` option. If `Quorum state` is QUORUM EXIST or QUORUM IS ON THE EDGE, then the quorum exists. If `Quorum state` is QUORUM ABSENT, then the quorum does not exist.

Please note that if the number of watchdog nodes is even, we regard that quorum exists when the number of live nodes is greater than or equal to half of total watchdog nodes.

In the absence of the quorum, Pgpool-II node that detects the backend failure will quarantine the failed backend node until the quorum exists again.

Although it is possible to force detaching the quarantine node by using `pcp_detach_node` command, it is not possible to attach the node again by using `pcp_attach_node` command.

The quarantine nodes behaves similar to the detached backend nodes but unlike failed/degenerated backends the quarantine status is not propagated to the other Pgpool-II nodes in the watchdog cluster, So even if the backend node is in the quarantine state on one Pgpool-II node, other Pgpool-II nodes may still continue to use that backend.

Although there are many similarities in quarantine and failover operations, but they both differ in a very fundamental way. The quarantine operations does not executes the [failover_command](#) and silently detaches the problematic node, So in the case when the master backend node is quarantined, the Pgpool-II will not promote the standby to take over the master responsibilities and until the master node is quarantined the Pgpool-II will not have any usable master backend node.

Moreover, unlike for the failed nodes, Pgpool-II keeps the health-check running on the quarantined nodes and as soon as the quarantined node becomes reachable again it gets automatically re-attached.

From Pgpool-II **V4.1** onward, if the watchdog-master node fails to build the consensus for primary backend node failover and the primary backend node gets into a quarantine state, then it resigns from its master/coordinator responsibilities and lowers its `wd_priority` for next leader election and let the cluster elect some different new leader.

Note: When the master node fails to build the consensus for standby backend node failure, it takes no action and similarly quarantined standby backend nodes on watchdog-master do not trigger a new leader election.

If this parameter is off, failover will be triggered even if quorum does not exist.

Default is on.

`failover_when_quorum_exists` is not available prior to Pgpool-II **V3.7**.

This parameter can only be set at server start.

Note: enabling `failover_when_quorum_exists` is not allowed in native replication mode.

`failover_require_consensus` (boolean)

When enabled, Pgpool-II will perform the degenerate/failover on a backend node if the watchdog quorum exists and at-least minimum number of nodes necessary for the quorum vote for the failover.

For example, in a three node watchdog cluster, the failover will only be performed until at least two nodes ask for performing the failover on the particular backend node.

If this parameter is off, failover will be triggered even if there's no consensus.

Default is on.

Caution

When `failover_require_consensus` is enabled, Pgpool-II does not execute the failover until it get enough votes from other Pgpool-II nodes. So it is strongly recommended to enable the backend health check on all Pgpool-II nodes to ensure proper detection of backend node failures. For more details of health check, see [Section 5.8](#).

Note: enabling `failover_require_consensus` is not allowed in native replication mode.

`failover_require_consensus` is not available prior to Pgpool-II **V3.7**. and it is only effective when [failover_when_quorum_exists](#) is enabled

This parameter can only be set at server start.

`allow_multiple_failover_requests_from_node` (boolean)

This parameter works in connection with the [failover_require_consensus](#). When enabled, a single Pgpool-II node can cast multiple votes for the failover.

For example, in a three node watchdog cluster, if one Pgpool-II node sends two failover requests for a particular backend node failover, Both requests will be counted as a separate vote in the favor of the failover and Pgpool-II will execute the failover, even if it does not get the vote from any other Pgpool-II node.

For example, if an error found in a health check round does not get enough vote and the error still persists, next round of health check will give one more vote. This parameter is useful if you want to detect a persistent error which might not be found by other watchdog nodes.

Default is off.

`allow_multiple_failover_requests_from_node` is not available prior to Pgpool-II **V3.7**. and it is only effective when both [failover_when_quorum_exists](#) and [failover_require_consensus](#) are enabled

This parameter can only be set at server start.

`enable_consensus_with_half_votes` (boolean)

This parameter configures how the majority rule computation is made by Pgpool-II for calculating the quorum and resolving the consensus for failover.

When enabled the existence of quorum and consensus on failover requires only half of the total number of votes configured in the cluster. Otherwise, both of these decisions require at least one more vote than half of the total number of votes. For failover, this parameter works in conjunction with the [failover_require_consensus](#). In both cases, whether making a decision of quorum existence or building

the consensus on failover this parameter only comes into play when the watchdog cluster is configured for even number of Pgpool-II nodes. The majority rule decision in the watchdog cluster having an odd number of participants. It is not affected by the value of this configuration parameter.

For example, when this parameter is enabled in a two node watchdog cluster, one Pgpool-II node needs to be alive to make the quorum exist. If the parameter is off, two nodes need to be alive to make quorum exist.

When this parameter is enabled in a four node watchdog cluster, two Pgpool-II node needs to be alive to make the quorum exist. If the parameter is off, three nodes need to be alive to make quorum exist.

By enabling this parameter, you should aware that you take a risk to make split-brain happen. For example, in four node cluster consisted of node A, B, C and D, it is possible that the cluster goes into two separated networks (A, B) and (C, D). For (A, B) and (C, D) the quorum still exist since for both groups there are two live nodes out of 4. The two groups choose their own master watchdog, which is a split-brain.

Default is off.

`enable_consensus_with_half_votes` is not available prior to Pgpool-II **V4.1**. The prior versions work as if the parameter is on.

This parameter can only be set at server start.

5.14.7. Life checking Pgpool-II

Watchdog checks pgpool-II status periodically. This is called "life check".

`wd_lifecheck_method` (string)

Specifies the method of life check. This can be either of 'heartbeat' (default), 'query' or 'external'.

heartbeat: In this mode, watchdog sends the heartbeat signals (UDP packets) periodically to other Pgpool-II. Similarly watchdog also receives the signals from other Pgpool-II. If there are no signal for a certain period, watchdog regards is as failure of the Pgpool-II.

query: In this mode, watchdog sends the monitoring queries to other Pgpool-II and checks the response. When installation location between Pgpool-II servers is far, query may be useful.

Caution

In query mode, you need to set `num_init_children` large enough if you plan to use watchdog. This is because the watchdog process connects to Pgpool-II as a client.

external: This mode disables the built in lifecheck of Pgpool-II watchdog and relies on external system to provide node health checking of local and remote watchdog nodes.

`external` mode is not available in versions prior to Pgpool-II **V3.5**.

This parameter can only be set at server start.

`wd_monitoring_interfaces_list` (string)

Specify a comma separated list of network device names, to be monitored by the watchdog process for the network link state. If all network interfaces in the list becomes inactive (disabled or cable unplugged), the watchdog will consider it as a complete network failure and the Pgpool-II node will commit the suicide. Specifying an "(empty) list disables the network interface monitoring. Setting it to 'any' enables the monitoring on all available network interfaces except the loopback. Default is " empty list (monitoring disabled).

`wd_monitoring_interfaces_list` is not available in versions prior to Pgpool-II **V3.5**.

This parameter can only be set at server start.

`wd_interval` (integer)

Specifies the interval between life checks of Pgpool-II in seconds. (A number greater than or equal to 1)
Default is 10.

This parameter can only be set at server start.

wd_priority (integer)

This parameter can be used to elevate the local watchdog node priority in the elections to select master watchdog node. The node with the higher `wd_priority` value will get selected as master watchdog node when cluster will be electing its new master node at the time of cluster startup or in the event of old master watchdog node failure

`wd_priority` is not available in versions prior to Pgpool-II **V3.5**.

This parameter can only be set at server start.

wd_ipc_socket_dir (string)

The directory where the UNIX domain socket accepting Pgpool-II watchdog IPC connections will be created. Default is `'/tmp'`. Be aware that this socket might be deleted by a cron job. We recommend to set this value to `'/var/run'` or such directory.

`wd_ipc_socket_dir` is not available in versions prior to Pgpool-II **V3.5**.

This parameter can only be set at server start.

5.14.8. Lifeclock Heartbeat mode configuration

wd_heartbeat_port (integer)

Specifies the UDP port number to receive heartbeat signals. Default is 9694. `wd_heartbeat_port` is only applicable if the [wd_lifecycle_method](#) is set to 'heartbeat'

This parameter can only be set at server start.

wd_heartbeat_keepalive (integer)

Specifies the interval time in seconds between sending the heartbeat signals. Default is 2.

`wd_heartbeat_keepalive` is only applicable if the [wd_lifecycle_method](#) is set to 'heartbeat'

This parameter can only be set at server start.

wd_heartbeat_deadtime (integer)

Specifies the time in seconds before marking the remote watchdog node as failed/dead node, if no heartbeat signal is received within that time. Default is 30 `wd_heartbeat_deadtime` is only applicable if the [wd_lifecycle_method](#) is set to 'heartbeat'

This parameter can only be set at server start.

heartbeat_destination0 (string)

Specifies the IP address or `hostname` of destination the remote Pgpool-II for sending the heartbeat signals. Multiple destinations can be configured for the heartbeat signals, the number at the end of the parameter name is referred as the "destination number", that starts from 0.

`heartbeat_destination` is only applicable if the [wd_lifecycle_method](#) is set to 'heartbeat'

This parameter can only be set at server start.

heartbeat_destination_port0 (integer)

Specifies the destination port number of the remote Pgpool-II for sending the heartbeat signals. Multiple destinations can be configured for the heartbeat signals, the number at the end of the parameter name is referred as the "destination number", that starts from 0.

`heartbeat_destination_port` is only applicable if the [wd_lifecycle_method](#) is set to 'heartbeat'

This parameter can only be set at server start.

heartbeat_device0 (string)

Specifies the network device name for sending the heartbeat signals to the destination specified by heartbeat_destinationX:heartbeat_destination_portX. Different heartbeat devices can be configured for each heartbeat destination by changing the value of X(destination number). At the end of parameter name. The destination index number starts from 0.

heartbeat_device is only applicable if the [wd_lifecycle_method](#) is set to 'heartbeat'

This parameter can only be set at server start.

5.14.9. Lifecycle Query mode configuration

wd_life_point (integer)

Specifies the number of times to retry a failed life check of Pgpool-II. Valid value could be a number greater than or equal to 1. Default is 3.

wd_life_point is only applicable if the [wd_lifecycle_method](#) is set to 'query'

This parameter can only be set at server start.

wd_lifecycle_query (string)

Specifies the query to use for the life check of remote Pgpool-II. Default is "SELECT 1".

wd_lifecycle_query is only applicable if the [wd_lifecycle_method](#) is set to 'query'

This parameter can only be set at server start.

wd_lifecycle_dbname (string)

Specifies the database name for the connection used for the life check of remote Pgpool-II. Default is "template1".

wd_lifecycle_dbname is only applicable if the [wd_lifecycle_method](#) is set to 'query'

This parameter can only be set at server start.

wd_lifecycle_user (string)

Specifies the user name for the connection used for the life check of remote Pgpool-II. Default is "nobody".

wd_lifecycle_user is only applicable if the [wd_lifecycle_method](#) is set to 'query'

This parameter can only be set at server start.

wd_lifecycle_password (string)

Specifies the password for the user used for the life check of remote Pgpool-II.

If [wd_lifecycle_password](#) is left blank Pgpool-II will first try to get the password for [wd_lifecycle_user](#) from [pool_passwd](#) file before using the empty password.

You can also specify AES256-CBC encrypted password in [wd_lifecycle_password](#) field. To specify the AES encrypted password, password string must be prefixed with AES after encrypting (using aes-256-cbc algorithm) and encoding to base64.

To specify the unencrypted clear text password, prefix the password string with TEXT. For example if you want to set mypass as a password, you should specify TEXTmypass in the password field. In the absence of a valid prefix, Pgpool-II will consider the string as a plain text password.

You can also use [pg_enc](#) utility to create the correctly formatted AES encrypted password strings.

Note: Pgpool-II will require a valid decryption key at the startup to use the encrypted passwords. see [Section 6.4.2](#) for more details on providing the decryption key to Pgpool-

`wd_lifecycle_password` is only applicable if the [wd_lifecycle_method](#) is set to 'query'

This parameter can only be set at server start.

Default is ""(empty).

5.14.10. Watchdog servers configurations

`other_pgpool_hostname0` (string)

Specifies the hostname of remote Pgpool-II server for watchdog node. The number at the end of the parameter name is referred as "server id", and it starts from 0.

This parameter can only be set at server start.

`other_pgpool_port0` (integer)

Specifies the port number of the remote Pgpool-II server for watchdog node. The number at the end of the parameter name is referred as "server id", and it starts from 0.

This parameter can only be set at server start.

`other_wd_port0` (integer)

Specifies the watchdog port number of the remote Pgpool-II server for watchdog node. The number at the end of the parameter name is referred as "server id", and it starts from 0.

This parameter can only be set at server start.

5.15. Misc Configuration Parameters

`relcache_expire` (integer)

Specifies the relation cache expiration time in seconds. The relation cache is used for caching the query result of PostgreSQL system catalogs that is used by Pgpool-II to get various information including the table structures and to check table types(e.g. To check if the referred table is a temporary table or not). The cache is maintained in the local memory space of Pgpool-II child process and its lifetime is same as of the child process. The cache is also maintained in shared memory to share among child processes,if enable [enable_shared_relcache](#). So If the table is modified using `ALTER TABLE` or some other means, the relcache becomes inconsistent. For this purpose, `relcache_expire` controls the life time of the cache. Default is 0, which means the cache never expires.

This parameter can only be set at server start.

`relcache_size` (integer)

Specifies the number of relcache entries. Default is 256. The cache is created about 10 entries per table. So you can estimate the required number of relation cache at "number of using table * 10".

Note: If the below message frequently appears in the Pgpool-II log, you may need to increase the `relcache_size` for better performance.

```
"pool_search_relcache: cache replacement happened"
```

This parameter can only be set at server start.

`enable_shared_relcache` (boolean)

By setting to on, relation cache is shared among Pgpool-II child processes using the in memory query cache (see [Section 5.12.1](#) for more details). Default is on. Each child process needs to access to the system catalog from PostgreSQL. By enabling this feature, other process can extract the catalog lookup result from the query cache and it should reduce the frequency of the query. Cache invalidation is not happen even if the system catalog is modified. So it is strongly recommend to set time out base cache invalidation by using [relcache_expire](#) parameter.

This parameter can be used even if [memory_cache_enabled](#) is off. In this case some query cache parameters([memqcache_method](#), [memqcache_maxcache](#) and each cache storage parameter) is used together.

Pgpool-II search the local relation cache first. If it is not found on the cache, the shared relation query cache is searched if this feature is enabled. If it is found on query cache, it is copied into the local relation cache. If a cache entry is not found on anywhere, Pgpool-II executes the query against PostgreSQL, and the result is stored into the shared relation cache and the local cache.

This parameter can only be set at server start.

`relcache_query_target` (enum)

The target node to send queries to create relation cache entries. If set to `master`, queries will be sent to master (primary) node. This is the default and recommended to most users because the query could get the latest information. If you want to lower the load of master (primary) node, you can set the parameter to `load_balance_node`, which will send queries to the load balance node. This is especially useful for such a system where Pgpool-II/primary server is on a continent A while other Pgpool-II/standby server is on other continent B. Clients on B want read data from the standby because it's much geographically closer. In this case you can set `backend_weight0` (this represents primary) to 0, `backend_weight1` to 1 (this represents standby) and set `relcache_query_target` to `load_balance_node`.

Note, however, if you send query to the standby node, recently created tables and rows might not be available on the standby server yet because of replication delay. Thus such a configuration is not recommended for systems where data modification activity is high.

This parameter can be changed by reloading the Pgpool-II configurations.

`check_temp_table` (enum)

Setting to `catalog` or `trace`, enables the temporary table check in the `SELECT` statements. To check the temporary table Pgpool-II queries the system catalog of primary/master PostgreSQL backend if `catalog` is specified, which increases the load on the primary/master server.

If `trace` is set, Pgpool-II traces temporary table creation and dropping to obtain temporary table info. So no need to access system catalogs. However, if temporary table creation is invisible to Pgpool-II (done in functions or triggers, for example), Pgpool-II cannot recognize the creation of temporary tables.

If you are absolutely sure that your system never uses temporary tables, then you can safely set to `none`.

Note: For a backward compatibility sake for 4.0 or before, Pgpool-II accepts `on`, which is same as `catalog` and `off`, which is same as `none`, they may be deleted in the future version.

Default is `catalog`.

This parameter can be changed by reloading the Pgpool-II configurations. You can also use [PGPOOL SET](#) command to alter the value of this parameter for a current session.

`check_unlogged_table` (boolean)

Setting to on, enables the unlogged table check in the `SELECT` statements. To check the unlogged table Pgpool-II queries the system catalog of primary/master PostgreSQL backend which increases the load on

the primary/master server. If you are absolutely sure that your system never uses the unlogged tables (for example, you are using 9.0 or earlier version of PostgreSQL) then you can safely turn off the `check_unlogged_table`. Default is on.

This parameter can be changed by reloading the Pgpool-II configurations. You can also use [PGPOOL SET](#) command to alter the value of this parameter for a current session.

`pid_file_name` (string)

Specifies the full path to a file to store the Pgpool-II process id. The `pid_file_name` path can be specified as relative to the location of `pgpool.conf` file or as an absolute path. Default is `"/var/run/pgpool/pgpool.pid"`.

This parameter can only be set at server start.

`logdir` (string)

Specifies the full path to a directory to store the `pgpool_status`. Default is `'/tmp'`.

This parameter can only be set at server start.

Chapter 6. Client Authentication

Since Pgpool-II is a middleware that works between PostgreSQL servers and a PostgreSQL database client, so when a client application connects to the Pgpool-II, Pgpool-II in turn connects to the PostgreSQL servers using the same credentials to serve the incoming client connection. Thus, all the access privileges and restrictions defined for the user in PostgreSQL gets automatically applied to all Pgpool-II clients, with an exceptions of the authentications on PostgreSQL side that depends on the client's IP addresses or host names. Reason being the connections to the PostgreSQL server are made by Pgpool-II on behalf of the connecting clients and PostgreSQL server can only see the IP address of the Pgpool-II server and not that of the actual client. Therefore, for the client host based authentications Pgpool-II has the `pool_hba` mechanism similar to the `pg_hba` mechanism for authenticating the incoming client connections.

6.1. The `pool_hba.conf` File

Just like the `pg_hba.conf` file for PostgreSQL, Pgpool-II supports a similar client authentication function using a configuration file called `pool_hba.conf`. If Pgpool-II is installed from source code, it also includes the sample `pool_hba.conf.sample` file in the default configuration directory (`"/usr/local/etc"`). By default, `pool_hba` authentication is disabled, and setting `enable_pool_hba` to `on` enables it. see the [enable_pool_hba](#) configuration parameter.

Note: If number of PostgreSQL servers is only one, or when running in raw mode (see [Section 3.3.2](#)), `pool_hba.conf` is not necessary thus `enable_pool_hba` may be being set to off. In this case the client authentication method is completely managed by PostgreSQL. Also number of PostgreSQL servers is more than one, or not running in raw mode, `enable_pool_hba` may be being set to off as long as the authentication method defined by PostgreSQL is `trust`.

The format of the `pool_hba.conf` file follows very closely PostgreSQL's `pg_hba.conf` format.

The general format of the `pool_hba.conf` file is a set of records, one per line. Blank lines are ignored, as is any text after the `#` comment character. Records cannot be continued across lines. A record is made up of a number of fields which are separated by spaces and/or tabs. Fields can contain white space if the field value is double-quoted. Quoting one of the keywords in a database, user, or address field (e.g., `all` or `replication`) makes the word lose its special meaning, and just match a database, user, or host with that name.

Each record specifies a connection type, a client IP address range (if relevant for the connection type), a database name, a user name, and the authentication method to be used for connections matching these parameters. The first record with a matching connection type, client address, requested database, and user name is used to perform authentication. There is no "fall-through" or "backup": if one record is chosen and the authentication fails, subsequent records are not considered. If no record matches, access is denied.

A record can have one of the following formats

```
local  database user auth-method [auth-options]
host   database user IP-address IP-mask auth-method [auth-options]
hostssl database user IP-address IP-mask auth-method [auth-options]
hostnossl database user IP-address IP-mask auth-method [auth-options]

host   database user address auth-method [auth-options]
hostssl database user address auth-method [auth-options]
hostnossl database user address auth-method [auth-options]
```

The meaning of the fields is as follows:

local

This record matches connection attempts using Unix-domain sockets. Without a record of this type, Unix-domain socket connections are disallowed.

host

This record matches connection attempts made using TCP/IP. *host* records match either SSL or non-SSL connection attempts.

Note: Remote TCP/IP connections will not be possible unless the server is started with an appropriate value for the [listen_addresses](#) configuration parameter, since the default behavior is to listen for TCP/IP connections only on the local loopback address `localhost`.

hostssl

This record matches connection attempts made using TCP/IP, but only when the connection is made with SSL encryption.

To make use of this option the Pgpool-II must be built with SSL support. Furthermore, SSL must be enabled by setting the [ssl](#) configuration parameter. Otherwise, the *hostssl* record is ignored.

hostnossl

This record type has the opposite behavior of *hostssl*; it only matches connection attempts made over TCP/IP that do not use SSL.

database

Specifies which database name(s) this record matches. The value `all` specifies that it matches all databases.

Note: "samegroup" for database field is not supported:

Since Pgpool-II does not know anything about users in the PostgreSQL backend server, the database name is simply compared against the entries in the database field of `pool_hba.conf`.

user

Specifies which database user name(s) this record matches. The value `all` specifies that it matches all users. Otherwise, this is the name of a specific database user

Note: group names following "+" for user field is not supported:

This is for the same reason as for the "samegroup" of database field. A user name is simply checked against the entries in the user field of `pool_hba.conf`.

address

Specifies the client machine address(es) that this record matches. This field can contain either a host name, an IP address range, or one of the special key words mentioned below.

An IP address range is specified using standard numeric notation for the range's starting address, then a slash (/) and a CIDR mask length. The mask length indicates the number of high-order bits of the client IP address that must match. Bits to the right of this should be zero in the given IP address. There must not be any white space between the IP address, the /, and the CIDR mask length.

Typical examples of an IPv4 address range specified this way are `172.20.143.89/32` for a single host, or `172.20.143.0/24` for a small network, or `10.6.0.0/16` for a larger one. An IPv6 address range might look like `::1/128` for a single host (in this case the IPv6 loopback address) or `fe80::7a31:c1ff:0000:0000/96` for a small network. `0.0.0.0/0` represents all IPv4 addresses, and `::0/0` represents all IPv6 addresses. To specify a single host, use a mask length of 32 for IPv4 or 128 for IPv6. In a network address, do not omit trailing zeroes.

An entry given in IPv4 format will match only IPv4 connections, and an entry given in IPv6 format will match only IPv6 connections, even if the represented address is in the IPv4-in-IPv6 range. Note that entries in IPv6 format will be rejected if the system's C library does not have support for IPv6 addresses.

You can also write `all` to match any IP address, `samehost` to match any of the server's own IP addresses, or `samenet` to match any address in any `subnet` that the server is directly connected to.

If a host name is specified (anything that is not an IP address range or a special key word is treated as a host name), that name is compared with the result of a reverse name resolution of the client's IP address (e.g., reverse DNS lookup, if DNS is used). Host name comparisons are case insensitive. If there is a match, then a forward name resolution (e.g., forward DNS lookup) is performed on the host name to check whether any of the addresses it resolves to are equal to the client's IP address. If both directions match, then the entry is considered to match. (The host name that is used in `pool_hba.conf` should be the one that address-to-name resolution of the client's IP address returns, otherwise the line won't be matched. Some host name databases allow associating an IP address with multiple host names, but the operating system will only return one host name when asked to resolve an IP address.)

A host name specification that starts with a dot (.) matches a suffix of the actual host name. So `.example.com` would match `foo.example.com` (but not just `example.com`).

When host names are specified in `pool_hba.conf`, you should make sure that name resolution is reasonably fast. It can be of advantage to set up a local name resolution cache such as `nscd`.

This field only applies to `host`, `hostssl`, and `hostnossl` records.

Specifying the host name in address field is not supported prior to Pgpool-II **V3.7**.

IP-address

IP-mask

These two fields can be used as an alternative to the **IP-address/ mask-length** notation. Instead of specifying the mask length, the actual mask is specified in a separate column. For example, `255.0.0.0` represents an IPv4 CIDR mask length of 8, and `255.255.255.255` represents a CIDR mask length of 32.

This field only applies to `host`, `hostssl`, and `hostnossl` records.

auth-method

Specifies the authentication method to use when a connection matches this record. The possible choices are summarized here; details are in [Section 6.2](#).

trust

Allow the connection unconditionally. This method allows anyone that can connect to the Pgpool-II.

reject

Reject the connection unconditionally. This is useful for "filtering out" certain hosts, for example a `reject` line could block a specific host from connecting.

md5

Require the client to supply a double-MD5-hashed password for authentication.

Note: To use `md5` authentication, you need to register the user name and password in `pool_passwd` file. See [Section 6.2.2](#) for more details. If you don't want to manage password by using `pool_passwd`, you could use [allow_clear_text_frontend_auth](#).

scram-sha-256

Perform SCRAM-SHA-256 authentication to verify the user's password.

Note: To use `scram-sha-256` authentication, you need to register the user name and password in `pool_passwd` file. See [Section 6.2.3](#) for more details. If you don't want to manage password by using `pool_passwd`, you could use [allow_clear_text_frontend_auth](#).

cert

Authenticate using SSL client certificates. See [Section 6.2.4](#) for more details.

pam

Authenticate using the Pluggable Authentication Modules (PAM) service provided by the operating system. See [Section 6.2.5](#) for details.

PAM authentication is supported using user information on the host where `Pgpool-II` is running. To enable PAM support the `Pgpool-II` must be configured with `--with-pam`

To enable PAM authentication, you must create a service-configuration file for `Pgpool-II` in the system's PAM configuration directory (that is usually located at `/etc/pam.d`). A sample service-configuration file is also installed as `share/pgpool.pam` under the install directory.

auth-options

After the **auth-method** field, there can be field(s) of the form **name= value** that specify options for the authentication method.

Since the `pool_hba.conf` records are examined sequentially for each connection attempt, the order of the records is significant. Typically, earlier records will have tight connection match parameters and weaker authentication methods, while later records will have looser match parameters and stronger authentication methods. For example, one might wish to use `trust` authentication for local TCP/IP connections but require a password for remote TCP/IP connections. In this case a record specifying `trust` authentication for connections from 127.0.0.1 would appear before a record specifying password authentication for a wider range of allowed client IP addresses.

Tip: All `pool_hba` authentication options described in this section are about the authentication taking place between a client and the `Pgpool-II`. A client still has to go through the PostgreSQL's authentication process and must have the `CONNECT` privilege for the database on the backend PostgreSQL server.

As far as `pool_hba` is concerned, it does not matter if a user name and/or database name given by a client (i.e. `psql -U testuser testdb`) really exists in the backend. `pool_hba` only

cares if a match in the `pool_hba.conf` can be found or not.

Some examples of `pool_hba.conf` entries. See the next section for details on the different authentication methods.

Example 6-1. Example `pool_hba.conf` Entries

```
# Allow any user on the local system to connect to any database with
# any database user name using Unix-domain sockets (the default for local
# connections).
#
# TYPE DATABASE      USER      ADDRESS      METHOD
local all            all                trust

# The same using local loopback TCP/IP connections.
#
# TYPE DATABASE      USER      ADDRESS      METHOD
host all            all        127.0.0.1/32  trust

# Allow any user from host 192.168.12.10 to connect to database
# "postgres" if the user's password is correctly supplied.
#
# TYPE DATABASE      USER      ADDRESS      METHOD
host postgres       all        192.168.12.10/32  md5
```

6.2. Authentication Methods

The following subsections describe the authentication methods in more detail.

6.2.1. Trust Authentication

When `trust` authentication is specified, Pgpool-II assumes that anyone who can connect to the server is authorized to access connect with whatever database user name they specify.

6.2.2. MD5 Password Authentication

This authentication method is the password-based authentication methods in which MD-5-hashed password is sent by client. Since Pgpool-II does not has the visibility of PostgreSQL's database user password and client application only sends the MD5-hash of the password, so `md5` authentication in Pgpool-II is supported using the `pool_passwd` authentication file.

Note: If Pgpool-II is operated in raw mode or there's only 1 backend configured, you don't need to setup `pool_passwd`.

6.2.2.1. Authentication file format

To use the `md5` authentication `pool_passwd` authentication file must contain the user password in either plain text `md5` or AES encrypted format.

The `pool_passwd` file should contain lines in the following format:

```
"username:plain_text_passwd"
```

```
"username:encrypted_passwd"
```

6.2.2.2. Setting md5 Authentication

here are the steps to enable md5 authentication:

1- Login as the database's operating system user and type "pg_md5 --md5auth --username=username password" user name and md5 encrypted password are registered into [pool_passwd](#). If pool_passwd does not exist yet, pg_md5 command will automatically create it for you.

Note: user name and password must be identical to those registered in PostgreSQL server.

2- Add an appropriate md5 entry to pool_hba.conf. See [Section 6.1](#) for more details.

3- After changing md5 password (in both pool_passwd and PostgreSQL of course), reload the pgpool configurations.

6.2.3. scram-sha-256 Authentication

This authentication method also known as SCRAM is a challenge-response based authentication that prevents the password sniffing on untrusted connections. Since Pgpool-II does not has the visibility of PostgreSQL's database user password, so SCRAM authentication is supported using the [pool_passwd](#) authentication file.

6.2.3.1. Authentication file entry for SCRAM

To use the SCRAM authentication [pool_passwd](#) authentication file must contain the user password in either plain text or AES encrypted format.

```
"username:plain_text_passwd"
```

```
"username:AES_encrypted_passwd"
```

Note: md5 type user passwords in [pool_passwd](#) file can't be used for scram authentication

6.2.3.2. Setting scram-sha-256 Authentication

Here are the steps to enable scram-sha-256 authentication:

1- Create [pool_passwd](#) file entry for database user and password in plain text or AES encrypted format. The

[pg_enc](#) utility that comes with Pgpool-II can be used to create the AES encrypted password entries in the [pool_passwd](#) file.

Note: User name and password must be identical to those registered in the PostgreSQL server.

2- Add an appropriate scram-sha-256 entry to `pool_hba.conf`. See [Section 6.1](#) for more details.

3- After changing SCRAM password (in both `pool_passwd` and PostgreSQL of course), reload the Pgpool-II configuration.

6.2.4. Certificate Authentication

This authentication method uses SSL client certificates to perform authentication. It is therefore only available for SSL connections. When using this authentication method, the Pgpool-II will require that the client provide a valid certificate. No password prompt will be sent to the client. The `cn` (Common Name) attribute of the certificate will be compared to the requested database user name, and if they match the login will be allowed.

Note: The certificate authentication works between only client and Pgpool-II. The certificate authentication does not work between Pgpool-II and PostgreSQL. For backend authentication you can use any other authentication method.

6.2.5. PAM Authentication

This authentication method uses PAM (Pluggable Authentication Modules) as the authentication mechanism. The default PAM service name is `pgpool`. PAM authentication is supported using user information on the host where Pgpool-II is executed. For more information about PAM, please read the [Linux-PAM Page](#).

To enable PAM authentication, you need to create a service-configuration file for Pgpool-II in the system's PAM configuration directory (which is usually at `/etc/pam.d`). A sample service-configuration file is installed as `share/pgpool-II/pgpool.pam` under the install directory.

Note: To enable PAM support the Pgpool-II must be configured with `"--with-pam"`

6.3. Using different methods for frontend and backend authentication

Since Pgpool-II **V4.0** it is possible to use different authentication for client application and backend PostgreSQL servers. For example, a client application can use `scram-sha-256` to connect to Pgpool-II which in turn can use `trust` or `md5` authentication to connect to PostgreSQL backend for the same session.

6.4. Using AES256 encrypted passwords in `pool_passwd`

SCRAM authentication guards against the man-in-the-middle type of attack, so Pgpool-II requires the user password to authenticate with the PostgreSQL backend.

However, storing the clear text passwords in the "pool_passwd" file is not a good idea.

You can instead store AES256 encrypted passwords, which will be used for authentication. The password is first encrypted using the AES256 encryption with the user provided key and then the encrypted password is base64 encoded and an AES prefix is added to the encoded string.

Note: You can use the [pg_enc](#) utility to create the properly formatted AES256 encrypted password.

6.4.1. Creating encrypted password entries

[pg_enc](#) can be used to create AES encrypted password entries in [pool_passwd](#) file. [pg_enc](#) requires the key for encrypting the password entries. Later that same key will be required by Pgpool-II to decrypt the passwords to use for authentication.

Note: Pgpool-II must be built with SSL (--with-openssl) support to use the encrypted password feature.

6.4.2. Providing decryption key to Pgpool-II

If you have AES encrypted passwords stored in the [pool_passwd](#) file, then Pgpool-II will require the decryption key to decrypt the passwords before using them, Pgpool-II tries to read the decryption key at startup from the [.pgpoolkey](#) file.

By default the Pgpool-II will look for the [.pgpoolkey](#) file in the user's home directory or the file referenced by environment variable `PGPOOLKEYFILE`. You can also specify the key file using the `(-k, --key-file=KEY_FILE)` command line argument to the [pgpool](#) command. The permissions on [.pgpoolkey](#) must disallow any access to world or group. Change the file permissions by the command `chmod 0600 ~/.pgpoolkey`.

Chapter 7. Performance Considerations

There are number of configuration parameters that affect the performance of Pgpool-II. In this chapter we present how to tune them.

7.1. Resource Requirement

Pgpool-II does not consume too much resource. However there are minimum requirements for resource. In this section we are going to explain one by one.

7.1.1. Memory Requirement

There are two types of memory usage in Pgpool-II: shared memory and process private memory. The former is allocated at the startup of Pgpool-II main server process and will not be freed until whole Pgpool-II servers shut down. The latter is allocated within each Pgpool-II child process and will be freed at the end of the process.

7.1.1.1. Shared Memory Requirement

Here is a formula to calculate the shared memory requirement.

$$\text{Shared memory requirement (in bytes)} = \text{num_init_children} * \text{max_pool} * 17408$$

For example if you have `num_init_children = 32` (the default) and `max_pool = 4` (the default), then you will need $32 * 4 * 17408 = 2228224$ bytes = 2.1 MB.

If you plan to use in memory query cache (see [Section 5.12](#) for more details) in the shared memory, you will need more RAM for it. See [memqcache_total_size](#) and [memqcache_max_num_cache](#) for required RAM size.

7.1.1.2. Process Memory Requirement

Here is a formula to calculate the process memory requirement.

$$\text{Process memory requirement in total (in mega bytes)} = \text{num_init_children} * 5$$

For example if you have `num_init_children = 32` (the default), you will need 160MB.

7.1.2. Disk Requirement

Pgpool-II does not consume much disk space. Also it does not require high speed disk because disk I/O traffic caused by Pgpool-II is small. However, if you plan to emit much logs, of course you need disk space for them.

7.2. Managing Client Connections

As the number of client connections accepted is growing, the number of Pgpool-II child process which can accept new connections from client is decreasing and finally reaches to 0. In this situation new clients need to wait until a child process becomes free. Under heavy load, it could be possible that the queue length of waiting clients is getting longer and longer and finally hits the system's limit (you might see "535 times the listen queue of a socket overflowed" error"). In this case you need to increase the queue limit. There are several ways to deal with this problem.

7.2.1. Controlling num_init_children

The obvious way to deal with the problem is increasing the number of child process. This can be done by tweaking [num_init_children](#). However increasing child process requires more CPU and memory resource. Also you have to be very careful about `max_connections` parameter of PostgreSQL because once the number of child process is greater than `max_connections`, PostgreSQL refuses to accept new connections, and failover will be triggered.

Another drawback of increasing `num_init_children` is, so called "thundering herd problem". When new connection request comes in, the kernel wake up any sleeping child process to issue `accept()` system call. This triggers fight of process to get the socket and could give heavy load to the system. To mitigate the problem, you could set `serialize_accept` to on so that there's only one process to grab the accepting socket.

7.2.2. Controlling listen_backlog_multiplier

Another solution would be increasing the connection request queue. This could be done by increasing [listen_backlog_multiplier](#).

7.2.3. When to use reserved_connections

However, none of above solutions guarantees that the connection accepting the queue would not be filled up. If a client connection request arrives quicker than the rate of processing queries, the queue will be filled

in someday. For example, if there are some heavy queries that take long time, it could easily trigger the problem.

The solution is setting [reserved_connections](#) so that overflowed connection requests are rejected as PostgreSQL already does. This gives visible errors to applications ("Sorry max_connections already") and force them retrying. So the solution should only be used when you cannot foresee the upper limit of system load.

7.3. Read Query Load Balancing

If there are multiple PostgreSQL nodes and Pgpool-II operates in streaming replication mode, logical replication mode, slony mode or replication mode (for those running mode see [Section 3.3.2](#) for more details), it is possible to distribute read queries among those database nodes to get more throughput since each database nodes processes smaller number of queries. To enable the feature you need to turn on [load_balance_mode](#).

At this point vast majority of systems use streaming replication mode, so from now on we focus on the mode.

7.3.1. Session Level Load Balancing vs. Statement Level Load Balancing

By default load balance mode is "session level" which means the node read queries are sent is determined when a client connects to Pgpool-II. For example, if we have node 0 and node 1, one of the node is selected randomly each time new session is created. In the long term, the possibility which node is chosen will be getting closer to the ratio specified by [backend_weight0](#) and [backend_weight1](#). If those two values are equal, the chance each node is chosen will be even.

On the other hand, if [statement_level_load_balance](#) is set to on, the load balance node is determined at the time each query starts. This is useful in case that application has its own connection pooling which keeps on connecting to Pgpool-II and the load balance node will not be changed once the application starts. Another use case is a batch application. It issues tremendous number of queries but there's only 1 session. With statement level load balancing it can utilize multiple servers.

7.3.2. Creating Specific Purpose Database Node

In OLAP environment sometimes it is desirable to have a large read-only database for specific purpose. By creating such a database is possible by creating a replica database using streaming replication. In this case it is possible to redirect read queries to the database in two ways: specifying database names(s) or specifying application name(s). For former, use [database_redirect_preference_list](#). For latter use [app_name_redirect_preference_list](#).

7.4. In Memory Query Caching

Pgpool-II allows to cache read query results for later use. This will bring huge benefit for a type of applications which issue same read queries many times. If there are two queries and the query strings (parameter for prepared statements if any) are identical, two queries are regarded as "same". For the first time the query is sent, Pgpool-II saves the query result, and use it for the second query without asking anything to PostgreSQL. This technique is explained in [Section 5.12](#).

7.4.1. When not to Use in Memory Query Caching

When a table is modified, query results against the table could be changed. To avoid inconsistency, Pgpool-II discards query cache data when corresponding table is modified. So frequently updated database will not be suitable to use in memory query caching. You can check if your database is suitable to use query caching or not, you could use [SHOW POOL_CACHE](#). If query cache hit ration is lower than 70%, probably you want to avoid using the query cache.

7.5. Relation Cache

Except in raw mode (see [Section 3.3.2](#)) or [load_balance_mode](#) is set to off, sometimes Pgpool-II needs to ask PostgreSQL to get meta information, such as whether a table is a temporary one or not. To get those information, Pgpool-II sends queries primary PostgreSQL which could be up to as many as 10 queries (in 4.1 or after, the number of queries has been decreased, it is not zero, however). To reduce the overhead, Pgpool-II maintains "relation cache". Next time same table is included in a query, Pgpool-II extracts the information from the cache.

There are some parameters to configure the relation cache. See [relcache_expire](#), [relcache_size](#), [check_temp_table](#), [check_unlogged_table](#) for more details.

7.5.1. Shared Relation Cache

The relation cache basically lives in process private memory, which is bound to a process. So even if a relation cache is created for a table, in different process the relation cache might not be created yet. After all, until a relation cache entry is created in all process, queries continue to sent to PostgreSQL. Pgpool-II 4.1 overcomes the issue by creating relation cache in shared memory. If a session creates a relation cache entry in the shared memory, other sessions will get the cache result by looking at the shared relation cache. See [enable_shared_relcache](#) configuration parameter section for more details. This feature is pretty effective and we recommend this feature be enabled.

7.6. Other Performance Considerations

This section introduces some other performance considerations.

7.6.1. Thundering Herd Problem

If [num_init_children](#) is large, it is possible that many Pgpool-II process are woke up and heavy context switching happens. This leads to high system load and hurt the overall system performance. This problem is called "the thundering herd problem". Enabling [serialize_accept](#) could solve the problem. Please note that for smaller [num_init_children](#), [serialize_accept](#) might make the system performance worse. Please take a look at the guidance in [serialize_accept](#) section.

7.6.2. Disaster recovery settings

To create a disaster recovery setting, it is possible to deploy a Pgpool-II plus PostgreSQL primary server, and another Pgpool-II plus standby PostgreSQL server in a geographically distant place. Clients close to the standby server send read only queries to the Pgpool-II, being close to the standby server. However, since standby Pgpool-II sends internal queries to system catalog of primary PostgreSQL server, query performance may be getting worse. To avoid the problem, it is possible to use [relcache_query_target](#) so that such queries are sent to the standby. See [relcache_query_target](#) for more details.

III. Examples

Various examples

Table of Contents

8. [Configuration Examples](#)

- 8.1. [Basic Configuration Example](#)
 - 8.2. [Watchdog Configuration Example](#)
 - 8.3. [Pgpool-II + Watchdog Setup Example](#)
 - 8.4. [AWS Configuration Example](#)
 - 8.5. [Aurora Configuration Example](#)
-

Chapter 8. Configuration Examples

8.1. Basic Configuration Example

8.1.1. Let's Begin!

First, we must learn how to install and configure Pgpool-II and database nodes before using replication.

8.1.1.1. Installing Pgpool-II

Installing Pgpool-II is very easy. In the directory which you have extracted the source tar ball, execute the following commands.

```
$ ./configure
$ make
$ make install
```

`configure` script collects your system information and use it for the compilation procedure. You can pass command line arguments to `configure` script to change the default behavior, such as the installation directory. Pgpool-II will be installed to `/usr/local` directory by default.

`make` command compiles the source code, and `make install` will install the executables. You must have write permission on the installation directory. In this tutorial, we will install Pgpool-II in the default `/usr/local` directory.

Note: Pgpool-II requires `libpq` library in PostgreSQL 7.4 or later (version 3 protocol).

If the `configure` script displays the following error message, the `libpq` library may not be installed, or it is not of version 3

```
configure: error: libpq is not installed or libpq is old
```

If the library is version 3, but the above message is still displayed, your `libpq` library is probably not recognized by the `configure` script. The `configure` script searches for `libpq` library under `/usr/local/pgsql`. If you have installed the PostgreSQL in a directory other than `/usr/local/pgsql`, use `--with-pgsql`, or `--with-pgsql-includedir` and `--with-pgsql-libdir` command line options when you execute `configure`.

8.1.1.2. Configuration Files

Pgpool-II configuration parameters are saved in the `pgpool.conf` file. The file is in "parameter = value" per line format. When you install Pgpool-II, `pgpool.conf.sample` is automatically created. We recommend copying and renaming it to `pgpool.conf`, and edit it as you like.

```
$ cp /usr/local/etc/pgpool.conf.sample /usr/local/etc/pgpool.conf
```

Pgpool-II only accepts connections from the localhost using port 9999 by the default. If you wish to receive connections from other hosts, set [listen_addresses](#) to `'*'`.

```
listen_addresses = 'localhost'
port = 9999
```

We will use the default parameters in this tutorial.

8.1.1.3. Configuring PCP Commands

Pgpool-II has an interface for administrative purpose to retrieve information on database nodes, shutdown Pgpool-II, etc. via network. To use PCP commands, user authentication is required. This authentication is different from PostgreSQL's user authentication. A user name and password need to be defined in the `pcp.conf` file. In the file, a user name and password are listed as a pair on each line, and they are separated by a colon (:). Passwords are encrypted in md5 hash format.

```
postgres:e8a48653851e28c69d0506508fb27fc5
```

When you install Pgpool-II, `pcp.conf.sample` is automatically created. We recommend copying and renaming it to `pcp.conf`, and edit it.

```
$ cp /usr/local/etc/pcp.conf.sample /usr/local/etc/pcp.conf
```

To encrypt your password into md5 hash format, use the `pg_md5` command, which is installed as one of Pgpool-II's executables. `pg_md5` takes text as a command line argument, and displays its md5-hashed text. For example, give "postgres" as the command line argument, and `pg_md5` displays md5-hashed text on its standard output.

```
$ /usr/local/bin/pg_md5 postgres  
e8a48653851e28c69d0506508fb27fc5
```

PCP commands are executed via network, so the port number must be configured with `pcp_port` parameter in `pgpool.conf` file. We will use the default 9898 for `pcp_port` in this tutorial.

```
pcp_port = 9898
```

8.1.1.4. Preparing Database Nodes

Now, we need to set up backend PostgreSQL servers for Pgpool-II. These servers can be placed within the same host as Pgpool-II, or on separate machines. If you decide to place the servers on the same host, different port numbers must be assigned for each server. If the servers are placed on separate machines, they must be configured properly so that they can accept network connections from Pgpool-II.

```
backend_hostname0 = 'localhost'  
backend_port0 = 5432  
backend_weight0 = 1  
backend_hostname1 = 'localhost'  
backend_port1 = 5433  
backend_weight1 = 1  
backend_hostname2 = 'localhost'  
backend_port2 = 5434  
backend_weight2 = 1
```

For `backend_hostname`, `backend_port`, `backend_weight`, set the node's hostname, port number, and ratio for load balancing. At the end of each parameter string, node ID must be specified by adding positive integers starting with 0 (i.e. 0, 1, 2..).

Note: `backend_weight` parameters for all nodes are set to 1, meaning that SELECT queries are equally distributed among three servers.

8.1.1.5. Starting/Stopping Pgpool-II

To fire up Pgpool-II, execute the following command on a terminal.

```
$ pgpool
```

The above command, however, prints no log messages because Pgpool-II detaches the terminal. If you want to show Pgpool-II log messages, you pass `-n` option to `pgpool` command so Pgpool-II is executed as non-daemon process, and the terminal will not be detached.

```
$ pgpool -n &
```

The log messages are printed on the terminal, so it is recommended to use the following options.

```
$ pgpool -n -d > /tmp/pgpool.log 2>&1 &
```

The `-d` option enables debug messages to be generated. The above command keeps appending log messages to `/tmp/pgpool.log`. If you need to rotate log files, pass the logs to a external command which has log rotation function. For example, you can use [rotatelogs](#) from Apache2:

```
$ pgpool -n 2>&1 | /usr/local/apache2/bin/rotatelogs \  
-l -f /var/log/pgpool/pgpool.log.%A 86400 &
```

This will generate a log file named "`pgpool.log.Thursday`" then rotate it 00:00 at midnight. Rotatelogs adds logs to a file if it already exists. To delete old log files before rotation, you could use cron:

```
55 23 * * * /usr/bin/find /var/log/pgpool -type f -mtime +5 -exec /bin/rm -f '{}';
```

Please note that rotatelogs may exist as `/usr/sbin/rotatelogs2` in some distributions. `-f` option generates a log file as soon as `rotatelogs` starts and is available in apache2 2.2.9 or greater. Also [cronolog](#) can be used.

```
$ pgpool -n 2>&1 | /usr/sbin/cronolog \  
--hardlink=/var/log/pgsql/pgpool.log \  
'/var/log/pgsql/%Y-%m-%d-pgpool.log' &
```

To stop Pgpool-II execute the following command.

```
$ pgpool stop
```

If any client is still connected, Pgpool-II waits for it to disconnect, and then terminates itself. Run the following command instead if you want to shutdown Pgpool-II forcibly.

```
$ pgpool -m fast stop
```

8.1.2. Your First Replication

Replication (see [Section 5.3.2](#)) enables the same data to be copied to multiple database nodes. In this section, we'll use three database nodes, which we have already set up in [Section 8.1.1](#), and takes you step by step to create a database replication system. Sample data to be replicated will be generated by the

[pgbench](#) benchmark program.

8.1.2.1. Configuring Replication

To enable the database replication function, set [replication_mode](#) to on in `pgpool.conf` file.

```
replication_mode = true
```

When [replication_mode](#) is on, Pgpool-II will send a copy of a received query to all the database nodes. In addition, when [load_balance_mode](#) is set to true, Pgpool-II will distribute SELECT queries among the database nodes.

```
load_balance_mode = true
```

In this section, we will enable both [replication_mode](#) and [load_balance_mode](#).

8.1.2.2. Checking Replication

To reflect the above changes in `pgpool.conf`, Pgpool-II must be restarted. Please refer to "Starting/Stopping Pgpool-II" [Section 8.1.1.5](#). After configuring `pgpool.conf` and restarting the Pgpool-II, let's try the actual replication and see if everything is working. First, we need to create a database to be replicated. We will name it "bench_replication". This database needs to be created on all the nodes. Use the [createdb](#) commands through Pgpool-II, and the database will be created on all the nodes.

```
$ createdb -p 9999 bench_replication
```

Then, we'll execute [pgbench](#) with `-i` option. `-i` option initializes the database with pre-defined tables and data.

```
$ pgbench -i -p 9999 bench_replication
```

The following table is the summary of tables and data, which will be created by [pgbench -i](#). If, on all the nodes, the listed tables and data are created, replication is working correctly.

Table 8-1. data summary

Table Name	Number of Rows
pgbench_branches	1
pgbench_tellers	10
pgbench_accounts	100000
pgbench_history	0

Let's use a simple shell script to check the above on all the nodes. The following script will display the number of rows in `pgbench_branches`, `pgbench_tellers`, `pgbench_accounts`, and `pgbench_history` tables on all the nodes (5432, 5433, 5434).

```
$ for port in 5432 5433 5434; do
> echo $port
> for table_name in pgbench_branches pgbench_tellers pgbench_accounts pgbench_history; do
>   echo $table_name
>   psql -c "SELECT count(*) FROM $table_name" -p $port bench_replication
> done
> done
```

8.2. Watchdog Configuration Example

This tutorial explains the simple way to try "Watchdog". What you need is 2 Linux boxes on which Pgpool-II is installed and a PostgreSQL on the same machine or in the other one. It is enough that 1 node for backend exists. You can use watchdog with Pgpool-II in any mode: replication mode, master/slave mode and raw mode.

This example uses "osspec16" as an Active node and "osspec20" as a Standby node. "Someserver" means one of them.

8.2.1. Common configurations

Set the following parameters in both of active and standby nodes.

8.2.1.1. Enabling watchdog

First of all, set [use_watchdog](#) to on.

```
use_watchdog = on
# Activates watchdog
```

8.2.1.2. Configure Up stream servers

Specify the up stream servers (e.g. application servers). Leaving it blank is also fine.

```
trusted_servers = ""
# trusted server list which are used
# to confirm network connection
# (hostA,hostB,hostC,...)
```

8.2.1.3. Watchdog Communication

Specify the TCP port number for watchdog communication.

```
wd_port = 9000
# port number for watchdog service
```

8.2.1.4. Virtual IP

Specify the IP address to be used as a virtual IP address in the [delegate_IP](#).

```
delegate_IP = '133.137.177.143'
# delegate IP address
```

Note: Make sure the IP address configured as a Virtual IP should be free and is not used by any other machine.

8.2.2. Individual Server Configurations

Next, set the following parameters for each Pgpool-II. Specify [other_pgpool_hostname](#), [other_pgpool_port](#) and [other_wd_port](#) with the values of other Pgpool-II server values.

8.2.2.1. Active (osspc16) Server configurations

```
other_pgpool_hostname0 = 'osspc20'  
# Host name or IP address to connect to for other pgpool 0  
other_pgpool_port0 = 9999  
# Port number for other pgpool 0  
other_wd_port0 = 9000  
# Port number for other watchdog 0
```

8.2.2.2. Standby (osspc20) Server configurations

```
other_pgpool_hostname0 = 'osspc16'  
# Host name or IP address to connect to for other pgpool 0  
other_pgpool_port0 = 9999  
# Port number for other pgpool 0  
other_wd_port0 = 9000  
# Port number for other watchdog 0
```

8.2.3. Starting Pgpool-II

Start Pgpool-II on each servers from root user with "-n" switch and redirect log messages into pgpool.log file.

8.2.3.1. Starting pgpool in Active server (osspc16)

First start the Pgpool-II on Active server.

```
[user@osspc16]$ su -  
[root@osspc16]# {installed_dir}/bin/pgpool -n -f {installed_dir}/etc/pgpool.conf > pgpool.log 2>&1
```

Log messages will show that Pgpool-II has the virtual IP address and starts watchdog process.

```
LOG: I am announcing my self as master/coordinator watchdog node  
LOG: I am the cluster leader node  
DETAIL: our declare coordinator message is accepted by all nodes  
LOG: I am the cluster leader node. Starting escalation process  
LOG: escalation process started with PID:59449  
LOG: watchdog process is initialized  
LOG: watchdog: escalation started  
LOG: I am the master watchdog node  
DETAIL: using the local backend node status
```

8.2.3.2. Starting pgpool in Standby server (osspc20)

Now start the Pgpool-II on Standby server.

```
[user@osspc20]$ su -  
[root@osspc20]# {installed_dir}/bin/pgpool -n -f {installed_dir}/etc/pgpool.conf > pgpool.log 2>&1
```

Log messages will show that Pgpool-II has joined the watchdog cluster as standby watchdog.

```
LOG: watchdog cluster configured with 1 remote nodes
LOG: watchdog remote node:0 on Linux_osspec16_9000:9000
LOG: interface monitoring is disabled in watchdog
LOG: IPC socket path: "/tmp/.s.PGPOOLWD_CMD.9000"
LOG: watchdog node state changed from [DEAD] to [LOADING]
LOG: new outbound connection to Linux_osspec16_9000:9000
LOG: watchdog node state changed from [LOADING] to [INITIALIZING]
LOG: watchdog node state changed from [INITIALIZING] to [STANDBY]
LOG: successfully joined the watchdog cluster as standby node
DETAIL: our join coordinator request is accepted by cluster leader node "Linux_osspec16_9000"
LOG: watchdog process is initialized
```

8.2.4. Try it out

Confirm to ping to the virtual IP address.

```
[user@someserver]$ ping 133.137.177.142
PING 133.137.177.143 (133.137.177.143) 56(84) bytes of data.
64 bytes from 133.137.177.143: icmp_seq=1 ttl=64 time=0.328 ms
64 bytes from 133.137.177.143: icmp_seq=2 ttl=64 time=0.264 ms
64 bytes from 133.137.177.143: icmp_seq=3 ttl=64 time=0.412 ms
```

Confirm if the Active server which started at first has the virtual IP address.

```
[root@osspec16]# ifconfig
eth0    ...

eth0:0  inet addr:133.137.177.143 ...

lo      ...
```

Confirm if the Standby server which started not at first doesn't have the virtual IP address.

```
[root@osspec20]# ifconfig
eth0    ...

lo      ...
```

Try to connect PostgreSQL by "psql -h delegate_IP -p port".

```
[user@someserver]$ psql -h 133.137.177.142 -p 9999 -l
```

8.2.5. Switching virtual IP

Confirm how the Standby server works when the Active server can't provide its service. Stop Pgpool-II on the Active server.

```
[root@osspec16]# {installed_dir}/bin/pgpool stop
```

Then, the Standby server starts to use the virtual IP address. Log shows:

```
LOG: remote node "Linux_osspec16_9000" is shutting down  
LOG: watchdog cluster has lost the coordinator node
```

```
LOG: watchdog node state changed from [STANDBY] to [JOINING]  
LOG: watchdog node state changed from [JOINING] to [INITIALIZING]  
LOG: I am the only alive node in the watchdog cluster  
HINT: skipping stand for coordinator state  
LOG: watchdog node state changed from [INITIALIZING] to [MASTER]  
LOG: I am announcing my self as master/coordinator watchdog node  
LOG: I am the cluster leader node  
DETAIL: our declare coordinator message is accepted by all nodes  
LOG: I am the cluster leader node. Starting escalation process  
LOG: watchdog: escalation started  
  
LOG: watchdog escalation process with pid: 59551 exit with SUCCESS.
```

Confirm to ping to the virtual IP address.

```
[user@someserver]$ ping 133.137.177.142  
PING 133.137.177.143 (133.137.177.143) 56(84) bytes of data.  
64 bytes from 133.137.177.143: icmp_seq=1 ttl=64 time=0.328 ms  
64 bytes from 133.137.177.143: icmp_seq=2 ttl=64 time=0.264 ms  
64 bytes from 133.137.177.143: icmp_seq=3 ttl=64 time=0.412 ms
```

Confirm that the Active server doesn't use the virtual IP address any more.

```
[root@osspec16]# ifconfig  
eth0  ...  
  
lo    ...
```

Confirm that the Standby server uses the virtual IP address.

```
[root@osspec20]# ifconfig  
eth0  ...  
  
eth0:0  inet addr:133.137.177.143 ...  
  
lo    ...
```

Try to connect PostgreSQL by "psql -h delegate_IP -p port".

```
[user@someserver]$ psql -h 133.137.177.142 -p 9999 -l
```

8.2.6. More

8.2.6.1. Lifeclock

There are the parameters about watchdog's monitoring. Specify the interval to check [wd_interval](#) and the type of lifeclock [wd_lifeclock_method](#). The `heartbeat` method specify the time to detect a fault [wd_heartbeat_deadtime](#), the port number to receive [wd_heartbeat_port](#), the interval to send [wd_heartbeat_keepalive](#), the IP address or hostname of destination [heartbeat_destination<emphasis>0</emphasis>](#) and finally the destination port number [heartbeat_destination_port<emphasis>0</emphasis>](#).

```

wd_lifecycle_method = 'heartbeat'
# Method of watchdog lifecycle ('heartbeat' or 'query' or 'external')
# (change requires restart)
wd_interval = 10
# lifecycle interval (sec) > 0
wd_heartbeat_port = 9694
# Port number for receiving heartbeat signal
# (change requires restart)
wd_heartbeat_keepalive = 2
# Interval time of sending heartbeat signal (sec)
# (change requires restart)
wd_heartbeat_deadtime = 30
# Deadtime interval for heartbeat signal (sec)
# (change requires restart)
heartbeat_destination0 = 'host0_ip1'
# Host name or IP address of destination 0
# for sending heartbeat signal.
# (change requires restart)
heartbeat_destination_port0 = 9694
# Port number of destination 0 for sending
# heartbeat signal. Usually this is the
# same as wd_heartbeat_port.
# (change requires restart)

```

8.2.6.2. Switching virtual IP address

There are the parameters for switching the virtual IP address. Specify switching commands [if_up_cmd](#), [if_down_cmd](#), the path to them [if_cmd_path](#), the command executed after switching to send ARP request [arping_cmd](#) and the path to it [arping_path](#).

```

if_cmd_path = '/sbin'
# path to the directory where if_up/down_cmd exists
if_up_cmd = 'ip addr add $_IP_$/24 dev eth0 label eth0:0'
# startup delegate IP command
if_down_cmd = 'ip addr del $_IP_$/24 dev eth0'
# shutdown delegate IP command

arping_path = '/usr/sbin'      # arping command path

arping_cmd = 'arping -U $_IP_$ -w 1'

```

You can also bring up and bring down the virtual IP using arbitrary scripts specified [wd_escalation_command](#) and [wd_de_escalation_command](#) parameters.

8.3. Pgpool-II + Watchdog Setup Example

This section shows an example of streaming replication configuration using Pgpool-II. In this example, we use 3 Pgpool-II servers to manage PostgreSQL servers to create a robust cluster system and avoid the single point of failure or split brain.

PostgreSQL 11 is used in this configuration example, all scripts have also been tested with PostgreSQL 12.

8.3.1. Requirements

We assume that all the Pgpool-II servers and the PostgreSQL servers are in the same subnet.

8.3.2. Cluster System Configuration

We use 3 servers with CentOS 7.4. Let these servers be server1, server2, server3. We install PostgreSQL and Pgpool-II on each server.

Figure 8-1. Cluster System Configuration

Note: The roles of Active, Standby, Primary, Standby are not fixed and may be changed by further operations.

Table 8-2. Hostname and IP address

Hostname	IP Address	Virtual IP
server1	192.168.137.101	
server2	192.168.137.102	192.168.137.150
server3	192.168.137.103	

Table 8-3. PostgreSQL version and Configuration

Item	Value	Detail
PostgreSQL Version	11.1	-
port	5432	-
\$PGDATA	/var/lib/pgsql/11/data	-
Archive mode	on	/var/lib/pgsql/archivedir
Replication Slots	Enable	-
Start automatically	Disable	-

Table 8-4. Pgpool-II version and Configuration

Item	Value	Detail
Pgpool-II Version	4.1.0	-
port	9999	Pgpool-II accepts connections
	9898	PCP process accepts connections
	9000	watchdog accepts connections
	9694	UDP port for receiving Watchdog's heartbeat signal
Config file	/etc/pgpool-II/pgpool.conf	Pgpool-II config file
Pgpool-II start user	postgres (Pgpool-II 4.1 or later)	Pgpool-II 4.0 or before, the default startup user is root
Running mode	streaming replication mode	-
Watchdog	on	Life check method: heartbeat
Start automatically	Disable	-

8.3.3. Installation

In this example, we install Pgpool-II 4.1 and PostgreSQL 11.1 by using RPM packages.

Install PostgreSQL by using PostgreSQL YUM repository.

```
# yum install https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86_64/pgdg-centos11-11-2.noarch.rpm
# yum install postgresql11 postgresql11-libs postgresql11-devel postgresql11-server
```

Install Pgpool-II by using Pgpool-II YUM repository.

```
# yum install http://www.pgpool.net/yum/rpms/4.1/redhat/rhel-7-x86_64/pgpool-II-release-4.1-1.noarch.rpm
# yum install pgpool-II-pg11-*
```

8.3.4. Before Starting

Before you start the configuration process, please check the following prerequisites.

- Set up PostgreSQL streaming replication on the primary server. In this example, we use WAL archiving.

First, we create the directory `/var/lib/pgsql/archivedir` to store WAL segments on all servers. In this example, only Primary node archives WAL locally.

```
[all servers]# su - postgres
[all servers]$ mkdir /var/lib/pgsql/archivedir
```

Then we edit the configuration file `$PGDATA/postgresql.conf` on `server1` (primary) as follows. Enable `wal_log_hints` to use `pg_rewind`. Since the Primary may become a Standby later, we set `hot_standby = on`.

```
listen_addresses = '*'
archive_mode = on
archive_command = 'cp "%p" "/var/lib/pgsql/archivedir/%f"'
max_wal_senders = 10
max_replication_slots = 10
wal_level = replica
hot_standby = on
wal_log_hints = on
```

We use the online recovery functionality of Pgpool-II to setup standby server after the primary server is started.

- Because of the security reasons, we create a user `repl` solely used for replication purpose, and a user `pgpool` for streaming replication delay check and health check of Pgpool-II.

Table 8-5. Users

User Name	Password	Detail
repl	repl	PostgreSQL replication user
pgpool	pgpool	Pgpool-II health check and replication delay check user
postgres	postgres	User running online recovery

```
[server1]# psql -U postgres -p 5432
postgres=# SET password_encryption = 'scram-sha-256';
postgres=# CREATE ROLE pgpool WITH LOGIN;
postgres=# CREATE ROLE repl WITH REPLICATION LOGIN;
postgres=# \password pgpool
postgres=# \password repl
postgres=# \password postgres
```

If you want to show "replication_state" and "replication_sync_state" column in [SHOW POOL NODES](#) command result, role `pgpool` needs to be PostgreSQL super user or or in `pg_monitor` group (Pgpool-II 4.1 or later). Grant `pg_monitor` to `pgpool`:

```
GRANT pg_monitor TO pgpool;
```

Note: If you plan to use [detach_false_primary](#)(Pgpool-II 4.0 or later), role "pgpool" needs to be PostgreSQL super user or or in "pg_monitor" group to use this feature.

Assuming that all the Pgpool-II servers and the PostgreSQL servers are in the same subnet and edit `pg_hba.conf` to enable `scram-sha-256` authentication method.

```
host all all samenet scram-sha-256
host replication all samenet scram-sha-256
```

- To use the automated failover and online recovery of Pgpool-II, the settings that allow **passwordless** SSH to all backend servers between Pgpool-II execution user (default root user) and `postgres` user and between `postgres` user and `postgres` user are necessary. Execute the following command on all servers to set up passwordless SSH. The generated key file name is `id_rsa_pgpool`.

```
[all servers]# cd ~/.ssh
[all servers]# ssh-keygen -t rsa -f id_rsa_pgpool
[all servers]# ssh-copy-id -i id_rsa_pgpool.pub postgres@server1
[all servers]# ssh-copy-id -i id_rsa_pgpool.pub postgres@server2
[all servers]# ssh-copy-id -i id_rsa_pgpool.pub postgres@server3

[all servers]# su - postgres
[all servers]$ cd ~/.ssh
[all servers]$ ssh-keygen -t rsa -f id_rsa_pgpool
[all servers]$ ssh-copy-id -i id_rsa_pgpool.pub postgres@server1
[all servers]$ ssh-copy-id -i id_rsa_pgpool.pub postgres@server2
[all servers]$ ssh-copy-id -i id_rsa_pgpool.pub postgres@server3
```

After setting, use `ssh postgres@serverX -i ~/.ssh/id_rsa_pgpool` command to make sure that you can log in without entering a password. Edit `/etc/ssh/sshd_config` if necessary and restart `sshd`.

- To allow `repl` user without specifying password for streaming replication and online recovery, and execute `pg_rewind` using `postgres`, we create the `.pgpass` file in `postgres` user's home directory and change the permission to `600` on each PostgreSQL server.

```
[all servers]# su - postgres
[all servers]$ vi /var/lib/pgsql/.pgpass
server1:5432:replication:repl:<repl user password>
server2:5432:replication:repl:<repl user password>
server3:5432:replication:repl:<repl user password>
server1:5432:postgres:postgres:<postgres user password>
server2:5432:postgres:postgres:<postgres user password>
server3:5432:postgres:postgres:<postgres user password>
[all servers]$ chmod 600 /var/lib/pgsql/.pgpass
```

- When connect to Pgpool-II and PostgreSQL servers, the target port must be accessible by enabling firewall management softwares. Following is an example for CentOS/RHEL7.

```
[all servers]# firewall-cmd --permanent --zone=public --add-service=postgresql
[all servers]# firewall-cmd --permanent --zone=public --add-port=9999/tcp --add-port=9898/tcp --add-port=9000/tcp --a
[all servers]# firewall-cmd --reload
```

8.3.5. Pgpool-II Configuration

8.3.5.1. Common Settings

Here are the common settings on `server1`, `server2` and `server3`.

When installing Pgpool-II from RPM, all the Pgpool-II configuration files are in `/etc/pgpool-II`. In this example, we copy the sample configuration file for streaming replication mode.

```
# cp -p /etc/pgpool-II/pgpool.conf.sample-stream /etc/pgpool-II/pgpool.conf
```

To allow Pgpool-II to accept all incoming connections, we set `listen_addresses = '*'`.

```
listen_addresses = '*'
```

Specify replication delay check user and password. In this example, we leave `sr_check_user` empty, and create the entry in `pool_passwd`. From Pgpool-II 4.0, if these parameters are left blank, Pgpool-II will first try to get the password for that specific user from `sr_check_password` file before using the empty password.

```
sr_check_user = 'pgpool'
sr_check_password = ''
```

Enable health check so that Pgpool-II performs failover. Also, if the network is unstable, the health check fails even though the backend is running properly, failover or degenerate operation may occur. In order to prevent such incorrect detection of health check, we set `health_check_max_retries = 3`. Specify `health_check_user` and `health_check_password` in the same way like `sr_check_user` and `sr_check_password`.

```
health_check_period = 5
# Health check period
# Disabled (0) by default
health_check_timeout = 30
# Health check timeout
# 0 means no timeout
health_check_user = 'pgpool'
health_check_password = ''

health_check_max_retries = 3
```

Specify the PostgreSQL backend information. Multiple backends can be specified by adding a number at the end of the parameter name.

```
# - Backend Connection Settings -

backend_hostname0 = 'server1'
# Host name or IP address to connect to for backend 0
backend_port0 = 5432
# Port number for backend 0
backend_weight0 = 1
# Weight for backend 0 (only in load balancing mode)
backend_data_directory0 = '/var/lib/pgsql/11/data'
# Data directory for backend 0
backend_flag0 = 'ALLOW_TO_FAILOVER'
# Controls various backend behavior
# ALLOW_TO_FAILOVER or DISALLOW_TO_FAILOVER
backend_hostname1 = 'server2'
backend_port1 = 5432
backend_weight1 = 1
backend_data_directory1 = '/var/lib/pgsql/11/data'
backend_flag1 = 'ALLOW_TO_FAILOVER'

backend_hostname2 = 'server3'
backend_port2 = 5432
backend_weight2 = 1
backend_data_directory2 = '/var/lib/pgsql/11/data'
backend_flag2 = 'ALLOW_TO_FAILOVER'
```

To show "replication_state" and "replication_sync_state" column in [SHOW POOL NODES](#) command result, [backend_application_name](#) parameter is required. Here we specify each backend's hostname in these parameters. (Pgpool-II 4.1 or later)

```
...
backend_application_name0 = 'server1'
...
backend_application_name1 = 'server2'
...
backend_application_name2 = 'server3'
```

8.3.5.2. Failover configuration

Specify failover.sh script to be executed after failover in `failover_command` parameter. If we use 3 PostgreSQL servers, we need to specify `follow_master_command` to run after failover on the primary node failover. In case of two PostgreSQL servers, `follow_master_command` setting is not necessary.

Pgpool-II replaces the following special characters with the backend specific information while executing the scripts. See [failover_command](#) for more details about each character.

```
failover_command = '/etc/pgpool-II/failover.sh %d %h %p %D %m %H %M %P %r %R %N %S'
follow_master_command = '/etc/pgpool-II/follow_master.sh %d %h %p %D %m %H %M %P %r %R'
```

Note: %N and %S are added in Pgpool-II 4.1. Please note that these characters cannot be specified if using Pgpool-II 4.0 or earlier.

Create `/etc/pgpool-II/failover.sh`, and add execute permission.

```
# vi /etc/pgpool-II/failover.sh
# vi /etc/pgpool-II/follow_master.sh
# chmod +x /etc/pgpool-II/{failover.sh,follow_master.sh}
```

- `/etc/pgpool-II/failover.sh`

```
#!/bin/bash
# This script is run by failover_command.

set -o xtrace
exec > >(logger -i -p local1.info) 2>&1

# Special values:
# %d = failed node id
# %h = failed node hostname
# %p = failed node port number
# %D = failed node database cluster path
# %m = new master node id
# %H = new master node hostname
# %M = old master node id
# %P = old primary node id
# %r = new master port number
# %R = new master database cluster path
# %N = old primary node hostname
# %S = old primary node port number
# %% = '%' character

FAILED_NODE_ID="$1"
FAILED_NODE_HOST="$2"
FAILED_NODE_PORT="$3"
FAILED_NODE_DBDATA="$4"
```

```

FAILED_NODE_PGDATA="$4"
NEW_MASTER_NODE_ID="$5"
NEW_MASTER_NODE_HOST="$6"
OLD_MASTER_NODE_ID="$7"
OLD_PRIMARY_NODE_ID="$8"
NEW_MASTER_NODE_PORT="$9"
NEW_MASTER_NODE_PGDATA="${10}"
OLD_PRIMARY_NODE_HOST="${11}"
OLD_PRIMARY_NODE_PORT="${12}"

PGHOME=/usr/pgsql-11

logger -i -p local1.info failover.sh: start: failed_node_id=$FAILED_NODE_ID old_primary_node_id=$OLD_PRIMARY_NODE_ID f

## If there's no master node anymore, skip failover.
if [ $NEW_MASTER_NODE_ID -lt 0 ]; then
    logger -i -p local1.info failover.sh: All nodes are down. Skipping failover.
    exit 0
fi

## Test passwordless SSH
ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@${NEW_MASTER_NODE_HOST} -i ~/.ssh/id_

if [ $? -ne 0 ]; then
    logger -i -p local1.info failover.sh: passwordless SSH to postgres@${NEW_MASTER_NODE_HOST} failed. Please setup pas
    exit 1
fi

## If Standby node is down, skip failover.
if [ $FAILED_NODE_ID -ne $OLD_PRIMARY_NODE_ID ]; then
    logger -i -p local1.info failover.sh: Standby node is down. Skipping failover.

    ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@$OLD_PRIMARY_NODE_HOST -i ~/.ssh/id_
    ${PGHOME}/bin/psql -p $OLD_PRIMARY_NODE_PORT -c "SELECT pg_drop_replication_slot('${FAILED_NODE_HOST}');"

    if [ $? -ne 0 ]; then
        logger -i -p local1.error failover.sh: drop replication slot "${FAILED_NODE_HOST}" failed
        exit 1
    fi

    exit 0
fi

## Promote Standby node.
logger -i -p local1.info failover.sh: Primary node is down, promote standby node ${NEW_MASTER_NODE_HOST}.

ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null \
postgres@${NEW_MASTER_NODE_HOST} -i ~/.ssh/id_rsa_pgpool ${PGHOME}/bin/pg_ctl -D ${NEW_MASTER_NODE_PGDA

if [ $? -ne 0 ]; then
    logger -i -p local1.error failover.sh: new_master_host=$NEW_MASTER_NODE_HOST promote failed
    exit 1
fi

logger -i -p local1.info failover.sh: end: new_master_node_id=$NEW_MASTER_NODE_ID started as the primary node
exit 0

```

- /etc/pgpool-II/follow_master.sh

```

#!/bin/bash
# This script is run after failover_command to synchronize the Standby with the new Primary.
# First try pg_rewind. If pg_rewind failed, use pg_basebackup.

set -o xtrace
exec >>(logger -i -p local1.info) 2>&1

# Special values:
# %d = failed node id
# %h = failed node hostname
# %p = failed node port number
# %D = failed node database cluster path
# %m = new master node id

```

```

# %H = new master node hostname
# %M = old master node id
# %P = old primary node id
# %r = new master port number
# %R = new master database cluster path
# %N = old primary node hostname
# %S = old primary node port number
# %% = '%' character

FAILED_NODE_ID="$1"
FAILED_NODE_HOST="$2"
FAILED_NODE_PORT="$3"
FAILED_NODE_PGDATA="$4"
NEW_MASTER_NODE_ID="$5"
NEW_MASTER_NODE_HOST="$6"
OLD_MASTER_NODE_ID="$7"
OLD_PRIMARY_NODE_ID="$8"
NEW_MASTER_NODE_PORT="$9"
NEW_MASTER_NODE_PGDATA="${10}"

PGHOME=/usr/pgsql-11
ARCHIVEDIR=/var/lib/pgsql/archivedir
REPLUSER=repl
PCP_USER=pgpool
PGPOOL_PATH=/usr/bin
PCP_PORT=9898

logger -i -p local1.info follow_master.sh: start: Standby node ${FAILED_NODE_ID}

## Test passwordless SSH
ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@${NEW_MASTER_NODE_HOST} -i ~/.ssh/id_

if [ $? -ne 0 ]; then
    logger -i -p local1.info follow_master.sh: passwordless SSH to postgres@${NEW_MASTER_NODE_HOST} failed. Please setu
    exit 1
fi

## Get PostgreSQL major version
PGVERSION=`${PGHOME}/bin/initdb -V | awk '{print $3}' | sed 's/\..*/' | sed 's/([0-9]*)[a-zA-Z].*/1/'`

if [ $PGVERSION -ge 12 ]; then
    RECOVERYCONF=${FAILED_NODE_PGDATA}/myrecovery.conf
else
    RECOVERYCONF=${FAILED_NODE_PGDATA}/recovery.conf
fi

## Check the status of Standby
ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null \
postgres@${FAILED_NODE_HOST} -i ~/.ssh/id_rsa_pgpool ${PGHOME}/bin/pg_ctl -w -D ${FAILED_NODE_PGDATA} status

## If Standby is running, synchronize it with the new Primary.
if [ $? -eq 0 ]; then

    logger -i -p local1.info follow_master.sh: pg_rewind for $FAILED_NODE_ID

    # Create replication slot "${FAILED_NODE_HOST}"
    ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@${NEW_MASTER_NODE_HOST} -i ~/.ssh/
    ${PGHOME}/bin/psql -p ${NEW_MASTER_NODE_PORT} -c \"SELECT pg_create_physical_replication_slot('${FAILED_NOE
    \"

    ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@${FAILED_NODE_HOST} -i ~/.ssh/id_rsa_

    set -o errexit

    ${PGHOME}/bin/pg_ctl -w -m f -D ${FAILED_NODE_PGDATA} stop

    cat > ${RECOVERYCONF} << EOT
primary_conninfo = 'host=${NEW_MASTER_NODE_HOST} port=${NEW_MASTER_NODE_PORT} user=${REPLUSER} applicat
recovery_target_timeline = 'latest'
restore_command = 'scp ${NEW_MASTER_NODE_HOST}:${ARCHIVEDIR}/%f %p'
primary_slot_name = '${FAILED_NODE_HOST}'
EOT

    if [ ${PGVERSION} -ge 12 ]; then
        touch ${FAILED_NODE_PGDATA}/standby.signal
    else

```

```

else
    echo \"standby_mode = 'on'\" >> ${RECOVERYCONF}
fi

${PGHOME}/bin/pg_rewind -D ${FAILED_NODE_PGDATA} --source-server=\"user=postgres host=${NEW_MASTER_NOI
"

if [ $? -ne 0 ]; then
    logger -i -p local1.error follow_master.sh: end: pg_rewind failed. Try pg_basebackup.

    ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@${FAILED_NODE_HOST} -i ~/.ssh/id_rs

    set -o errexit

    # Execute pg_basebackup
    rm -rf ${FAILED_NODE_PGDATA}
    rm -rf ${ARCHIVEDIR}/*
    ${PGHOME}/bin/pg_basebackup -h ${NEW_MASTER_NODE_HOST} -U $REPLUSER -p ${NEW_MASTER_NODE_PORT}

    if [ ${PGVERSION} -ge 12 ]; then
        sed -i -e \"\\$ainclude_if_exists = '$(echo ${RECOVERYCONF} | sed -e 's/\\|\\|/g')'\" \\
            -e \"/^include_if_exists = '$(echo ${RECOVERYCONF} | sed -e 's/\\|\\|/g')'/d\" ${FAILED_NODE_PGDATA}/post
    fi

    cat > ${RECOVERYCONF} << EOT
primary_conninfo = 'host=${NEW_MASTER_NODE_HOST} port=${NEW_MASTER_NODE_PORT} user=${REPLUSER} applicat
recovery_target_timeline = 'latest'
restore_command = 'scp ${NEW_MASTER_NODE_HOST}:${ARCHIVEDIR}/%f %p'
primary_slot_name = '${FAILED_NODE_HOST}'
EOT

    if [ ${PGVERSION} -ge 12 ]; then
        touch ${FAILED_NODE_PGDATA}/standby.signal
    else
        echo \"standby_mode = 'on'\" >> ${RECOVERYCONF}
    fi
"

if [ $? -ne 0 ]; then
    # drop replication slot
    ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@${NEW_MASTER_NODE_HOST} -i ~,
        ${PGHOME}/bin/psql -p ${NEW_MASTER_NODE_PORT} -c \"SELECT pg_drop_replication_slot('${FAILED_NODE_NO
"

    logger -i -p local1.error follow_master.sh: end: pg_basebackup failed
    exit 1
fi
fi

# start Standby node on ${FAILED_NODE_HOST}
ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null \
    postgres@${FAILED_NODE_HOST} -i ~/.ssh/id_rsa_pgpool $PGHOME/bin/pg_ctl -l /dev/null -w -D ${FAILED_NODE_PG

# If start Standby successfully, attach this node
if [ $? -eq 0 ]; then

    # Run pcp_attach_node to attach Standby node to Pgpool-II.
    ${PGPOOL_PATH}/pcp_attach_node -w -h localhost -U $PCP_USER -p ${PCP_PORT} -n ${FAILED_NODE_ID}

    if [ $? -ne 0 ]; then
        logger -i -p local1.error follow_master.sh: end: pcp_attach_node failed
        exit 1
    fi

    # If start Standby failed, drop replication slot "${FAILED_NODE_HOST}"
    else

        ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@${NEW_MASTER_NODE_HOST} -i ~/.ss
        ${PGHOME}/bin/psql -p ${NEW_MASTER_NODE_PORT} -c "SELECT pg_drop_replication_slot('${FAILED_NODE_HOST}')"

        logger -i -p local1.error follow_master.sh: end: follow master command failed
        exit 1
    fi

else
    logger -i -p local1.info follow_master.sh: failed_nod_id=${FAILED_NODE_ID} is not running. skipping follow master comman
    exit 0

```



```

    exit 0
fi

logger -i -p local1.info follow_master.sh: end: follow master command complete
exit 0

```

8.3.5.3. Pgpool-II Online Recovery Configurations

Next, in order to perform online recovery with Pgpool-II we specify the PostgreSQL user name and online recovery command `recovery_1st_stage`. Because **Superuser** privilege in PostgreSQL is required for performing online recovery, we specify `postgres` user in `recovery_user`. Then, we create `recovery_1st_stage` and `pgpool_remote_start` in database cluster directory of PostgreSQL primary server (server1), and add execute permission.

```

recovery_user = 'postgres'
# Online recovery user
recovery_password = ''
# Online recovery password

recovery_1st_stage_command = 'recovery_1st_stage'

```

```

[server1]# su - postgres
[server1]$ vi /var/lib/pgsql/11/data/recovery_1st_stage
[server1]$ vi /var/lib/pgsql/11/data/pgpool_remote_start
[server1]$ chmod +x /var/lib/pgsql/11/data/{recovery_1st_stage,pgpool_remote_start}

```

- `/var/lib/pgsql/11/data/recovery_1st_stage`

```

#!/bin/bash
# This script is executed by "recovery_1st_stage" to recovery a Standby node.

set -o xtrace
exec >>(logger -i -p local1.info) 2>&1

PRIMARY_NODE_PGDATA="$1"
DEST_NODE_HOST="$2"
DEST_NODE_PGDATA="$3"
PRIMARY_NODE_PORT="$4"
DEST_NODE_ID="$5"
DEST_NODE_PORT="$6"

PRIMARY_NODE_HOST=$(hostname)
PGHOME=/usr/pgsql-11
ARCHIVEDIR=/var/lib/pgsql/archivedir
REPLUSER=repl

logger -i -p local1.info recovery_1st_stage: start: pg_basebackup for Standby node $DEST_NODE_ID

## Test passwordless SSH
ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@${DEST_NODE_HOST} -i ~/.ssh/id_rsa_pgpool

if [ $? -ne 0 ]; then
    logger -i -p local1.info recovery_1st_stage: passwordless SSH to postgres@${DEST_NODE_HOST} failed. Please setup passwordless SSH.
    exit 1
fi

## Get PostgreSQL major version
PGVERSION=` ${PGHOME}/bin/initdb -V | awk '{print $3}' | sed 's/\./*/' | sed 's/\([0-9]*\)[a-zA-Z].*/\1/'
if [ $PGVERSION -ge 12 ]; then
    RECOVERYCONF=${DEST_NODE_PGDATA}/myrecovery.conf
else
    RECOVERYCONF=${DEST_NODE_PGDATA}/recovery.conf
fi

## Create replication slot "${DEST_NODE_HOST}"
${PGHOME}/bin/psql -n ${PRIMARY_NODE_PORT} << FOO

```

```

SELECT pg_create_physical_replication_slot('${DEST_NODE_HOST}');
EOQ

## Execute pg_basebackup to recovery Standby node
ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@${DEST_NODE_HOST} -i ~/.ssh/id_rsa_pgpool

set -o errexit

rm -rf ${DEST_NODE_PGDATA}
rm -rf ${ARCHIVEDIR}/*

${PGHOME}/bin/pg_basebackup -h ${PRIMARY_NODE_HOST} -U ${REPLUSER} -p ${PRIMARY_NODE_PORT} -D ${DEST_NODE_PGDATA}

if [ ${PGVERSION} -ge 12 ]; then
    sed -i -e "\\\$ainclude_if_exists = '${(echo ${RECOVERYCONF} | sed -e 's/\\/\\/g')}'" \
        -e "\/^include_if_exists = '${(echo ${RECOVERYCONF} | sed -e 's/\\/\\/g')}'/d\" ${DEST_NODE_PGDATA}/postgresql.conf
fi

cat > ${RECOVERYCONF} << EOT
primary_conninfo = 'host=${PRIMARY_NODE_HOST} port=${PRIMARY_NODE_PORT} user=${REPLUSER} application_name=
recovery_target_timeline = 'latest'
restore_command = 'scp ${PRIMARY_NODE_HOST}:${ARCHIVEDIR}/%f %p'
primary_slot_name = '${DEST_NODE_HOST}'
EOT

if [ ${PGVERSION} -ge 12 ]; then
    touch ${DEST_NODE_PGDATA}/standby.signal
else
    echo "\"standby_mode = 'on'\" >> ${RECOVERYCONF}
fi

sed -i \"s/#*port = .*port = ${DEST_NODE_PORT}/^\" ${DEST_NODE_PGDATA}/postgresql.conf
"

if [ $? -ne 0 ]; then

    ${PGHOME}/bin/psql -p ${PRIMARY_NODE_PORT} << EOQ
SELECT pg_drop_replication_slot('${DEST_NODE_HOST}');
EOQ

    logger -i -p local1.error recovery_1st_stage: end: pg_basebackup failed. online recovery failed
    exit 1
fi

logger -i -p local1.info recovery_1st_stage: end: recovery_1st_stage complete
exit 0

```

- /var/lib/pgsql/11/data/pgpool_remote_start

```
#!/bin/bash
# This script is run after recovery_1st_stage to start Standby node.

set -o xtrace
exec > >(logger -i -p local1.info) 2>&1

PGHOME=/usr/pgsql-11
DEST_NODE_HOST="$1"
DEST_NODE_PGDATA="$2"

logger -i -p local1.info pgpool_remote_start: start: remote start Standby node $DEST_NODE_HOST

## Test passwordless SSH
ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@${DEST_NODE_HOST} -i ~/.ssh/id_rsa_pgpool

if [ $? -ne 0 ]; then
    logger -i -p local1.info pgpool_remote_start: passwordless SSH to postgres@${DEST_NODE_HOST} failed. Please setup passwordless SSH.
    exit 1
fi

## Start Standby node
ssh -T -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null postgres@$DEST_NODE_HOST -i ~/.ssh/id_rsa_pgpool \
    $PGHOME/bin/pg_ctl -l /dev/null -w -D $DEST_NODE_PGDATA start
"

if [ $? -ne 0 ]; then
    logger -i -p local1.error pgpool_remote_start: $DEST_NODE_HOST PostgreSQL start failed.
    exit 1
fi

logger -i -p local1.info pgpool_remote_start: end: $DEST_NODE_HOST PostgreSQL started successfully.
exit 0
```

In order to use the online recovery functionality, the functions of `pgpool_recovery`, `pgpool_remote_start`, `pgpool_switch_xlog` are required, so we need install `pgpool_recovery` on `template1` of PostgreSQL server `server1`.

```
[server1]# su - postgres
[server1]$ psql template1 -c "CREATE EXTENSION pgpool_recovery"
```

8.3.5.4. Client Authentication Configuration

Because in the section [Before Starting](#), we already set PostgreSQL authentication method to `scram-sha-256`, it is necessary to set a client authentication by Pgpool-II to connect to backend nodes. When installing with RPM, the Pgpool-II configuration file `pool_hba.conf` is in `/etc/pgpool-II`. By default, `pool_hba` authentication is disabled, set `enable_pool_hba = on` to enable it.

```
enable_pool_hba = on
```

The format of `pool_hba.conf` file follows very closely PostgreSQL's `pg_hba.conf` format. Set `pgpool` and `postgres` user's authentication method to `scram-sha-256`.

```
host all pgpool 0.0.0.0/0 scram-sha-256
host all postgres 0.0.0.0/0 scram-sha-256
```

Note: Please note that in Pgpool-II 4.0 only AES encrypted password or clear text password can be specified in [health_check_password](#), [sr_check_password](#), [wd_lifeccheck_password](#), [recovery_password](#) in `pgpool.conf`.

The default password file name for authentication is [pool_passwd](#). To use `scram-sha-256` authentication, the decryption key to decrypt the passwords is required. We create the `.pgpoolkey` file in Pgpool-II start user `postgres`'s (Pgpool-II 4.1 or later) home directory. (Pgpool-II 4.0 or before, by default Pgpool-II is started as `root`)

```
[all servers]# su - postgres
[all servers]$ echo 'some string' > ~/.pgpoolkey
[all servers]$ chmod 600 ~/.pgpoolkey
```

Execute command `pg_enc -m -k /path/to/.pgpoolkey -u username -p` to register user name and AES encrypted password in file `pool_passwd`. If `pool_passwd` doesn't exist yet, it will be created in the same directory as `pgpool.conf`.

```
[all servers]# su - postgres
[all servers]$ pg_enc -m -k ~/.pgpoolkey -u pgpool -p
db password: [pgpool user's password]
[all servers]$ pg_enc -m -k ~/.pgpoolkey -u postgres -p
db password: [postgres user's password]

# cat /etc/pgpool-II/pool_passwd
pgpool:AESheq2ZMZjynddMWk5sKP/Rw==
postgres:AESHs/pWL5rtXy2lwuzroHfqg==
```

8.3.5.5. Watchdog Configuration

Enable watchdog functionality on `server1`, `server2`, `server3`.

```
use_watchdog = on
```

Specify virtual IP address that accepts connections from clients on `server1`, `server2`, `server3`. Ensure that the IP address set to virtual IP isn't used yet.

```
delegate_IP = '192.168.137.150'
```

To bring up/down the virtual IP and send the ARP requests, we set [if_up_cmd](#), [if_down_cmd](#) and [arping_cmd](#). The network interface used in this example is "enp0s8". Since root privilege is required to execute `if_up/down_cmd` or `arping_cmd` command, use `setuid` on these command or allow Pgpool-II startup user, `postgres` user (Pgpool-II 4.1 or later) to run `sudo` command without a password. If installed from RPM, the `postgres` user has been configured to run `ip/arping` via `sudo` without a password.

```
if_up_cmd = '/usr/bin/sudo /sbin/ip addr add $_IP_$/24 dev enp0s8 label enp0s8:0'
if_down_cmd = '/usr/bin/sudo /sbin/ip addr del $_IP_$/24 dev enp0s8'
arping_cmd = '/usr/bin/sudo /usr/sbin/arping -U $_IP_ $ -w 1 -I enp0s8'
```

Note: If "Defaults requiretty" is set in the `/etc/sudoers`, please ensure that the `pgpool` startup user can execute the `if_up_cmd`, `if_down_cmd` and `arping_cmd` command without a `tty`.

Set [if_cmd_path](#) and [arping_path](#) according to the command path. If `if_up/down_cmd` or `arping_cmd` starts with `/`, these parameters will be ignored.

```
if_cmd_path = '/sbin'
arping_path = '/usr/sbin'
```

Specify the hostname and port number of each Pgpool-II server.

- server1

```
wd_hostname = 'server1'
wd_port = 9000
```

- server2

```
wd_hostname = 'server2'
wd_port = 9000
```

- server3

```
wd_hostname = 'server3'
wd_port = 9000
```

Specify the hostname, Pgpool-II port number, and watchdog port number of monitored Pgpool-II servers on each Pgpool-II server.

- server1

```
# - Other pgpool Connection Settings -

other_pgpool_hostname0 = 'server2'
# Host name or IP address to connect to for other pgpool 0
# (change requires restart)
other_pgpool_port0 = 9999
# Port number for other pgpool 0
# (change requires restart)
other_wd_port0 = 9000
# Port number for other watchdog 0
# (change requires restart)
other_pgpool_hostname1 = 'server3'
other_pgpool_port1 = 9999
other_wd_port1 = 9000
```

- server2

```
# - Other pgpool Connection Settings -

other_pgpool_hostname0 = 'server1'
# Host name or IP address to connect to for other pgpool 0
# (change requires restart)
other_pgpool_port0 = 9999
# Port number for other pgpool 0
# (change requires restart)
other_wd_port0 = 9000
# Port number for other watchdog 0
# (change requires restart)
other_pgpool_hostname1 = 'server3'
other_pgpool_port1 = 9999
other_wd_port1 = 9000
```

- server3

```
# - Other pgpool Connection Settings -

other_pgpool_hostname0 = 'server1'
# Host name or IP address to connect to for other pgpool 0
# (change requires restart)
other_pgpool_port0 = 9999
# Port number for other pgpool 0
# (change requires restart)
other_wd_port0 = 9000
# Port number for other watchdog 0
# (change requires restart)
other_pgpool_hostname1 = 'server2'
other_pgpool_port1 = 9999
other_wd_port1 = 9000
```

Specify the hostname and port number of destination for sending heartbeat signal on `server1`, `server2`, `server3`.

- `server1`

```
heartbeat_destination0 = 'server2'
# Host name or IP address of destination 0
# for sending heartbeat signal.
# (change requires restart)
heartbeat_destination_port0 = 9694
# Port number of destination 0 for sending
# heartbeat signal. Usually this is the
# same as wd_heartbeat_port.
# (change requires restart)
heartbeat_device0 = ""
# Name of NIC device (such like 'eth0')
# used for sending/receiving heartbeat
# signal to/from destination 0.
# This works only when this is not empty
# and pgpool has root privilege.
# (change requires restart)

heartbeat_destination1 = 'server3'
heartbeat_destination_port1 = 9694
heartbeat_device1 = ""
```

- `server2`

```
heartbeat_destination0 = 'server1'
# Host name or IP address of destination 0
# for sending heartbeat signal.
# (change requires restart)
heartbeat_destination_port0 = 9694
# Port number of destination 0 for sending
# heartbeat signal. Usually this is the
# same as wd_heartbeat_port.
# (change requires restart)
heartbeat_device0 = ""
# Name of NIC device (such like 'eth0')
# used for sending/receiving heartbeat
# signal to/from destination 0.
# This works only when this is not empty
# and pgpool has root privilege.
# (change requires restart)

heartbeat_destination1 = 'server3'
heartbeat_destination_port1 = 9694
heartbeat_device1 = ""
```

- `server3`

```
heartbeat_destination0 = 'server1'
# Host name or IP address of destination 0
# for sending heartbeat signal.
# (change requires restart)
heartbeat_destination_port0 = 9694
# Port number of destination 0 for sending
# heartbeat signal. Usually this is the
# same as wd_heartbeat_port.
# (change requires restart)
heartbeat_device0 = ""
# Name of NIC device (such like 'eth0')
# used for sending/receiving heartbeat
# signal to/from destination 0.
# This works only when this is not empty
# and pppool has root privilege.
# (change requires restart)

heartbeat_destination1 = 'server2'
heartbeat_destination_port1 = 9694
heartbeat_device1 = ""
```

8.3.5.6. /etc/sysconfig/pgpool Configuration

If you want to ignore the `pgpool_status` file at startup of Pgpool-II, add "- D" to the start option `OPTS` to `/etc/sysconfig/pgpool`.

```
[all servers]# vi /etc/sysconfig/pgpool
...
OPTS="-D -n"
```

8.3.5.7. Logging

In the example, we output Pgpool-II's log to `syslog`.

```
log_destination = 'syslog'
# Where to log
# Valid values are combinations of stderr,
# and syslog. Default to stderr.

syslog_facility = 'LOCAL1'
# Syslog local facility. Default to LOCAL0
```

Create Pgpool-II log file.

```
[all servers]# mkdir /var/log/pgpool-II
[all servers]# touch /var/log/pgpool-II/pgpool.log
```

Edit config file of `syslog` `/etc/rsyslog.conf`.

```
[all servers]# vi /etc/rsyslog.conf
...
*.info;mail.none;authpriv.none;cron.none;LOCAL1.none /var/log/messages
LOCAL1.* /var/log/pgpool-II/pgpool.log
```

Setting `logrotate` same as `/var/log/messages`.

```
[all servers]# vi /etc/logrotate.d/syslog
...
/var/log/messages
/var/log/pgpool-II/pgpool.log
/var/log/secure
```

Restart rsyslog service.

```
[all servers]# systemctl restart rsyslog
```

8.3.5.8. PCP Command Configuration

Since user authentication is required to use the PCP command, specify user name and md5 encrypted password in `pcp.conf`. Here we create the encrypted password for `pgpool` user, and add "username:encrypted password" in `/etc/pgpool-II/pcp.conf`.

```
[all servers]# echo 'pgpool:`pg_md5 PCP passowrd` >> /etc/pgpool-II/pcp.conf
```

8.3.5.9. .pcppass

Since `follow_master_command` script has to execute PCP command without entering the password, we create `.pcppass` in the home directory of `Pgpool-II` startup user (root user).

```
[all servers]# echo 'localhost:9898:pgpool:pgpool' > ~/.pcppass
[all servers]# chmod 600 ~/.pcppass
```

The settings of `Pgpool-II` is completed.

8.3.6. Starting/Stopping Pgpool-II

Next we start `Pgpool-II`. Before starting `Pgpool-II`, please start PostgreSQL servers first. Also, when stopping PostgreSQL, it is necessary to stop `Pgpool-II` first.

- Starting `Pgpool-II`

In section [Before Starting](#), we already set the auto-start of `Pgpool-II`. To start `Pgpool-II`, restart the whole system or execute the following command.

```
# systemctl start pgpool.service
```

- Stopping `Pgpool-II`

```
# systemctl stop pgpool.service
```

8.3.7. How to use

Let's start to use `Pgpool-II`. First, let's start `Pgpool-II` on `server1`, `server2`, `server3` by using the following command.


```
# systemctl start pgpool.service
```

8.3.7.1. Set up PostgreSQL standby server

First, we should set up PostgreSQL standby server by using Pgpool-II online recovery functionality. Ensure that `recovery_1st_stage` and `pgpool_remote_start` scripts used by `pcp_recovery_node` command are in database cluster directory of PostgreSQL primary server (`server1`).

```
# pcp_recovery_node -h 192.168.137.150 -p 9898 -U pgpool -n 1
Password:
pcp_recovery_node -- Command Successful

# pcp_recovery_node -h 192.168.137.150 -p 9898 -U pgpool -n 2
Password:
pcp_recovery_node -- Command Successful
```

After executing `pcp_recovery_node` command, verify that `server2` and `server3` are started as PostgreSQL standby server.

```
# psql -h 192.168.137.150 -p 9999 -U pgpool postgres -c "show pool_nodes"
Password for user pgpool
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay | replication_state | rep
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
 0      | server1  | 5432 | up     | 0.333333 | primary | 0          | false             | 0                | streaming         | 2019-08-0
 1      | server2  | 5432 | up     | 0.333333 | standby | 0          | true              | 0                | streaming         | async      | 20
 2      | server3  | 5432 | up     | 0.333333 | standby | 0          | false             | 0                | streaming         | async      | 20
(3 rows)
```

8.3.7.2. Switching active/standby watchdog

Confirm the watchdog status by using `pcp_watchdog_info`. The Pgpool-II server which is started first run as MASTER.

```
# pcp_watchdog_info -h 192.168.137.150 -p 9898 -U pgpool
Password:
3 YES server1:9999 Linux server1 server1

server1:9999 Linux server1 server1 9999 9000 4 MASTER #The Pgpool-II server started first becomes "MASTER".
server2:9999 Linux server2 server2 9999 9000 7 STANDBY #run as standby
server3:9999 Linux server3 server3 9999 9000 7 STANDBY #run as standby
```

Stop active server `server1`, then `server2` or `server3` will be promoted to active server. To stop `server1`, we can stop Pgpool-II service or shutdown the whole system. Here, we stop Pgpool-II service.

```
[server1]# systemctl stop pgpool.service

# pcp_watchdog_info -p 9898 -h 192.168.137.150 -U pgpool
Password:
3 YES server2:9999 Linux server2 server2

server2:9999 Linux server2 server2 9999 9000 4 MASTER #server2 is promoted to MASTER
server1:9999 Linux server1 server1 9999 9000 10 SHUTDOWN #server1 is stopped
server3:9999 Linux server3 server3 9999 9000 7 STANDBY #server3 runs as STANDBY
```

Start Pgpool-II (`server1`) which we have stopped again, and verify that `server1` runs as a standby.

```
[server1]# systemctl start pgpool.service

[server1]# pcp_watchdog_info -p 9898 -h 192.168.137.150 -U pgpool
Password:
3 YES server2:9999 Linux server2 server2

server2:9999 Linux server2 server2 9999 9000 4 MASTER
server1:9999 Linux server1 server1 9999 9000 7 STANDBY
server3:9999 Linux server3 server3 9999 9000 7 STANDBY
```

8.3.7.3. Failover

First, use `psql` to connect to PostgreSQL via virtual IP, and verify the backend information.

```
# psql -h 192.168.137.150 -p 9999 -U pgpool postgres -c "show pool_nodes"
Password for user pgpool:
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay | replication_state | rep
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | server1 | 5432 | up | 0.333333 | primary | 0 | false | 0 | | | 2019-08-C
1 | server2 | 5432 | up | 0.333333 | standby | 0 | true | 0 | streaming | async | 20
2 | server3 | 5432 | up | 0.333333 | standby | 0 | false | 0 | streaming | async | 20
(3 rows)
```

Next, stop primary PostgreSQL server `server1`, and verify automatic failover.

```
[server1]$ pg_ctl -D /var/lib/pgsql/11/data -m immediate stop
```

After stopping PostgreSQL on `server1`, failover occurs and PostgreSQL on `server2` becomes new primary DB.

```
# psql -h 192.168.137.150 -p 9999 -U pgpool postgres -c "show pool_nodes"
Password for user pgpool:
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay | replication_state | rep
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | server1 | 5432 | down | 0.333333 | standby | 0 | false | 0 | | | 2019-08-C
1 | server2 | 5432 | up | 0.333333 | primary | 0 | true | 0 | | | 2019-08-C
2 | server3 | 5432 | up | 0.333333 | standby | 0 | false | 0 | streaming | async | 20
(3 rows)
```

`server3` is running as standby of new primary `server2`.

```
[server3]# psql -h server3 -p 5432 -U pgpool postgres -c "select pg_is_in_recovery()"
pg_is_in_recovery
-----
t

[server2]# psql -h server2 -p 5432 -U pgpool postgres -c "select pg_is_in_recovery()"
pg_is_in_recovery
-----
f

[server2]# psql -h server2 -p 5432 -U pgpool postgres -c "select * from pg_stat_replication" -x
-[ RECORD 1 ]-----+
pid          | 11059
usesysid     | 16392
username     | repl
application_name | server3
client_addr  | 192.168.137.103
client_hostname |
client_port  | 48694
backend_start | 2019-08-06 11:36:07.479161+09
backend_xmin  |
state        | streaming
sent_lsn     | 0/75000148
write_lsn    | 0/75000148
flush_lsn    | 0/75000148
replay_lsn   | 0/75000148
write_lag    |
flush_lag    |
replay_lag   |
sync_priority | 0
sync_state   | async
reply_time   | 2019-08-06 11:42:59.823961+09
```

8.3.7.4. Online Recovery

Here, we use Pgpool-II online recovery functionality to restore `server1` (old primary server) as a standby. Before restoring the old primary server, please ensure that `recovery_1st_stage` and `pgpool_remote_start` scripts exist in database cluster directory of current primary server `server2`.

```
# pcp_recovery_node -h 192.168.137.150 -p 9898 -U pgpool -n 0
Password:
pcp_recovery_node -- Command Successful
```

Then verify that `server1` is started as a standby.

```
# psql -h 192.168.137.150 -p 9999 -U pgpool postgres -c "show pool_nodes"
Password for user pgpool:
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay | replication_state | rep
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
0       | server1  | 5432 | up     | 0.333333 | standby | 0           | false             | 0                 | streaming         | async | 20
1       | server2  | 5432 | up     | 0.333333 | primary | 0           | false             | 0                 | streaming         | async | 2019-08-0
2       | server3  | 5432 | up     | 0.333333 | standby | 0           | true              | 0                 | streaming         | async | 20
(3 rows)
```

8.4. AWS Configuration Example

This tutorial explains the simple way to try "Watchdog" on [AWS](#) and using the [Elastic IP Address](#) as the Virtual IP for the high availability solution.

Note: You can use watchdog with Pgpool-II in any mode: replication mode, master/slave

mode and raw mode.

8.4.1. AWS Setup

For this example, we will use two node Pgpool-II watchdog cluster. So we will set up two Linux Amazon EC2 instances and one Elastic IP address. So for this example, do the following steps:

- Launch two Linux Amazon EC2 instances. For this example, we name these instances as "instance-1" and "instance-2"
- Configure the security group for the instances and allow inbound traffic on ports used by pgpool-II and watchdog.
- Install the Pgpool-II on both instances.
- Allocate an Elastic IP address. For this example, we will use "35.163.178.3" as an Elastic IP address"

8.4.2. Pgpool-II configurations

Mostly the Pgpool-II configurations for this example will be same as in the [Section 8.2](#), except the [delegate_IP](#) which we will not set in this example instead we will use [wd_escalation_command](#) and [wd_de_escalation_command](#) to switch the Elastic IP address to the master/Active Pgpool-II node.

8.4.2.1. Pgpool-II configurations on Instance-1

```
use_watchdog = on
delegate_IP = ""
wd_hostname = 'instance-1-private-ip'
other_pgpool_hostname0 = 'instance-2-private-ip'
other_pgpool_port0 = 9999
other_wd_port0 = 9000
wd_escalation_command = '$path_to_script/aws-escalation.sh'
wd_de_escalation_command = '$path_to_script/aws-de-escalation.sh'
```

8.4.2.2. Pgpool-II configurations on Instance-2

```
use_watchdog = on
delegate_IP = ""
wd_hostname = 'instance-2-private-ip'
other_pgpool_hostname0 = 'instance-1-private-ip'
other_pgpool_port0 = 9999
other_wd_port0 = 9000
wd_escalation_command = '$path_to_script/aws-escalation.sh'
wd_de_escalation_command = '$path_to_script/aws-de-escalation.sh'
```

8.4.3. escalation and de-escalation Scripts

Create the [aws-escalation.sh](#) and [aws-de-escalation.sh](#) scripts on both instances and point the [wd_escalation_command](#) and [wd_de_escalation_command](#) to the respective scripts.

Note: You may need to configure the AWS CLI first on all AWS instances to enable the execution of commands used by [wd-escalation.sh](#) and [wd-de-escalation.sh](#). See

8.4.3.1. escalation script

This script will be executed by the watchdog to assign the Elastic IP on the instance when the watchdog becomes the active/master node. Change the INSTANCE_ID and ELASTIC_IP values as per your AWS setup values.

aws-escalation.sh:

```
#!/bin/sh

ELASTIC_IP=35.163.178.3
# replace it with the Elastic IP address you
# allocated from the aws console
INSTANCE_ID=i-0a9b64e449b17ed4b
# replace it with the instance id of the Instance
# this script is installed on

echo "Assigning Elastic IP $ELASTIC_IP to the instance $INSTANCE_ID"
# bring up the Elastic IP
aws ec2 associate-address --instance-id $INSTANCE_ID --public-ip $ELASTIC_IP

exit 0
```

8.4.3.2. de-escalation script

This script will be executed by watchdog to remove the Elastic IP from the instance when the watchdog resign from the active/master node.

aws-de-escalation.sh:

```
#!/bin/sh

ELASTIC_IP=35.163.178.3
# replace it with the Elastic IP address you
# allocated from the aws console

echo "disassociating the Elastic IP $ELASTIC_IP from the instance"
# bring down the Elastic IP
aws ec2 disassociate-address --public-ip $ELASTIC_IP

exit 0
```

AWS Command References

"[Configure AWS CLI](#)", **AWS Documentation: Configuring the AWS Command Line Interface** .

"[associate-address](#)", **AWS Documentation: associate-address reference** .

"[disassociate-address](#)", **AWS Documentation: disassociate-address reference** .

8.4.4. Try it out

Start Pgpool-II on each server with "-n" switch and redirect log messages to the pgpool.log file. The log message of master/active Pgpool-II node will show the message of Elastic IP assignment.

```
LOG: I am the cluster leader node. Starting escalation process
LOG: escalation process started with PID:23543
LOG: watchdog: escalation started
Assigning Elastic IP 35.163.178.3 to the instance i-0a9b64e449b17ed4b
{
  "AssociationId": "eipassoc-39853c42"
}

LOG: watchdog escalation successful
LOG: watchdog escalation process with pid: 23543 exit with SUCCESS.
```

Confirm to ping to the Elastic IP address.

```
[user@someserver]$ ping 35.163.178.3
PING 35.163.178.3 (35.163.178.3) 56(84) bytes of data.
64 bytes from 35.163.178.3: icmp_seq=1 ttl=64 time=0.328 ms
64 bytes from 35.163.178.3: icmp_seq=2 ttl=64 time=0.264 ms
64 bytes from 35.163.178.3: icmp_seq=3 ttl=64 time=0.412 ms
```

Try to connect PostgreSQL by "psql -h ELASTIC_IP -p port".

```
[user@someserver]$ psql -h 35.163.178.3 -p 9999 -l
```

8.4.5. Switching Elastic IP

To confirm if the Standby server acquires the Elastic IP when the Active server becomes unavailable, Stop the Pgpool-II on the Active server. Then, the Standby server should start using the Elastic IP address, And the Pgpool-II log will show the below messages.

```
LOG: remote node "172.31.2.94:9999 [Linux ip-172-31-2-94]" is shutting down
LOG: watchdog cluster has lost the coordinator node
```

```
LOG: watchdog node state changed from [STANDBY] to [JOINING]
LOG: watchdog node state changed from [JOINING] to [INITIALIZING]
LOG: I am the only alive node in the watchdog cluster
HINT: skipping stand for coordinator state
LOG: watchdog node state changed from [INITIALIZING] to [MASTER]
LOG: I am announcing my self as master/coordinator watchdog node
LOG: I am the cluster leader node
DETAIL: our declare coordinator message is accepted by all nodes
LOG: I am the cluster leader node. Starting escalation process
LOG: escalation process started with PID:23543
LOG: watchdog: escalation started
Assigning Elastic IP 35.163.178.3 to the instance i-0dd3e60734a6ebe14
{
  "AssociationId": "eipassoc-39853c42"
}

LOG: watchdog escalation successful
LOG: watchdog escalation process with pid: 61581 exit with SUCCESS.
```

Confirm to ping to the Elastic IP address again.

```
[user@someserver]$ ping 35.163.178.3
PING 35.163.178.3 (35.163.178.3) 56(84) bytes of data.
64 bytes from 35.163.178.3: icmp_seq=1 ttl=64 time=0.328 ms
64 bytes from 35.163.178.3: icmp_seq=2 ttl=64 time=0.264 ms
64 bytes from 35.163.178.3: icmp_seq=3 ttl=64 time=0.412 ms
```

Try to connect PostgreSQL by "psql -h ELASTIC_IP -p port".

```
[user@someserver]$ psql -h 35.163.178.3 -p 9999 -l
```

8.5. Aurora Configuration Example

Amazon Aurora for PostgreSQL Compatibility (Aurora) is a managed service for PostgreSQL. From user's point of view, Aurora can be regarded as a streaming replication cluster with some exceptions. First, fail over and online recovery are managed by Aurora. So you don't need to set [failover_command](#), [follow_master_command](#), and recovery related parameters. In this section we explain how to set up Pgpool-II for Aurora.

8.5.1. Setting pgpool.conf for Aurora

- Create `pgpool.conf` from `pgpool.conf.sample-stream`.
- Set [sr_check_period](#) to 0 to disable streaming replication delay checking. This is because Aurora does not provide necessary functions to check the replication delay.
- Enable [enable_pool_hba](#) to on so that md5 authentication is enabled (Aurora always use md5 authentication).
- Create `pool_password`. See [Section 6.2.2](#) for more details.
- Set [backend_hostname0](#) for the Aurora writer node. Set other [backend_hostname](#) for the Aurora reader node. Set appropriate [backend_weight](#) as usual. You don't need to set [backend_data_directory](#)
- Set `ALWAYS_MASTER` flag to the [backend_flag](#) for [backend_hostname0](#).
- Enable health checking. Set [health_check_period](#) to 5. Set [health_check_user](#), [health_check_password](#), [health_check_user](#) and [health_check_database](#) to appropriate values. Enable health check retry. Aurora shutdowns all DB nodes while switching over or failover. If the retry is not performed, Pgpool-II thinks that all DB nodes are in down status so that it is required to restart Pgpool-II. Set [health_check_max_retries](#) to 20. Set [health_check_retry_delay](#) to 1 to avoid the problem.
- Disable [failover_on_backend_error](#) to avoid failover when connecting to the backend or detecting errors on backend side while executing queries for the same reasons above.

IV. Reference

This part contains reference information for the Pgool-II.

The reference entries are also available as traditional "man" pages.

Table of Contents

I. [Server commands](#)

[pgpool](#) -- Pgpool-II main server

II. [PCP commands](#)

[pcp_common_options](#) -- common options used in PCP commands

[pcp_node_count](#) -- displays the total number of database nodes

[pcp_node_info](#) -- displays the information on the given node ID

[pcp_watchdog_info](#) -- displays the watchdog status of the Pgpool-II

[pcp_proc_count](#) -- displays the list of Pgpool-II children process IDs

[pcp_proc_info](#) -- displays the information on the given Pgpool-II child process ID

[pcp_pool_status](#) -- displays the parameter values as defined in `pgpool.conf`

[pcp_detach_node](#) -- detaches the given node from Pgpool-II. Existing connections to Pgpool-II are forced to be disconnected.

[pcp_attach_node](#) -- attaches the given node to Pgpool-II.

[pcp_promote_node](#) -- promotes the given node as new master to Pgpool-II
[pcp_stop_pgpool](#) -- terminates the Pgpool-II process
[pcp_recovery_node](#) -- attaches the given backend node with recovery

III. [Other commands](#)

[pg_md5](#) -- produces encrypted password in md5
[pg_enc](#) -- AES256 password encryption utility
[pgproto](#) -- tests PostgreSQL or any other servers that understand the frontend/backend protocol.
[pgpool_setup](#) -- Create a temporary installation of Pgpool-II cluster
[watchdog_setup](#) -- Create a temporary installation of Pgpool-II clusters with watchdog

IV. [SQL type commands](#)

[PGPOOL SHOW](#) -- show the value of a configuration parameter
[PGPOOL SET](#) -- change a configuration parameter
[PGPOOL RESET](#) -- restore the value of a configuration parameter to the default value
[SHOW POOL STATUS](#) -- sends back the list of configuration parameters with their name, value, and description
[SHOW POOL NODES](#) -- sends back a list of all configured nodes
[SHOW POOL PROCESSES](#) -- sends back a list of all Pgpool-II processes waiting for connections and dealing with a connection
[SHOW POOL POOLS](#) -- sends back a list of pools handled by Pgpool-II.
[SHOW POOL VERSION](#) -- displays a string containing the Pgpool-II release number.
[SHOW POOL CACHE](#) -- displays cache storage statistics

V. [pgpool_adm extension](#)

[pgpool_adm_pcp_node_info](#) -- a function to display the information on the given node ID
[pgpool_adm_pcp_pool_status](#) -- a function to retrieves parameters in pgpool.conf.
[pgpool_adm_pcp_node_count](#) -- a function to retrieves number of backend nodes.
[pgpool_adm_pcp_attach_node](#) -- a function to attach given node ID
[pgpool_adm_pcp_detach_node](#) -- a function to detach given node ID

I. Server commands

This part contains reference information for server commands. Currently only `pgpool` falls into this category.

Table of Contents

[pgpool](#) -- Pgpool-II main server

pgpool

Name

`pgpool` -- Pgpool-II main server

Synopsis

`pgpool` [**option**...]

`pgpool` [**option**...] **stop**

`pgpool` [**option**...] **reload**

Description

the Pgpool-II main server

Usages

pgpool runs in 3 modes: start, stop and reload. If none of stop or reload is given, it is assumed that "start" is specified.

Common options

These are common options for 3 modes.

`-a hba_config_file`
`--hba-file=hba_config_file`

Set the path to the `pool_hba.conf` configuration file. Mandatory if the file is placed other than the standard location.

`-f config_file`
`--config-file=config_file`

Set the path to the `pgpool.conf` configuration file. Mandatory if the file is placed other than the standard location.

`-F pc_config_file`
`--pcp-file=pcp_config_file`

Set the path to the `pcp.conf` configuration file. Mandatory if the file is placed other than the standard location.

`-k key_file`
`--key-file=key_file`

Set the path to the `.pgpoolkey` file. Mandatory if you use AES256 encrypted password and the file is placed other than the standard location and used.

`-h`
`--help`

Print help.

Starting Pgpool-II main server

Here are options for the start mode.

`-d`
`--debug`

Run Pgpool-II in debug mode. Lots of debug messages are produced.

`-n`
`--dont-detach`

Don't run in daemon mode, does not detach control ttys.

`-x`
`--debug-assertions`

Turns on various assertion checks, This is a debugging aid.

`-C`
`--clear-oidmaps`

Clear query cache oidmaps when [memqcache_method](#) is `memcached`.

If `memqcache_method` is `shmem`, Pgpool-II always discards oidmaps at the start-up time. So this option is not necessary.

`-D`
`--discard-status`

Discard `pgpool_status` file and do not restore previous status.

Stopping Pgpool-II main server

Here are options for the stop mode.

```
-m shutdown_mode  
--mode=shutdown_mode
```

Stop Pgpool-II. `shutdown_mode` is either `smart`, `fast` or `immediate`. If `smart` is specified, Pgpool-II will wait for all clients are disconnected. If `fast` or `immediate` are specified, Pgpool-II immediately stops itself without waiting for all clients are disconnected. There's no difference between `fast` and `immediate` in the current implementation.

Reloading Pgpool-II configuration files

Reload configuration file of Pgpool-II. No specific options exist for reload mode. Common options are applicable.

II. PCP commands

This part contains reference information for PCP commands. PCP commands are UNIX commands which manipulate pgpool-II via the network. Please note that the parameter format for all PCP commands has been changed since Pgpool-II 3.5.

1. PCP connection authentication

PCP user names and passwords must be declared in `pcp.conf` in `$prefix/etc` directory (see [Section 3.2](#) to know to create the file). `-F` option can be used when starting Pgpool-II if `pcp.conf` is placed somewhere else.

2. PCP password file

The file `.pcppass` in a user's home directory or the file referenced by environment variable `PCPPASSFILE` can contain passwords to be used if no password has been specified for the `pcp` connection.

This file should contain lines of the following format:

```
hostname:port:username:password
```

(You can add a reminder comment to the file by copying the line above and preceding it with `#`.) Each of the first three fields can be a literal value, or `*`, which matches anything. The password field from the first line that matches the current connection parameters will be used. (Therefore, put more-specific entries first when you are using wildcards.) If an entry needs to contain `:` or `\`, escape this character with `\`. A host name of `localhost` matches both TCP (host name `localhost`) and Unix domain socket connections coming from the local machine.

The permissions on `.pcppass` must disallow any access to world or group; achieve this by the command `chmod 0600 ~/.pcppass`. If the permissions are less strict than this, the file will be ignored.

Table of Contents

[pcp_common_options](#) -- common options used in PCP commands

[pcp_node_count](#) -- displays the total number of database nodes

[pcp_node_info](#) -- displays the information on the given node ID

[pcp_watchdog_info](#) -- displays the watchdog status of the Pgpool-II

[pcp_proc_count](#) -- displays the list of Pgpool-II children process IDs

[pcp_proc_info](#) -- displays the information on the given Pgpool-II child process ID

[pcp_pool_status](#) -- displays the parameter values as defined in `pgpool.conf`

[pcp_detach_node](#) -- detaches the given node from Pgpool-II. Existing connections to Pgpool-II are forced to be disconnected.

[pcp_attach_node](#) -- attaches the given node to Pgpool-II.

[pcp_promote_node](#) -- promotes the given node as new master to Pgpool-II

[pcp_stop_pgpool](#) -- terminates the Pgpool-II process

[pcp_recovery_node](#) -- attaches the given backend node with recovery

pcp_common_options

Name

`pcp_common_options` -- common options used in PCP commands

Synopsis

`pcp_command` [**option**...]

Description

There are some arguments common to all PCP commands. Most of these are for authentication and the rest are about verbose mode, debug message, and so on.

Options

`-h hostname`
`--host=hostname`

The host name of the machine on which the server is running. If the value begins with a slash, it is used as the directory for the Unix-domain socket.

`-p port`
`--port=port`

The PCP port number (default:"9898").

`-U username`
`--username=username`

The user name for PCP authentication (default: OS user name).

`-w`
`--no-password`

Never prompt for password. And if a password is not available by a `.pcppass` file, the connection attempt will fail. This option can be useful in batch jobs and scripts where no user is present to enter a password.

`-W`
`--password`

Force password prompt (should happen automatically).

`-d`
`--debug`

Enable debug message.

`-v`
`--verbose`

Enable verbose output.

-V
--version

Print the command version, then exit.

-?
--help

Shows help for the command line arguments, then exit.

Environment

PCPPASSFILE

Specifies the path to pcp password file.

pcp_node_count

Name

pcp_node_count -- displays the total number of database nodes

Synopsis

pcp_node_count [**option...**]

Description

pcp_node_count displays the total number of database nodes defined in `pgpool.conf`. It does not distinguish between nodes status, ie attached/detached. ALL nodes are counted.

Options

See [pcp_common_options](#).

Example

Here is an example output:

```
$ pcp_node_count -p 11001
Password:
2
```

pcp_node_info

Name

pcp_node_info -- displays the information on the given node ID

Synopsis

pcp_node_info [**option...**] [**node_id**]

Description

pcp_node_info displays the information on the given node ID.

Options

```
-n node_id
--node-id=node_id
```

The index of backend node to get information of.

Other options

See [pcp_common_options](#).

Example

Here is an example output:

```
$ pcp_node_info -h localhost -U postgres 1
/tmp 11003 2 0.500000 up standby 0 streaming async 2019-04-23 13:58:40
```

The result is in the following order:

1. hostname
2. port number
3. status
4. load balance weight
5. status name
6. backend role
7. replication delay
8. replication state (taken from pg_stat_replication, if PostgreSQL is 9.1 or later)
9. sync replication state (taken from pg_stat_replication, if PostgreSQL is 9.2 or later)
10. last status change time

Status is represented by a digit from [0 to 3]. To correctly 7, 8, 9 are displayed, [sr_check_period](#) must not be 0. 8, 9 will not be displayed if [sr_check_user](#) is not PostgreSQL super user nor it's not in "pg_monitor" group.

Note: To make [sr_check_user](#) in pg_monitor group, execute following SQL command by PostgreSQL super user (replace "sr_check_user" with the setting of [sr_check_user](#)):

```
GRANT pg_monitor TO sr_check_user;
```

For PostgreSQL 9.6, there's no pg_monitor group and [sr_check_user](#) must be PostgreSQL super user.

- 0 - This state is only used during the initialization. PCP will never display it.
- 1 - Node is up. No connections yet.
- 2 - Node is up. Connections are pooled.
- 3 - Node is down.

The load balance weight is displayed in normalized format.

The `--verbose` option can help understand the output. For example:

```
$ pcp_node_info --verbose -h localhost -U postgres 1
Hostname      : /tmp
Port         : 11003
Status       : 2
Weight       : 0.500000
Status Name   : up
Role         : standby
Replication Delay : 0
Replication State : streaming
Replication Sync State : async
Last Status Change : 2019-04-23 13:58:40
```

pcp_watchdog_info

Name

pcp_watchdog_info -- displays the watchdog status of the Pgpool-II

Synopsis

pcp_watchdog_info [**options...**] [**watchdog_id**]

Description

pcp_node_info displays the information on the given node ID.

Options

-n watchdog_id
--node-id=watchdog_id

The index of other Pgpool-II to get information for.

Index 0 gets one's self watchdog information.

If omitted then gets information of all watchdog nodes.

Other options

See [pcp_common_options](#).

Example

Here is an example output:

```
$ pcp_watchdog_info -h localhost -U postgres

3 NO Linux_host1.localdomain_9991 host1

Linux_host1.localdomain_9991 host1 9991 9001 7 STANDBY
Linux_host2.localdomain_9992 host2 9992 9002 4 MASTER
Linux_host3.localdomain_9993 host3 9993 9003 7 STANDBY
```

The result is in the following order:

The first output line describes the watchdog cluster information:

1. Total watchdog nodes in the cluster
2. Is VIP is up on current node?
3. Master node name
4. Master node host

Next is the list of watchdog nodes:

1. node name
2. hostname
3. pgpool port
4. watchdog port
5. current node state
6. current node state name

The `--verbose` option can help understand the output. For example:

```
$ pcp_watchdog_info -h localhost -v -U postgres

Watchdog Cluster Information
Total Nodes      : 3
Remote Nodes    : 2
Quorum state    : QUORUM EXIST
Alive Remote Nodes : 2
VIP up on local node : NO
Master Node Name  : Linux_host2.localdomain_9992
Master Host Name  : localhost

Watchdog Node Information
Node Name       : Linux_host1.localdomain_9991
Host Name       : host1
Delegate IP     : 192.168.1.10
Pgpool port     : 9991
Watchdog port   : 9001
Node priority   : 1
Status         : 7
Status Name     : STANDBY

Node Name       : Linux_host2.localdomain_9992
Host Name       : host2
Delegate IP     : 192.168.1.10
Pgpool port     : 9992
Watchdog port   : 9002
Node priority   : 1
Status         : 4
Status Name     : MASTER

Node Name       : Linux_host3.localdomain_9993
Host Name       : host3
Delegate IP     : 192.168.1.10
Pgpool port     : 9993
Watchdog port   : 9003
Node priority   : 1
Status         : 7
Status Name     : STANDBY
```

pcp_proc_count

Name

`pcp_proc_count --` displays the list of Pgpool-II children process IDs

Synopsis

pcp_proc_count [**options...**]

Description

pcp_proc_count displays the list of Pgpool-II children process IDs. If there is more than one process, IDs will be delimited by a white space.

Options

See [pcp_common_options](#).

pcp_proc_info

Name

pcp_proc_info -- displays the information on the given Pgpool-II child process ID

Synopsis

```
pcp_proc_info [options...] [pgpool_child_processid]
```

Description

pcp_proc_info displays the information on the given Pgpool-II child process ID.

Options

-a
--all

Display all child processes and their available connection slots.

-P *PID*
--process-id=*PID*

PID of Pgpool-II child process.

Other options

See [pcp_common_options](#).

If -a nor -P is not specified, process information of all connected Pgpool-II child process will be printed. In this case if there's no connected Pgpool-II child process, nothing but "No process information available" message will be printed.

Example

Here is an example output:

```
$ pcp_proc_info -p 11001 1406
test t-ishii 2018-07-09 16:43:53 2018-07-09 16:44:08 3 0 1 1435 1 1406 0
test t-ishii 2018-07-09 16:43:53 2018-07-09 16:44:08 3 0 1 1436 1 1406 1
```

The result is in the following order:

1. connected database name
2. connected user name
3. process start-up timestamp
4. connection created timestamp
5. protocol major version
6. protocol minor version
7. connection-reuse counter
8. PostgreSQL backend process id
9. 1 if frontend connected 0 if not
10. pgpool child process id
11. PostgreSQL backend id

If `-a` or `--all` option is not specified and there is no connection to the backends, nothing will be displayed. If there are multiple connections, one connection's information will be displayed on each line multiple times. Timestamps are displayed in EPOCH format.

The `--verbose` option can help understand the output. For example:

```
$ pcp_proc_info -p 11001 --verbose 1406
Database   : test
Username   : t-ishii
Start time : 2018-07-09 16:43:53
Creation time: 2018-07-09 16:44:08
Major      : 3
Minor      : 0
Counter    : 1
Backend PID : 1435
Connected  : 1
PID        : 1406
Backend ID : 0
Database   : test
Username   : t-ishii
Start time : 2018-07-09 16:43:53
Creation time: 2018-07-09 16:44:08
Major      : 3
Minor      : 0
Counter    : 1
Backend PID : 1436
Connected  : 1
PID        : 1406
Backend ID : 1
```

pcp_pool_status

Name

`pcp_pool_status --` displays the parameter values as defined in `pgpool.conf`

Synopsis

`pcp_pool_status [options...]`

Description

`pcp_pool_status` displays the parameter values as defined in `pgpool.conf`.

Options

See [pcp_common_options](#).

Example

Here is an example output:

```
$ pcp_pool_status -h localhost -U postgres
name : listen_addresses
value: localhost
desc : host name(s) or IP address(es) to listen to

name : port
value: 9999
desc : pgpool accepting port number

name : socket_dir
value: /tmp
desc : pgpool socket directory

name : pcp_port
value: 9898
desc : PCP port # to bind
```

pcp_detach_node

Name

`pcp_detach_node --` detaches the given node from Pgpool-II. Existing connections to Pgpool-II are forced to be disconnected.

Synopsis

```
pcp_detach_node [options...] [node_id] [gracefully]
```

Description

`pcp_detach_node` detaches the given node from Pgpool-II. Existing connections to Pgpool-II are forced to be disconnected.

Options

```
-n node_id
--node_id=node_id
```

The index of backend node to detach.

```
-g
--gracefully
```

wait until all clients are disconnected (unless `client_idle_limit_in_recovery` is -1 or `recovery_timeout` is expired).

Other options

See [pcp_common_options](#).

pcp_attach_node

Name

`pcp_attach_node --` attaches the given node to Pgpool-II.

Synopsis

```
pcp_attach_node [options...] [node_id]
```

Description

`pcp_attach_node` attaches the given node to Pgpool-II.

Options

`-n node_id`
`--node_id=node_id`

The index of backend node to attach.

Other options

See [pcp_common_options](#).

pcp_promote_node

Name

`pcp_promote_node --` promotes the given node as new master to Pgpool-II

Synopsis

`pcp_promote_node [options...] [node_id] [gracefully]`

Description

`pcp_promote_node` promotes the given node as new master to Pgpool-II. In master/slave streaming replication only. Please note that this command does not actually promote standby PostgreSQL backend: it just changes the internal status of Pgpool-II and trigger failover and users have to promote standby PostgreSQL outside Pgpool-II.

Options

`-n node_id`
`--node-id=node_id`

The index of backend node to promote as new master.

`-g`
`--gracefully`

Wait until all clients are disconnected (unless `client_idle_limit_in_recovery` is -1 or `recovery_timeout` is expired).

Other options

See [pcp_common_options](#).

pcp_stop_pgpool

Name

`pcp_stop_pgpool --` terminates the Pgpool-II process

Synopsis

`pcp_stop_pgpool [options...] [mode]`

Description

`pcp_stop_pgpool` terminates the Pgpool-II process.

Options

-m *mode*
--mode=*mode*

Shutdown mode for terminating the Pgpool-II process.

The available modes are as follows:

- s, smart : smart mode
- f, fast : fast mode
- i, immediate : immediate mode

Other options

See [pcp_common_options](#).

pcp_recovery_node

Name

pcp_recovery_node -- attaches the given backend node with recovery

Synopsis

pcp_recovery_node [**options...**] [**node_id**]

Description

pcp_recovery_node attaches the given backend node with recovery. See [Section 5.10](#) to set up necessary parameters of pgpool.conf.

Options

-n *node_id*
--node-id=*node_id*

The index of backend node.

Other options

See [pcp_common_options](#).

III. Other commands

This part contains reference information for various Pgpool-II commands.

Table of Contents

[pg_md5](#) -- produces encrypted password in md5

[pg_enc](#) -- AES256 password encryption utility

[pgproto](#) -- tests PostgreSQL or any other servers that understand the frontend/backend protocol.

[pgpool_setup](#) -- Create a temporary installation of Pgpool-II cluster

[watchdog_setup](#) -- Create a temporary installation of Pgpool-II clusters with watchdog

pg_md5

Name

pg_md5 -- produces encrypted password in md5

Synopsis

```
pg_md5 [option...] -p
```

```
pg_md5 [option...] password
```

Description

pg_md5 produces encrypted password in md5.

Options

```
-p  
--prompt
```

Prompt password using standard input.

```
-m  
--md5auth
```

Produce md5 authentication password to authentication file `pool_passwd`.

```
-u your_username  
--username=your_username
```

When producing a md5 authentication password, create the `pool_passwd` entry for `your_username`.

```
-f config_file  
--config-file=config_file
```

Specify the path to the `pgpool.conf` configuration file.

Example

The following are examples to encrypt your password into md5 hash format for `pcp.conf`.

```
$ pg_md5 -p  
password: [your password]
```

or

```
$ pg_md5 [your password]  
acbd18db4cc2f85cedef654fccc4a4d8
```

pg_md5 can also be used for adding an entry of user name and md5 encrypted password to [pool_passwd](#) authentication file.

```
$ pg_md5 -m -f /path/to/pgpool.conf -u username -p  
password: [your password]  
  
$ cat /path/to/pool_passwd  
username:md55a231fcdb710d73268c4f44283487ba2
```

To just display the md5 hashed string, not adding an entry to [pool_passwd](#), pass a string concatenating password and user name. For example, if password is "password" and user name is "user", the output would be:

```
$ pg_md5 passworduser
4d45974e13472b5a0be3533de4666414
```

Please note that the actual entry to be inserted into [pool_passwd](#) should have "md5" on top of the result string. That is: "md54d45974e13472b5a0be3533de4666414".

pg_enc

Name

pg_enc -- AES256 password encryption utility

Synopsis

pg_enc [**option...**] **-p**

pg_enc [**option...**] **password**

Description

pg_enc AES256 password encryption utility.

Options

-k *KEY_FILE*
--key-file=*KEY_FILE*

Set the path to the encryption key file. Default is the `.pgpoolkey` file located in the users home directory.

-K *ENCRYPTION_KEY*
--enc-key=*ENCRYPTION_KEY*

Encryption key to be used for encrypting database passwords.

-f *CONFIG_FILE*
--config-file=*CONFIG_FILE*

Specifies the `pgpool.conf` file.

-p
--prompt

Prompt for database password using standard input.

-P
--prompt-for-key

Prompt for encryption key using standard input.

-m
--update-pass

Create encrypted password entry in the `pool_passwd` file.

-u *your_username*
--username=*your_username*

Creates the `pool_passwd` entry for the database user called `your_username`.

-h
--help

Prints the help for `pg_enc`.

Example

Here is an example output:

```
pg_enc -p
db password: [your password]
```

or

```
./pg_enc foo
trying to read key from file /home/pgpool/.pgpoolkey

jglid1QRgiCl/vfhHUDyVA==
pool_passwd string: AESjglid1QRgiCl/vfhHUDyVA==
```

pg_enc can be used for pool_passwd passwords with:

```
pg_enc -m -f /path/to/pgpool.conf -u username -p
db password: [your password]
```

which will add an entry for `username` with the password given.

pgproto

Name

pgproto -- tests PostgreSQL or any other servers that understand the frontend/backend protocol.

Synopsis

pgproto [**option...**]

Description

pgproto tests PostgreSQL or any other servers that understand the frontend/backend protocol.

Options

`-h hostname`
`--hostname=hostname`

The host name of the machine on which the server is running. If the value begins with a slash, it is used as the directory for the Unix-domain socket (default: Unix-domain socket).

`-p port`
`--port=port`

The port number (default:5432).

`-u username`
`--user=username`

The user name (default: OS user name).

`-d databasename`
`--database=databasename`

The database name (default: same as user).

`-f filename`
`--proto-data-file=filename`

Text file describing message data to be sent to PostgreSQL (default: pgproto.data).

-r *naptime*
--read-nap=*naptime*

The nap time in micro seconds (default:0). Greater than 0 will let pgproto sleep between each data reading from socket. This is useful to simulate slow clients.

-D
--debug

Enable debug message.

-v
--version

Print the command version, then exit.

-?
--help

Shows help for the command line arguments, then exit.

Example

In the example below, the first character in the file (i.e. 'Q') indicates the message kind specified in the PostgreSQL frontend/backend protocol.

Exceptions are 'Y' and 'y'. 'Y' reads messages from backend until 'Ready for query' is received. 'y' reads messages from backend while messages are coming from backend then stops if messages are not available for 1 second. 'Y' is used for waiting for reply of 'Q' (simple query) or after 'S' (sync) in extended queries. 'y' can be used for receiving messages after 'H' (flush).

If you want to include a " (double quotation) in a string data type, for example "SELECT * FROM "aaa"", you can qualify it by using \ (back slash) like ""SELECT * FROM "aaa"" A command line spread over multiple lines can be created using \ as well.

```
'Q' "SELECT * FROM aaa \  
WHERE a = 1"
```

Here is an example input file:

```
#  
# Test data example  
#  
'Q' "SELECT * FROM aaa"  
'Y'  
'P' "S1" "BEGIN" 0  
'B' "" "S1" 0 0 0  
'E' "" 0  
'C' 'S' "S1"  
'P' "foo" "SELECT 1" 0  
'B' "myportal" "foo" 0 0 0  
'E' "myportal" 0  
'P' "S2" "COMMIT" 0  
'B' "" "S2" 0 0 0  
'E' "" 0  
'C' 'S' "S2"  
'S'  
'Y'  
'X'
```

Here is an example output:


```

$ pgproto -p 11000 -d test -f sample.data
FE=> Query (query="SELECT * FROM aaa")
<= BE RowDescription
<= BE CommandComplete(SELECT 0)
<= BE ReadyForQuery(I)
FE=> Parse(stmt="S1", query="BEGIN")
FE=> Bind(stmt="S1", portal="")
FE=> Execute(portal="")
FE=> Close(stmt="S1")
FE=> Parse(stmt="foo", query="SELECT 1")
FE=> Bind(stmt="foo", portal="myportal")
FE=> Execute(portal="myportal")
FE=> Parse(stmt="S2", query="COMMIT")
FE=> Bind(stmt="S2", portal="")
FE=> Execute(portal="")
FE=> Close(stmt="S2")
FE=> Sync
<= BE ParseComplete
<= BE BindComplete
<= BE CommandComplete(BEGIN)
<= BE CloseComplete
<= BE ParseComplete
<= BE BindComplete
<= BE DataRow
<= BE CommandComplete(SELECT 1)
<= BE ParseComplete
<= BE BindComplete
<= BE CommandComplete(COMMIT)
<= BE CloseComplete
<= BE ReadyForQuery(I)
FE=> Terminate

```

Other example data files:

Copy

```

#
# Test data example
#

# CopyIn
#
'Q' "COPY t1 FROM STDIN"
# CopyData
'd' "abc"
# CopyDone
'c'
'Y'

# CopyOut
#
'Q' "COPY t1 TO STDOUT"
'Y'

#
# Copy fail case
#
'Q' "COPY t1 FROM STDIN"
# CopyData
'd' "abc"
# CopyFail
'f' "pgproto copy fail test"
'Y'
'X'

```

Function Call

```

#
# Test data example
#

# Function call (lo_creat)
# from PostgreSQL's src/include/catalog/pg_proc.data
# { oid => '957', descr => 'large object create',
#   proname => 'lo_creat', provolatile => 'v', proparallel => 'u',
#   proreptype => 'oid', proargtypes => 'int4', prosrc => 'be_lo_creat' },

'F' 957 1 0 1 1 "0" 0
'Y'
'X'

```

pgpool_setup

Name

pgpool_setup -- Create a temporary installation of Pgpool-II cluster

Synopsis

pgpool_setup [**option**...]

Description

pgpool_setup creates a temporary installation of Pgpool-II cluster, which includes a Pgpool-II installation and specified number of PostgreSQL installations under current directory. Current directory must be empty before running pgpool_setup.

pgpool_setup is for testing purpose only and should not be used to create production installations.

Currently pgpool_setup supports streaming replication mode, native replication mode, raw mode and logical replication mode. To support watchdog, see [watchdog_setup](#) for details.

Options

pgpool_setup accepts the following command-line arguments:

-m mode

Specifies the running mode. **mode** can be *r* (native replication mode), *s* (streaming replication mode), *n* (raw mode), *l* (logical replication mode) or *y* (slony mode). If this is omitted, *s* is assumed.

-n num_clusters

Specifies the number of PostgreSQL installations. If this is omitted, 2 is used.

-p base_port

Specify the base port number used by Pgpool-II and PostgreSQL. Pgpool-II port is *base_port*. *pcp* port is *base_port* + 1. The first PostgreSQL node's port is *base_port* + 2, second PostgreSQL node's port is *base_port* + 3 and so on.

If **-pg** option is specified, the first PostgreSQL node's port is assigned to *pg_base_port*, the second PostgreSQL node's port is *pg_base_port* + 1 and so on.

If this is omitted, 11000 is used.

-pg pg_base_port

Specify the base port number used by PostgreSQL. The first PostgreSQL node's port is *base_port* + 2, second PostgreSQL node's port is *base_port* + 3 and so on.

If this is omitted, *base_port*+2 is used.

--no-stop

Do not stop pgpool and PostgreSQL after the work.

-d

Start pgpool with debug mode.

-s

In streaming replication mode, use replication slot instead of archive. Since the archive directory is shared by all PostgreSQL clusters, if a standby is promoted, the time line in the archive directory will be changed and other standby servers will be stopped. Using a replication slot does not have this problem and is always preferable if you can use PostgreSQL 9.4 or later, which supports replication slot. The replication slot name used by `pgpool_setup` is `pgpool_setup_slot`.

-r

Use `pg_rewind` command in recovery script (`basebackup.sh`). If the command fails, switch to use ordinal `rsync` command. In certain cases recovery using `pg_rewind` is much faster than `rsync` since it does not copy whole database cluster.

Environment variables

`pgpool_setup` recognizes following environment variables:

`PGPOOL_INSTALL_DIR`

Specifies the Pgpool-II installation directory. Pgpool-II binaries is expected to be placed under `PGPOOL_INSTALL_DIR/bin` and `pgpool.conf` and `pool_hba.conf` etc. are expected to be placed under `PGPOOL_INSTALL_DIR/etc`. The default is `/usr/local`.

`PGPOOLDIR`

Specifies the path to Pgpool-II configuration files. The default is `PGPOOL_INSTALL_DIR/etc`.

`PGBIN`

Specifies the path to PostgreSQL commands such as `initdb`, `pg_ctl` and `psql`. The default is `/usr/local/pgsql/bin`.

`PGLIB`

Specifies the path to PostgreSQL shared libraries. The default is `/usr/local/pgsql/lib`.

`PGSOCKET_DIR`

Specifies the path to Unix socket directory. The default is `/tmp`.

`INITDBARG`

Specifies the arguments for `initdb` command. The default is `"--no-locale -E UTF_8"`.

`USE_REPLICATION_SLOT`

If `"true"`, in streaming replication mode, use replication slot instead of archive. This brings the same effect as `"-s"` option is specified.

`USE_PG_REWIND`

If `"true"`, in streaming replication mode, use `pg_rewind` in `basebackup.sh` script. This brings the same effect as `"-r"` option is specified.

Example

```

$ pgpool_setup
Starting set up in streaming replication mode
creating startall and shutdownall
creating failover script
creating database cluster /home/t-ishii/tmp/test/data0...done.
update postgresql.conf
creating pgpool_remote_start
creating basebackup.sh
creating recovery.conf
creating database cluster /home/t-ishii/tmp/test/data1...done.
update postgresql.conf
creating pgpool_remote_start
creating basebackup.sh
creating recovery.conf
temporarily start data0 cluster to create extensions
temporarily start pgpool-II to create standby nodes
INFO: unrecognized configuration parameter "debug_level"
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay
-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | /tmp | 11002 | up | 0.500000 | primary | 0 | true | 0
1 | /tmp | 11003 | down | 0.500000 | standby | 0 | false | 0
(2 rows)

```

```

recovery node 1...pcp_recovery_node -- Command Successful
done.

```

```

creating follow master script

```

```

Pager usage is off.

```

```

node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay
-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | /tmp | 11002 | up | 0.500000 | primary | 0 | false | 0
1 | /tmp | 11003 | up | 0.500000 | standby | 0 | true | 0
(2 rows)

```

```

shutdown all

```

```

pgpool-II setting for streaming replication mode is done.

```

```

To start the whole system, use /home/t-ishii/tmp/test/startall.

```

```

To shutdown the whole system, use /home/t-ishii/tmp/test/shutdownall.

```

```

pcp command user name is "t-ishii", password is "t-ishii".

```

```

Each PostgreSQL, pgpool-II and pcp port is as follows:

```

```

#1 port is 11002

```

```

#2 port is 11003

```

```

pgpool port is 11000

```

```

pcp port is 11001

```

```

The info above is in README.port.

```

```

$ ls

```

```

README.port bashrc.ports data1 log pgpool_reload run startall

```

```

archivedir data0 etc pcppass pgpool_setup.log shutdownall

```

```

$ ./startall

```

```

waiting for server to start...11840 2016-08-18 13:08:51 JST LOG: redirecting log output to logging collector process

```

```

11840 2016-08-18 13:08:51 JST HINT: Future log output will appear in directory "pg_log".

```

```

done

```

```

server started

```

```

waiting for server to start...11853 2016-08-18 13:08:52 JST LOG: redirecting log output to logging collector process

```

```

11853 2016-08-18 13:08:52 JST HINT: Future log output will appear in directory "pg_log".

```

```

done

```

```

server started

```

```

$ psql -p 11000 test

```

```

Pager usage is off.

```

```

psql (9.5.4)

```

```

Type "help" for help.

```

```

test=# show pool_nodes;

```

```

node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay
-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | /tmp | 11002 | up | 0.500000 | primary | 0 | false | 0
1 | /tmp | 11003 | up | 0.500000 | standby | 0 | true | 0
(2 rows)

```

Name

`watchdog_setup` -- Create a temporary installation of Pgpool-II clusters with watchdog

Synopsis

`watchdog_setup` [**option**...]

Description

`watchdog_setup` creates a temporary installation of Pgpool-II clusters with watchdog enabled, which includes a Pgpool-II installation and specified number of PostgreSQL installations under current directory. Current directory must be empty before running `watchdog_setup`.

`watchdog_setup` is for testing purpose only and should not be used to create production installations. Also please note that heartbeat is not used.

`watchdog_setup` uses [pgpool_setup](#) as a workhorse.

Currently `watchdog_setup` supports streaming replication mode, native replication mode, logical replication mode and raw mode.

Options

`watchdog_setup` accepts the following command-line arguments:

`-wn num_pgpool`

Specifies the number of Pgpool-II installations. If this is omitted, 3 is used.

`-wp watchdog_base_port`

Specify the starting base port number used by Pgpool-II and PostgreSQL. For the first Pgpool-II, Pgpool-II port is `watchdog_base_port`. pcp port is `watchdog_base_port + 1`, watchdog port is `watchdog_base_port + 2`. `wd_heartbeat_port` is `watchdog_base_port + 3` (though heartbeat is not used). The first PostgreSQL node's port is `watchdog_base_port + 4`, second PostgreSQL node's port is `watchdog_base_port + 5` and so on.

If this is omitted, 50000 is used.

`-m mode`

Specifies the running mode. **mode** can be `r` (native replication mode), `s` (streaming replication mode), or `n` (raw mode). If this is omitted, `s` is used.

`-n num_clusters`

Specifies the number of PostgreSQL installations. If this is omitted, 2 is used.

`--no-stop`

Do not stop pgpool and PostgreSQL after the work.

`-d`

Start pgpool with debug mode.

Environment variables

`watchdog_setup` recognizes following environment variables:

`PGPOOL_SETUP`

Specifies the path to `pgpool_setup` command. The default is "pgpool_setup", thus it is assumed that `pgpool_setup` is in the command search path.

PGPOOL_INSTALL_DIR

Specifies the Pgpool-II installation directory. Pgpool-II binaries is expected to be placed under PGPOOL_INSTALL_DIR/bin and pgpool.conf and pool_hba.conf etc. are expected to be placed under PGPOOL_INSTALL_DIR/etc. The default is /usr/local.

PGPOOLDIR

Specifies the path to Pgpool-II configuration files. The default is PGPOOL_INSTALL_DIR/etc.

PGBIN

Specifies the path to PostgreSQL commands such as initdb, pg_ctl and psql. The default is /usr/local/pgsql/bin.

PGLIB

Specifies the path to PostgreSQL shared libraries. The default is /usr/local/pgsql/lib.

PGSOCKET_DIR

Specifies the path to Unix socket directory. The default is /tmp.

INITDBARG

Specifies the arguments for initdb command. The default is "--no-locale -E UTF_8".

Example

```
$ watchdog_setup
Starting set up
===== setting up pgpool 0 =====
Starting set up in streaming replication mode
creating startall and shutdownall
creating failover script
creating database cluster /home/t-ishii/work/pgpool-II/current/pgpool2/src/test/a/pgpool0/data0...done.
update postgresql.conf
creating pgpool_remote_start
creating basebackup.sh
creating recovery.conf
creating database cluster /home/t-ishii/work/pgpool-II/current/pgpool2/src/test/a/pgpool0/data1...done.
update postgresql.conf
creating pgpool_remote_start
creating basebackup.sh
creating recovery.conf
temporarily start data0 cluster to create extensions
temporarily start pgpool-II to create standby nodes
INFO: unrecognized configuration parameter "debug_level"
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay
-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | /tmp | 51000 | up | 0.500000 | primary | 0 | false | 0
1 | /tmp | 51001 | down | 0.500000 | standby | 0 | true | 0
(2 rows)

recovery node 1...pcp_recovery_node -- Command Successful
done.
creating follow master script
Pager usage is off.
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay
-----+-----+-----+-----+-----+-----+-----+-----+-----
0 | /tmp | 51000 | up | 0.500000 | primary | 0 | false | 0
1 | /tmp | 51001 | up | 0.500000 | standby | 0 | true | 0
(2 rows)

shutdown all

pgpool-II setting for streaming replication mode is done.
To start the whole system, use /home/t-ishii/work/pgpool-II/current/pgpool2/src/test/a/pgpool0/startall.
To shutdown the whole system, use /home/t-ishii/work/pgpool-II/current/pgpool2/src/test/a/pgpool0/shutdownall.
pcp command user name is "t-ishii", password is "t-ishii".
Each PostgreSQL, pgpool-II and pcp port is as follows:
#1 port is 51000
#2 port is 51001
pgpool port is 50000
```

pcp port is 50001

The info above is in README.port.

===== setting up pgpool 1 =====

Starting set up in streaming replication mode

creating startall and shutdownall

creating failover script

creating database cluster /home/t-ishii/work/pgpool-II/current/pgpool2/src/test/a/pgpool1/data0...done.

update postgresql.conf

creating pgpool_remote_start

creating basebackup.sh

creating recovery.conf

creating database cluster /home/t-ishii/work/pgpool-II/current/pgpool2/src/test/a/pgpool1/data1...done.

update postgresql.conf

creating pgpool_remote_start

creating basebackup.sh

creating recovery.conf

temporarily start data0 cluster to create extensions

temporarily start pgpool-II to create standby nodes

INFO: unrecognized configuration parameter "debug_level"

node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay

-----+-----+-----+-----+-----+-----+-----+-----+-----

0 | /tmp | 51000 | up | 0.500000 | primary | 0 | true | 0

1 | /tmp | 51001 | down | 0.500000 | standby | 0 | false | 0

(2 rows)

recovery node 1...pcp_recovery_node -- Command Successful

done.

creating follow master script

Pager usage is off.

node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay

-----+-----+-----+-----+-----+-----+-----+-----+-----

0 | /tmp | 51000 | up | 0.500000 | primary | 0 | true | 0

1 | /tmp | 51001 | up | 0.500000 | standby | 0 | false | 0

(2 rows)

shutdown all

pgpool-II setting for streaming replication mode is done.

To start the whole system, use /home/t-ishii/work/pgpool-II/current/pgpool2/src/test/a/pgpool1/startall.

To shutdown the whole system, use /home/t-ishii/work/pgpool-II/current/pgpool2/src/test/a/pgpool1/shutdownall.

pcp command user name is "t-ishii", password is "t-ishii".

Each PostgreSQL, pgpool-II and pcp port is as follows:

#1 port is 51000

#2 port is 51001

pgpool port is 50004

pcp port is 50005

The info above is in README.port.

===== setting up pgpool 2 =====

Starting set up in streaming replication mode

creating startall and shutdownall

creating failover script

creating database cluster /home/t-ishii/work/pgpool-II/current/pgpool2/src/test/a/pgpool2/data0...done.

update postgresql.conf

creating pgpool_remote_start

creating basebackup.sh

creating recovery.conf

creating database cluster /home/t-ishii/work/pgpool-II/current/pgpool2/src/test/a/pgpool2/data1...done.

update postgresql.conf

creating pgpool_remote_start

creating basebackup.sh

creating recovery.conf

temporarily start data0 cluster to create extensions

temporarily start pgpool-II to create standby nodes

INFO: unrecognized configuration parameter "debug_level"

node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay

-----+-----+-----+-----+-----+-----+-----+-----+-----

0 | /tmp | 51000 | up | 0.500000 | primary | 0 | true | 0

1 | /tmp | 51001 | down | 0.500000 | standby | 0 | false | 0

(2 rows)

recovery node 1...pcp_recovery_node -- Command Successful

done.

creating follow master script

Pager usage is off.

node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay

-----+-----+-----+-----+-----+-----+-----+-----+-----

0 | /tmp | 51000 | up | 0.500000 | primary | 0 | true | 0

```
1 | /tmp | 51001 | up | 0.500000 | standby | 0 | false | 0
(2 rows)
```

shutdown all

pgpool-II setting for streaming replication mode is done.

To start the whole system, use /home/t-ishii/work/pgpool-II/current/pgpool2/src/test/a/pgpool2/startall.

To shutdown the whole system, use /home/t-ishii/work/pgpool-II/current/pgpool2/src/test/a/pgpool2/shutdownall.

pcp command user name is "t-ishii", password is "t-ishii".

Each PostgreSQL, pgpool-II and pcp port is as follows:

#1 port is 51000

#2 port is 51001

pgpool port is 50008

pcp port is 50009

The info above is in README.port.

\$ ls

pgpool0 pgpool1 pgpool2 shutdownall startall

\$ sh startall

waiting for server to start....16123 2016-08-18 16:26:53 JST LOG: redirecting log output to logging collector process

16123 2016-08-18 16:26:53 JST HINT: Future log output will appear in directory "pg_log".

done

server started

waiting for server to start....16136 2016-08-18 16:26:54 JST LOG: redirecting log output to logging collector process

16136 2016-08-18 16:26:54 JST HINT: Future log output will appear in directory "pg_log".

done

server started

t-ishii@localhost: psql -p 50000 test

Pager usage is off.

psql (9.5.4)

Type "help" for help.

test=# \q

\$ pcp_watchdog_info -p 50001 -v

Password:

Watchdog Cluster Information

Total Nodes : 3

Remote Nodes : 2

Quorum state : QUORUM EXIST

Alive Remote Nodes : 2

VIP up on local node : NO

Master Node Name : Linux_tishii-CF-SX3HE4BP_50004

Master Host Name : localhost

Watchdog Node Information

Node Name : Linux_tishii-CF-SX3HE4BP_50000

Host Name : localhost

Delegate IP : Not_Set

Pgpool port : 50000

Watchdog port : 50002

Node priority : 1

Status : 7

Status Name : STANDBY

Node Name : Linux_tishii-CF-SX3HE4BP_50004

Host Name : localhost

Delegate IP : Not_Set

Pgpool port : 50004

Watchdog port : 50006

Node priority : 1

Status : 4

Status Name : MASTER

Node Name : Linux_tishii-CF-SX3HE4BP_50008

Host Name : localhost

Delegate IP : Not_Set

Pgpool port : 50008

Watchdog port : 50010

Node priority : 1

Status : 7

Status Name : STANDBY

IV. SQL type commands

This part contains reference information for various SQL type Pgpool-II commands. These commands can be issued inside the SQL session using the standard PostgreSQL client like `psql`. They are not forwarded to the backend DB: instead they are processed by Pgpool-II server. Please note that SQL type commands cannot be used in extended query mode. You will get parse errors from PostgreSQL.

Table of Contents

[PGPOOL SHOW](#) -- show the value of a configuration parameter

[PGPOOL SET](#) -- change a configuration parameter

[PGPOOL RESET](#) -- restore the value of a configuration parameter to the default value

[SHOW POOL STATUS](#) -- sends back the list of configuration parameters with their name, value, and description

[SHOW POOL NODES](#) -- sends back a list of all configured nodes

[SHOW POOL PROCESSES](#) -- sends back a list of all Pgpool-II processes waiting for connections and dealing with a connection

[SHOW POOL_POOLS](#) -- sends back a list of pools handled by Pgpool-II.

[SHOW POOL_VERSION](#) -- displays a string containing the Pgpool-II release number.

[SHOW POOL_CACHE](#) -- displays cache storage statistics

PGPOOL SHOW

Name

PGPOOL SHOW -- show the value of a configuration parameter

Synopsis

```
PGPOOL SHOW configuration_parameter
PGPOOL SHOW configuration_parameter_group
PGPOOL SHOW ALL
```

Description

PGPOOL SHOW will display the current value of Pgpool-II configuration parameters. This command is similar to the [SHOW](#) command in PostgreSQL with an addition of PGPOOL keyword to distinguish it from the PostgreSQL SHOW command.

Parameters

configuration_parameter

The name of a Pgpool-II configuration parameter. Available parameters are documented in [Chapter 5](#)

configuration_parameter_group

The name of the Pgpool-II configuration parameter group. Currently there are three parameter groups.

backend

Configuration group of all backend config parameters.

other_pgpool

Configuration group of all watchdog node config parameters.

heartbeat

configuration group of all watchdog heartbeat node config parameters.

health_check

configuration group of all health check parameters.

ALL

Show the values of all configuration parameters, with descriptions.

Examples

Show the current setting of the parameter [port](#):

```
PGPOOL SHOW port;
port
-----
9999
(1 row)
```

Show the current setting of the parameter [black_function_list](#):

```
PGPOOL SHOW black_function_list;
black_function_list
-----
nextval,setval
(1 row)
```

Show the current settings of all the configuration parameters belonging to backend group:

```
PGPOOL SHOW backend;
item          | value          | description
-----+-----+-----
backend_hostname0 | 127.0.0.1      | hostname or IP address of PostgreSQL backend.
backend_port0   | 5434           | port number of PostgreSQL backend.
backend_weight0 | 0              | load balance weight of backend.
backend_data_directory0 | /var/lib/pgsql/data | data directory of the backend.
backend_flag0   | ALLOW_TO_FAILOVER | Controls various backend behavior.
backend_hostname1 | 127.0.0.1      | hostname or IP address of PostgreSQL backend.
backend_port1   | 5432           | port number of PostgreSQL backend.
backend_weight1 | 1              | load balance weight of backend.
backend_data_directory1 | /home/work/installed/pg | data directory of the backend.
backend_flag1   | ALLOW_TO_FAILOVER | Controls various backend behavior.
(10 rows)
```

Show all settings:

```

PGPOOL SHOW ALL;
item          |      value      |      description
-----+-----+-----
backend_hostname0 | 127.0.0.1      | hostname or IP address of PostgreSQL backend.
backend_port0    | 5434           | port number of PostgreSQL backend.
backend_weight0  | 0              | load balance weight of backend.
backend_data_directory0 | /var/lib/pgsql/data | data directory of the backend.
backend_flag0    | ALLOW_TO_FAILOVER | Controls various backend behavior.
backend_hostname1 | 127.0.0.1      | hostname or IP address of PostgreSQL backend.
backend_port1    | 5432           | port number of PostgreSQL backend.
backend_weight1  | 1              | load balance weight of backend.
backend_data_directory1 | /home/work/installed/pg | data directory of the backend.
backend_flag1    | ALLOW_TO_FAILOVER | Controls various backend behavior.
other_pgpool_hostname0 | localhost      | Hostname of other pgpool node for watchdog connection.
.
.
.
ssl           | off            | Enables SSL support for frontend and backend connections
(138 rows)

```

See Also

[PGPOOL SET](#)

PGPOOL SET

Name

PGPOOL SET -- change a configuration parameter

Synopsis

```
PGPOOL SET configuration_parameter { TO | = } { value | 'value' | DEFAULT }
```

Description

The PGPOOL SET command changes the value of Pgpool-II configuration parameters for the current session. This command is similar to the SET command in PostgreSQL with an addition of PGPOOL keyword to distinguish it from the PostgreSQL SET command. Many of the configuration parameters listed in [Chapter 5](#) can be changed on-the-fly with PGPOOL SET and it only affects the value used by the current session.

Examples

Change the value of [client_idle_limit](#) parameter:

```
PGPOOL SET client_idle_limit = 350;
```

Reset the value of [client_idle_limit](#) parameter to default:

```
PGPOOL SET client_idle_limit TO DEFAULT;
```

Change the value of [log_min_messages](#) parameter:

```
PGPOOL SET log_min_messages TO INFO;
```

See Also

[PGPOOL RESET](#), [PGPOOL SHOW](#)

PGPOOL RESET

Name

PGPOOL RESET -- restore the value of a configuration parameter to the default value

Synopsis

```
PGPOOL RESET configuration_parameter  
PGPOOL RESET ALL
```

Description

PGPOOL RESET command restores the value of Pgpool-II configuration parameters to the default value. The default value is defined as the value that the parameter would have had, if no `PGPOOL SET` had ever been issued for it in the current session. This command is similar to the [RESET](#) command in PostgreSQL with an addition of PGPOOL keyword to distinguish it from the PostgreSQL RESET command.

Parameters

configuration_parameter

Name of a settable Pgpool-II configuration parameter. Available parameters are documented in [Chapter 5](#).

ALL

Resets all settable Pgpool-II configuration parameters to default values.

Examples

Reset the value of [client_idle_limit](#) parameter:

```
PGPOOL RESET client_idle_limit;
```

Reset the value of all parameter to default:

```
PGPOOL RESET ALL;
```

See Also

[PGPOOL SET](#), [PGPOOL SHOW](#)

SHOW POOL STATUS

Name

SHOW POOL STATUS -- sends back the list of configuration parameters with their name, value, and description

Synopsis

```
SHOW POOL_STATUS
```

Description

SHOW POOL_STATUS displays the current value of Pgpool-II configuration parameters.

This command is similar to the [PGPOOL SHOW](#) command, but this is the older version of it. It is recommended to use [PGPOOL SHOW](#) instead.

See Also

[PGPOOL SHOW](#)

SHOW POOL NODES

Name

SHOW POOL_NODES -- sends back a list of all configured nodes

Synopsis

```
SHOW POOL_NODES
```

Description

SHOW POOL_NODES displays the node id, the hostname, the port, the status, the weight (only meaningful if you use the load balancing mode), the role, the SELECT query counts issued to each backend, whether each node is the load balance node or not, the replication delay (only if in streaming replication mode) and last status change time. In addition to this replication state and sync state are shown for standby nodes in Pgpool-II 4.1 or after. The possible values in the status column are explained in the [pcp_node_info](#) reference. If the hostname is something like "/tmp", that means Pgpool-II is connecting to backend by using UNIX domain sockets. The SELECT count does not include internal queries used by Pgpool-II. Also the counters are reset to zero upon starting up of Pgpool-II. The last status change time is initially set to the time Pgpool-II starts. After that whenever "status" or "role" is changed, it is updated.

Here is an example session:

```
test=# show pool_nodes;
node_id | hostname | port | status | lb_weight | role | select_cnt | load_balance_node | replication_delay | replication_state | re
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
 0      | /tmp     | 11002 | up     | 0.500000 | primary | 0          | false             | 0                |                    | 2019-04-2
 1      | /tmp     | 11003 | up     | 0.500000 | standby | 0          | true              | 0                | streaming         | async      | 20
(2 rows)
```

SHOW POOL_PROCESSES

Name

SHOW POOL_PROCESSES -- sends back a list of all Pgpool-II processes waiting for connections and dealing with a connection

Synopsis

```
SHOW POOL_PROCESSES
```

Description

SHOW POOL_PROCESSES sends back a list of all Pgpool-II processes waiting for connections and dealing with a connection.

It has 6 columns:

- `pool_pid` is the PID of the displayed Pgpool-II process.
- `start_time` is the timestamp of when this process was launched.
- `database` is the database name of the currently active backend for this process.
- `username` is the user name used in the connection of the currently active backend for this process.
- `create_time` is the creation time and date of the connection.
- `pool_counter` counts the number of times this pool of connections (process) has been used by clients.

Here is an example session:

```
test=# show pool_processes;
pool_pid | start_time | database | username | create_time | pool_counter
-----+-----+-----+-----+-----+-----
19696 | 2016-10-17 13:24:17 | postgres | t-ishii | 2016-10-17 13:35:12 | 1
19697 | 2016-10-17 13:24:17 | | | | 
19698 | 2016-10-17 13:24:17 | | | | 
19699 | 2016-10-17 13:24:17 | | | | 
19700 | 2016-10-17 13:24:17 | | | | 
19701 | 2016-10-17 13:24:17 | | | | 
19702 | 2016-10-17 13:24:17 | | | | 
19703 | 2016-10-17 13:24:17 | | | | 
19704 | 2016-10-17 13:24:17 | | | | 
19705 | 2016-10-17 13:24:17 | | | | 
19706 | 2016-10-17 13:24:17 | | | | 
19707 | 2016-10-17 13:24:17 | | | | 
19708 | 2016-10-17 13:24:17 | | | | 
19709 | 2016-10-17 13:24:17 | | | | 
19710 | 2016-10-17 13:24:17 | | | | 
19711 | 2016-10-17 13:24:17 | | | | 
19712 | 2016-10-17 13:24:17 | | | | 
19713 | 2016-10-17 13:24:17 | | | | 
19714 | 2016-10-17 13:24:17 | | | | 
19715 | 2016-10-17 13:24:17 | | | | 
19716 | 2016-10-17 13:24:17 | | | | 
19717 | 2016-10-17 13:24:17 | | | | 
19718 | 2016-10-17 13:24:17 | | | | 
19719 | 2016-10-17 13:24:17 | | | | 
19720 | 2016-10-17 13:24:17 | | | | 
20024 | 2016-10-17 13:33:46 | | | | 
19722 | 2016-10-17 13:24:17 | test | t-ishii | 2016-10-17 13:34:42 | 1
19723 | 2016-10-17 13:24:17 | | | | 
19724 | 2016-10-17 13:24:17 | | | | 
19725 | 2016-10-17 13:24:17 | | | | 
19726 | 2016-10-17 13:24:17 | | | | 
19727 | 2016-10-17 13:24:17 | | | | 
(32 rows)
```

SHOW POOL_POOLS

Name

SHOW POOL_POOLS -- sends back a list of pools handled by Pgpool-II.

Summary

synopsis

```
SHOW POOL_POOLS
```

Description

SHOW POOL_POOLS sends back a list of pools handled by Pgpool-II

It has 11 columns:

- `pool_pid` is the PID of the displayed Pgpool-II process.
- `start_time` is the timestamp of when this process was launched.
- `pool_id` is the pool identifier (should be between 0 and `max_pool` - 1)
- `backend_id` is the backend identifier (should be between 0 and the number of configured backends minus one)
- `database` is the database name for this process's pool id connection.
- `username` is the user name for this process's pool id connection.
- `create_time` is the creation time and date of the connection.
- `majorversion` and `minorversion` are the protocol version numbers used in this connection.
- `pool_counter` counts the number of times this pool of connections (process) has been used by clients.
- `pool_backendpid` is the PID of the PostgreSQL process.
- `pool_connected` is true (1) if a frontend is currently using this backend.

It'll always return `num_init_children` * `max_pool` * `number_of_backends` lines. Here is an example session:

```
test=# show pool_pools;
pool_pid | start_time | pool_id | backend_id | database | username | create_time | majorversion | minorversion | pool
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
19696 | 2016-10-17 13:24:17 | 0 | 0 | postgres | t-ishii | 2016-10-17 13:35:12 | 3 | 0 | 1 | 2007
19696 | 2016-10-17 13:24:17 | 0 | 1 | postgres | t-ishii | 2016-10-17 13:35:12 | 3 | 0 | 1 | 2008
19696 | 2016-10-17 13:24:17 | 1 | 0 | | | | 0 | 0 | 0 | 0
19696 | 2016-10-17 13:24:17 | 1 | 1 | | | | 0 | 0 | 0 | 0
19696 | 2016-10-17 13:24:17 | 2 | 0 | | | | 0 | 0 | 0 | 0
19696 | 2016-10-17 13:24:17 | 2 | 1 | | | | 0 | 0 | 0 | 0
19696 | 2016-10-17 13:24:17 | 3 | 0 | | | | 0 | 0 | 0 | 0
19696 | 2016-10-17 13:24:17 | 3 | 1 | | | | 0 | 0 | 0 | 0
19697 | 2016-10-17 13:24:17 | 0 | 0 | | | | 0 | 0 | 0 | 0
19697 | 2016-10-17 13:24:17 | 0 | 1 | | | | 0 | 0 | 0 | 0
19697 | 2016-10-17 13:24:17 | 1 | 0 | | | | 0 | 0 | 0 | 0
19697 | 2016-10-17 13:24:17 | 1 | 1 | | | | 0 | 0 | 0 | 0
19697 | 2016-10-17 13:24:17 | 2 | 0 | | | | 0 | 0 | 0 | 0
19697 | 2016-10-17 13:24:17 | 2 | 1 | | | | 0 | 0 | 0 | 0
19697 | 2016-10-17 13:24:17 | 3 | 0 | | | | 0 | 0 | 0 | 0
19697 | 2016-10-17 13:24:17 | 3 | 1 | | | | 0 | 0 | 0 | 0
19698 | 2016-10-17 13:24:17 | 0 | 0 | | | | 0 | 0 | 0 | 0
19698 | 2016-10-17 13:24:17 | 0 | 1 | | | | 0 | 0 | 0 | 0
19698 | 2016-10-17 13:24:17 | 1 | 0 | | | | 0 | 0 | 0 | 0
19698 | 2016-10-17 13:24:17 | 1 | 1 | | | | 0 | 0 | 0 | 0
19698 | 2016-10-17 13:24:17 | 2 | 0 | | | | 0 | 0 | 0 | 0
19698 | 2016-10-17 13:24:17 | 2 | 1 | | | | 0 | 0 | 0 | 0
19698 | 2016-10-17 13:24:17 | 3 | 0 | | | | 0 | 0 | 0 | 0
19698 | 2016-10-17 13:24:17 | 3 | 1 | | | | 0 | 0 | 0 | 0
19699 | 2016-10-17 13:24:17 | 0 | 0 | | | | 0 | 0 | 0 | 0
19699 | 2016-10-17 13:24:17 | 0 | 1 | | | | 0 | 0 | 0 | 0
19699 | 2016-10-17 13:24:17 | 1 | 0 | | | | 0 | 0 | 0 | 0
19699 | 2016-10-17 13:24:17 | 1 | 1 | | | | 0 | 0 | 0 | 0
19699 | 2016-10-17 13:24:17 | 2 | 0 | | | | 0 | 0 | 0 | 0
19699 | 2016-10-17 13:24:17 | 2 | 1 | | | | 0 | 0 | 0 | 0
19699 | 2016-10-17 13:24:17 | 3 | 0 | | | | 0 | 0 | 0 | 0
19699 | 2016-10-17 13:24:17 | 3 | 1 | | | | 0 | 0 | 0 | 0
```


19719	2016-10-17 13:24:17	1	0				0	0	0	0	0
19719	2016-10-17 13:24:17	1	1				0	0	0	0	0
19719	2016-10-17 13:24:17	2	0				0	0	0	0	0
19719	2016-10-17 13:24:17	2	1				0	0	0	0	0
19719	2016-10-17 13:24:17	3	0				0	0	0	0	0
19719	2016-10-17 13:24:17	3	1				0	0	0	0	0
19720	2016-10-17 13:24:17	0	0				0	0	0	0	0
19720	2016-10-17 13:24:17	0	1				0	0	0	0	0
19720	2016-10-17 13:24:17	1	0				0	0	0	0	0
19720	2016-10-17 13:24:17	1	1				0	0	0	0	0
19720	2016-10-17 13:24:17	2	0				0	0	0	0	0
19720	2016-10-17 13:24:17	2	1				0	0	0	0	0
19720	2016-10-17 13:24:17	3	0				0	0	0	0	0
19720	2016-10-17 13:24:17	3	1				0	0	0	0	0
20024	2016-10-17 13:33:46	0	0	test	t-ishii	2016-10-17 14:30:53	3	0	0	1	22055
20024	2016-10-17 13:33:46	0	1	test	t-ishii	2016-10-17 14:30:53	3	0	0	1	22056
20024	2016-10-17 13:33:46	1	0				0	0	0	0	0
20024	2016-10-17 13:33:46	1	1				0	0	0	0	0
20024	2016-10-17 13:33:46	2	0				0	0	0	0	0
20024	2016-10-17 13:33:46	2	1				0	0	0	0	0
20024	2016-10-17 13:33:46	3	0				0	0	0	0	0
20024	2016-10-17 13:33:46	3	1				0	0	0	0	0
20600	2016-10-17 13:46:58	0	0				0	0	0	0	0
20600	2016-10-17 13:46:58	0	1				0	0	0	0	0
20600	2016-10-17 13:46:58	1	0				0	0	0	0	0
20600	2016-10-17 13:46:58	1	1				0	0	0	0	0
20600	2016-10-17 13:46:58	2	0				0	0	0	0	0
20600	2016-10-17 13:46:58	2	1				0	0	0	0	0
20600	2016-10-17 13:46:58	3	0				0	0	0	0	0
20600	2016-10-17 13:46:58	3	1				0	0	0	0	0
19723	2016-10-17 13:24:17	0	0				0	0	0	0	0
19723	2016-10-17 13:24:17	0	1				0	0	0	0	0
19723	2016-10-17 13:24:17	1	0				0	0	0	0	0
19723	2016-10-17 13:24:17	1	1				0	0	0	0	0
19723	2016-10-17 13:24:17	2	0				0	0	0	0	0
19723	2016-10-17 13:24:17	2	1				0	0	0	0	0
19723	2016-10-17 13:24:17	3	0				0	0	0	0	0
19723	2016-10-17 13:24:17	3	1				0	0	0	0	0
19724	2016-10-17 13:24:17	0	0				0	0	0	0	0
19724	2016-10-17 13:24:17	0	1				0	0	0	0	0
19724	2016-10-17 13:24:17	1	0				0	0	0	0	0
19724	2016-10-17 13:24:17	1	1				0	0	0	0	0
19724	2016-10-17 13:24:17	2	0				0	0	0	0	0
19724	2016-10-17 13:24:17	2	1				0	0	0	0	0
19724	2016-10-17 13:24:17	3	0				0	0	0	0	0
19724	2016-10-17 13:24:17	3	1				0	0	0	0	0
19725	2016-10-17 13:24:17	0	0				0	0	0	0	0
19725	2016-10-17 13:24:17	0	1				0	0	0	0	0
19725	2016-10-17 13:24:17	1	0				0	0	0	0	0
19725	2016-10-17 13:24:17	1	1				0	0	0	0	0
19725	2016-10-17 13:24:17	2	0				0	0	0	0	0
19725	2016-10-17 13:24:17	2	1				0	0	0	0	0
19725	2016-10-17 13:24:17	3	0				0	0	0	0	0
19725	2016-10-17 13:24:17	3	1				0	0	0	0	0
19726	2016-10-17 13:24:17	0	0				0	0	0	0	0
19726	2016-10-17 13:24:17	0	1				0	0	0	0	0
19726	2016-10-17 13:24:17	1	0				0	0	0	0	0
19726	2016-10-17 13:24:17	1	1				0	0	0	0	0
19726	2016-10-17 13:24:17	2	0				0	0	0	0	0
19726	2016-10-17 13:24:17	2	1				0	0	0	0	0
19726	2016-10-17 13:24:17	3	0				0	0	0	0	0
19726	2016-10-17 13:24:17	3	1				0	0	0	0	0
19727	2016-10-17 13:24:17	0	0				0	0	0	0	0
19727	2016-10-17 13:24:17	0	1				0	0	0	0	0
19727	2016-10-17 13:24:17	1	0				0	0	0	0	0
19727	2016-10-17 13:24:17	1	1				0	0	0	0	0
19727	2016-10-17 13:24:17	2	0				0	0	0	0	0
19727	2016-10-17 13:24:17	2	1				0	0	0	0	0
19727	2016-10-17 13:24:17	3	0				0	0	0	0	0
19727	2016-10-17 13:24:17	3	1				0	0	0	0	0

(256 rows)

SHOW POOL_VERSION

Name

SHOW POOL_VERSION -- displays a string containing the Pgpool-II release number.

Synopsis

```
SHOW POOL_VERSION
```

Description

SHOW POOL_VERSION displays a string containing the Pgpool-II release number. Here is an example session:

```
test=# show pool_version;
pool_version
-----
3.6.0 (subaruboshi)
(1 row)
```

SHOW POOL_CACHE

Name

SHOW POOL_CACHE -- displays cache storage statistics

Synopsis

```
SHOW POOL_CACHE
```

Description

SHOW POOL_CACHE displays [in memory query cache](#) statistics if in memory query cache is enabled. Here is an example session:

```
test=# \x
\x
Expanded display is on.
test=# show pool_cache;
show pool_cache;
-[ RECORD 1 ]-----+-----
num_cache_hits      | 891703
num_selects         | 99995
cache_hit_ratio     | 0.90
num_hash_entries    | 131072
used_hash_entries   | 99992
num_cache_entries   | 99992
used_cache_entries_size | 12482600
free_cache_entries_size | 54626264
fragment_cache_entries_size | 0
```

V. pgpool_adm extension

pgpool_adm is a set of extensions to allow SQL access to [Reference II, PCP commands](#) (actually, pcp libraries). It uses foreign data wrapper as shown in the diagram below.

Figure 1. How pgpool_adm works

It is possible to call the functions from either via pgpool-II (1) or via PostgreSQL (2). In case (1), Pgpool-II accepts query from user (1), then forward to PostgreSQL (3). PostgreSQL connects to Pgpool-II (5) and Pgpool-II reply back to PostgreSQL with the result (3). PostgreSQL returns the result to Pgpool-II (5) and Pgpool-II forwards the data to the user (6).

In case (2), PostgreSQL accepts query from user (2). PostgreSQL connects to Pgpool-II (5) and Pgpool-II reply back to PostgreSQL with the result (3). PostgreSQL replies back the data to the user (6).

There are two forms to call pgpool_adm functions: first form accepts Pgpool-II host name (or IP address), pcp port number, pcp user name, its password and another parameters.

In the second form, Pgpool-II server name is required. The server name must be already defined using "CREATE FOREIGN SERVER" command of PostgreSQL. The pcp port number is hard coded as 9898, the pcp user name is assumes to be same as caller's PostgreSQL user name. password is extracted from \$HOME/.pcppass.

1. Installing pgpool_adm

pgpool_adm is an extension and should be installed on all PostgreSQL servers.

```
$ cd src/sql/pgpool_adm
$ make
$ make install
```

Then issue following SQL command for every database you want to access.

```
$ psql ...
$ CREATE EXTENSION pgpool_adm
```

Table of Contents

[pgpool_adm_pcp_node_info](#) -- a function to display the information on the given node ID

[pgpool_adm_pcp_pool_status](#) -- a function to retrieves parameters in pgpool.conf.

[pgpool_adm_pcp_node_count](#) -- a function to retrieves number of backend nodes.

[pgpool_adm_pcp_attach_node](#) -- a function to attach given node ID

[pgpool_adm_pcp_detach_node](#) -- a function to detach given node ID

pgpool_adm_pcp_node_info

Name

pgpool_adm_pcp_node_info -- a function to display the information on the given node ID

Synopsis

pcp_node_info returns record(integer node_id, text host, integer port, text username, text password, out status text, out weight float4, out role text, out replication_delay bigint, out replication_state text, out replication_sync_state text, out last_status_change timestamp);

pcp_node_info returns record(integer node_id, text pcp_server, out status text, out weight float4, out role text, out replication_delay bigint, out replication_state text, out replication_sync_state text, out last_status_change timestamp);

Description

pcp_node_info displays the information on the given node ID.

Arguments

node_id

The index of backend node to get information of.

pcp_server

The foreign server name for pcp server.

Other arguments

See [pcp_common_options](#).

Example

Here is an example output:

```
test=# SELECT * FROM pcp_node_info(1,'',11001,'t-ishii','t-ishii');
 host | port | status      | weight | role | replication_delay | replication_state | replication_sync_state | last_status_change
-----+-----+-----+-----+-----+-----+-----+-----+-----
 /tmp | 11003 | Connection in use | 0 | Standby | 0 | streaming | async | 2019-04-23 15:02:46
(1 row)
```

Note: `role`, `replication_delay`, `last_status_change` out parameters are new from Pgpool-II 4.0. If you have already installed pre-4.0 `pgpool_adm` extension, you can upgrade to the new one by using ALTER EXTENSION SQL command.

```
ALTER EXTENSION pgpool_adm UPDATE;
```

Note: `replication_state` and `replication_sync_state` out parameters are new from Pgpool-II 4.1. If you have already installed pre-4.1 `pgpool_adm` extension, you can upgrade to the new one by using ALTER EXTENSION SQL command.

```
ALTER EXTENSION pgpool_adm UPDATE;
```

pgpool_adm_pcp_pool_status

Name

`pgpool_adm_pcp_pool_status` -- a function to retrieves parameters in `pgpool.conf`.

Synopsis

`pcp_pool_status` returns record(text host, integer port, text username, text password, out item text, out value text, out description text);

`pcp_pool_status` returns record(text pcp_server, out item text, out value text, out description text);

Description

pcp_pool_status retrieves parameters in pgpool.conf.

Arguments

pcp_server

The foreign server name for pcp server.

Other arguments

See [pcp_common_options](#).

Example

Here is an example output:

```
test=# SELECT * FROM pcp_pool_status('localhost',11001,'t-ishii','t-ishii') WHERE item ~ 'backend.*0';
item      |          value          | description
-----+-----+-----
backend_hostname0 | /tmp                    | backend #0 hostname
backend_port0    | 11002                   | backend #0 port number
backend_weight0  | 0.500000                | weight of backend #0
backend_data_directory0 | /home/t-ishii/work/pgpool-II/current/aaa/data0 | data directory for backend #0
backend_status0  | 2                        | status of backend #0
backend_flag0   | ALLOW_TO_FAILOVER       | backend #0 flag
(6 rows)
```

pgpool_adm_pcp_node_count

Name

pgpool_adm_pcp_node_count -- a function to retrieves number of backend nodes.

Synopsis

pcp_node_count returns integer(text host, integer port, text username, text password);

pcp_node_count returns integer(text pcp_server);

Description

pcp_node_count retrieves number of DB nodes.

Arguments

pcp_server

The foreign server name for pcp server.

Other arguments

See [pcp_common_options](#).

Example

Here is an example output:

```
test=# SELECT * FROM pcp_node_count('localhost',11001,'t-ishii','t-ishii');
node_count
-----
2
(1 row)
```

pgpool_adm_pcp_attach_node

Name

pgpool_adm_pcp_attach_node -- a function to attach given node ID

Synopsis

pcp_attach_node returns record(integer node_id, text host, integer port, text username, text password, out node_attached boolean);

pcp_attach_node returns record(integer node_id, text pcp_server, out node_attached boolean);

Description

pcp_attach_node attaches a node to Pgpool-II.

Arguments

node_id

The index of backend node to attach.

pcp_server

The foreign server name for pcp server.

Other arguments

See [pcp_common_options](#).

Example

Here is an example output:

```
test=# SELECT * FROM pcp_attach_node(1,'localhost',11001,'t-ishii','t-ishii');
node_attached
-----
t
(1 row)
```

pgpool_adm_pcp_detach_node

Name

pgpool_adm_pcp_detach_node -- a function to detach given node ID

Synopsis

pcp_detach_node returns record(integer node_id, boolean gracefully, text host, integer port, text username, text password, out node_detached boolean);

pcp_detach_node returns record(integer node_id, boolean gracefully, text pcp_server, out node_detached boolean);

Description

pcp_detach_node detaches a node from Pgpool-II.

Arguments

node_id

The index of backend node to detach.

gracefully

If true, wait for all session of pgpool-II terminates.

pcp_server

The foreign server name for pcp server.

Other arguments

See [pcp_common_options](#).

Example

Here is an example output:

```
test=# SELECT * FROM pcp_detach_node(1, 'false', 'localhost',11001,'t-ishii','t-ishii');
node_detached
-----
t
(1 row)
```

V. Appendixes

something...

Table of Contents

A. [Release Notes](#)

- A.1. [Release 4.1.2](#)
- A.2. [Release 4.1.1](#)
- A.3. [Release 4.1.0](#)
- A.4. [Release 4.0.9](#)
- A.5. [Release 4.0.8](#)
- A.6. [Release 4.0.7](#)
- A.7. [Release 4.0.6](#)
- A.8. [Release 4.0.5](#)
- A.9. [Release 4.0.4](#)
- A.10. [Release 4.0.3](#)
- A.11. [Release 4.0.2](#)
- A.12. [Release 4.0.1](#)
- A.13. [Release 4.0](#)
- A.14. [Release 3.7.14](#)
- A.15. [Release 3.7.13](#)
- A.16. [Release 3.7.12](#)
- A.17. [Release 3.7.11](#)
- A.18. [Release 3.7.10](#)
- A.19. [Release 3.7.9](#)
- A.20. [Release 3.7.8](#)
- A.21. [Release 3.7.7](#)
- A.22. [Release 3.7.6](#)
- A.23. [Release 3.7.5](#)
- A.24. [Release 3.7.4](#)
- A.25. [Release 3.7.3](#)

A.26. [Release 3.7.2](#)
A.27. [Release 3.7.1](#)
A.28. [Release 3.7](#)
A.29. [Release 3.6.21](#)
A.30. [Release 3.6.20](#)
A.31. [Release 3.6.19](#)
A.32. [Release 3.6.18](#)
A.33. [Release 3.6.17](#)
A.34. [Release 3.6.16](#)
A.35. [Release 3.6.15](#)
A.36. [Release 3.6.14](#)
A.37. [Release 3.6.13](#)
A.38. [Release 3.6.12](#)
A.39. [Release 3.6.11](#)
A.40. [Release 3.6.10](#)
A.41. [Release 3.6.9](#)
A.42. [Release 3.6.8](#)
A.43. [Release 3.6.7](#)
A.44. [Release 3.6.6](#)
A.45. [Release 3.6.5](#)
A.46. [Release 3.6.4](#)
A.47. [Release 3.6.3](#)
A.48. [Release 3.6.2](#)
A.49. [Release 3.6.1](#)
A.50. [Release 3.6](#)
A.51. [Release 3.5.25](#)
A.52. [Release 3.5.24](#)
A.53. [Release 3.5.23](#)
A.54. [Release 3.5.22](#)
A.55. [Release 3.5.21](#)
A.56. [Release 3.5.20](#)
A.57. [Release 3.5.19](#)
A.58. [Release 3.5.18](#)
A.59. [Release 3.5.17](#)
A.60. [Release 3.5.16](#)
A.61. [Release 3.5.15](#)
A.62. [Release 3.5.14](#)
A.63. [Release 3.5.13](#)
A.64. [Release 3.5.12](#)
A.65. [Release 3.5.11](#)
A.66. [Release 3.5.10](#)
A.67. [Release 3.5.9](#)
A.68. [Release 3.5.8](#)
A.69. [Release 3.5.7](#)
A.70. [Release 3.5.6](#)
A.71. [Release 3.5.5](#)
A.72. [Release 3.4.26](#)
A.73. [Release 3.4.25](#)
A.74. [Release 3.4.24](#)
A.75. [Release 3.4.23](#)
A.76. [Release 3.4.22](#)
A.77. [Release 3.4.21](#)
A.78. [Release 3.4.20](#)
A.79. [Release 3.4.19](#)
A.80. [Release 3.4.18](#)
A.81. [Release 3.4.17](#)
A.82. [Release 3.4.16](#)
A.83. [Release 3.4.15](#)
A.84. [Release 3.4.14](#)
A.85. [Release 3.4.13](#)
A.86. [Release 3.4.12](#)
A.87. [Release 3.4.11](#)

- A.88. [Release 3.4.10](#)
 - A.89. [Release 3.4.9](#)
 - A.90. [Release 3.3.22](#)
 - A.91. [Release 3.3.21](#)
 - A.92. [Release 3.3.20](#)
 - A.93. [Release 3.3.19](#)
 - A.94. [Release 3.3.18](#)
 - A.95. [Release 3.3.17](#)
 - A.96. [Release 3.3.16](#)
 - A.97. [Release 3.3.15](#)
 - A.98. [Release 3.3.14](#)
 - A.99. [Release 3.3.13](#)
 - A.100. [Release 3.2.22](#)
 - A.101. [Release 3.2.21](#)
 - A.102. [Release 3.2.20](#)
 - A.103. [Release 3.2.19](#)
 - A.104. [Release 3.2.18](#)
 - A.105. [Release 3.1.21](#)
-

Appendix A. Release Notes

The release notes contain the significant changes in each Pgpool-II release, with major features and migration issues listed at the top. The release notes do not contain changes that affect only a few users or changes that are internal and therefore not user-visible.

A complete list of changes for each release can be obtained by viewing the Git logs for each release. The [pgpool-committers email list](#) records all source code changes as well. There is also a [web interface](#) that shows changes to specific files.

The name appearing next to each item represents the major developer for that item. Of course all changes involve community discussion and patch review, so each item is truly a community effort.

A.1. Release 4.1.2

Release Date: 2020-05-21

A.1.1. Bug fixes

- Fix build error on some system (Fedora 32). (Tatsuo Ishii)
- Doc: Add note about [if_up_cmd](#) and [if_down_cmd](#) command. (Bo Peng)
- Fix pgpool ssl front end accept all ciphers. ([bug 608](#)) (Muhammad Usama)
- Downgrade too verbose authentication logs. (Tatsuo Ishii)
- Fix unnecessary checks. (Tatsuo Ishii)

Patch contributed by sherlockcpp. Discussion: [\[pgpool-general: 7004\]](#)

- Doc: Fix typo. (Tatsuo Ishii)

Patch contributed by Umar Hayat. Discussion: [\[pgpool-hackers: 3587\]](#)

- Fix for segmentation fault in PCP process. ([bug 598](#)) (Muhammad Usama)
- Fix a warning message is never output in watchdog. (Tatsuo Ishii)

Patch provided by sherlockcpp. Discussion: ([\[pgpool-general: 6956\]](#))

- Doc: Fix typo. (Tatsuo Ishii, Bo Peng)

Patch provided by sherlockcpp.

- Fix Pgpool-II hangs when an Execute message is issued right after Sync message and query cache hits. (Tatsuo Ishii)

- Fix problems in watchdog source code processing json data. ([bug 596](#)) (Tatsuo Ishii)

Patch is provided by Greg Nancarrow (Fujitsu Australia).

- Fix SCRAM auth handling bug. ([bug 596](#)) (Tatsuo Ishii)

Patch is provided by Greg Nancarrow (Fujitsu Australia).

- Fix possible data inconsistency in native replication mode. (Tatsuo Ishii)

Discussions: ([\[pgpool-general: 6896\]](#)) ([\[pgpool-hackers: 3540\]](#))

- Fix watchdog ping probes fail with long hostnames due to small buffer. ([bug 516](#)) (Bo Peng)

- Doc: Enhance documents. (Tatsuo Ishii)

- Fix "last status change" timestamp is not set properly. (Tatsuo Ishii)

A.2. Release 4.1.1

Release Date: 2020-02-20

A.2.1. Changes

- Disallowing the quorum aware failover option for the native replication mode. (Muhammad Usama)

In native replication mode, Pgpool-II is responsible for replicating the data on all backend nodes, and if a node becomes *quarantined* then Pgpool-II stops sending the writes to that node. This is dangerous since it can cause data inconsistency.

So as per the discussion, we reached the conclusion to disallow [failover when quorum exists](#) with the native replication mode so that backend node should never get into quarantine when Pgpool-II is configured in the replication mode.

- Check if socket file exists at startup and remove them if PID file doesn't exist to avoid bind() failure. (Bo Peng)

A.2.2. Bug fixes

- Fix incorrect query rewriting in native replication mode. ([bug 551](#)) (Bo Peng)

- Doc: Update documentation and fix documentation typos. (Takuma Hoshiai, Tatsuo Ishii, Bo Peng)

- Update pgpool-recovery function definitions. (Tatsuo Ishii)

- Fix child process segfault after reload if [health_check_database](#) is empty. ([bug 571](#)) (Bo Peng)

- Suppress unnecessary error message when there's no standby server. (Tatsuo Ishii)

- Fix [pgpool_setup](#) to support PostgreSQL 12. (Tatsuo Ishii)

- Doc: Change the sample `follow_master_command` script `%M %H` parameter's order. (Bo Peng)
 - Fix occasional regression test failure. (Tatsuo Ishii)
 - Unbreak notification response message treatment in 4.1. ([bug 573](#)) (Tatsuo Ishii)
 - Fix replication delay worker segfault when `application_name` in `primary_conninfo` is an empty string. ([bug 565](#)) (Tatsuo Ishii)
 - Fix `PGPOOL_SHOW` command doesn't display `ALWAYS_MASTER`, when `backend_flag` = `'ALWAYS_MASTER'`. (Takuma Hoshiai)
 - Fix the bug with ignored `syslog` setting. (Bo Peng)
-

A.3. Release 4.1.0

Release Date: 2019-10-31

A.3.1. Overview

This version implements long awaited features including [statement_level_load_balance](#) and [auto_failback](#). Also it enhances number of areas related to performance. Finally it imports PostgreSQL 12's new SQL parser.

Major enhancements in Pgpool-II 4.1 include:

- Statement level load balancing. Previously Pgpool-II only allows session level load balancing. This version allows to use `statement level load balancing`, which is useful for frontends permanently connecting to Pgpool-II but want to use existing standby server resources.
 - Auto failback allows to automatically attach streaming replication standby servers, which are considered safe enough to failback.
 - Enhance performance in number of areas.
 - Shared relation cache allows to reuse relation cache among sessions to reduce internal queries against PostgreSQL system catalogs.
 - Have separate SQL parser for DML statements to eliminate unnecessary parsing effort.
 - Load balancing control for specific queries.
 - Import PostgreSQL 12 SQL parser.
-

A.3.2. Migration to Version 4.1

Version 4.1 contains some changes that may affect compatibility with previous releases. Observe the following incompatibilities:

- Add `replication_state` and `replication_sync_state` columns of [SHOW POOL NODES](#) and friends. (Tatsuo Ishii)

This allows to show important information from `pg_stat_replication`, which is available from PostgreSQL 9.1 (also with `replication_state_sync`, it's only available since 9.2 however). For this purpose new [backend_application_name](#) parameter is added to each `backend_host` configuration parameters. `pg_stat_replication` is called from streaming replication delay checking process. So if [sr_check_period](#) is 0, those new columns are not available.

Also [pcp_node_info](#) and [pgpool_adm_pcp_node_info](#) function are modified.

- Add parameter [enable_consensus_with_half_votes](#) to configure majority rule calculations. (Muhammad

Usama, Tatsuo Ishii)

This changes the behavior of the decision of quorum existence and failover consensus on even number (i.e. 2, 4, 6...) of watchdog clusters. Odd number of clusters (3, 5, 7...) are not affected. When this parameter is off (the default), a 2 node watchdog cluster needs to have both 2 nodes are alive to have a quorum. If the quorum does not exist and 1 node goes down, then 1) VIP will be lost, 2) failover script is not executed and 3) no watchdog master exists. Especially #2 could be troublesome because no new primary PostgreSQL exists if existing primary goes down. Probably 2 node watchdog cluster users want to turn on this parameter to keep the existing behavior. On the other hand 4 or more even number of watchdog cluster users will benefit from this parameter is off because now it prevents possible split brain when a half of watchdog nodes go down.

- If installing from RPMs, by default Pgpool-II is started by `postgres` user. (Bo Peng)

Because of the security reason, the Pgpool-II default startup user is changed to `postgres` user.

If installing from RPMs, `postgres` user will be allowed to run `if_up/down_cmd` and `arping_cmd` with `sudo` without a password. If `if_up/down_cmd` or `arping_cmd` starts with `"/"`, the setting specified in `if_cmd_path` or `arping_path` will be ignored.

- Down grade LOG to DEBUG5 in sent message module. (Tatsuo Ishii)
-

A.3.3. Major Enhancements

- Allow to use statement level load balancing. (Bo Peng)

This feature enables selecting load balancing node per statement. The current feature for load balancing, the load balancing node is decided at the session start time and will not be changed until the session ends. When set to `statement_level_load_balance = on`, the load balancing node is decided for each read query. For example, in applications that use connection pooling remain connections open to the backend server, because the session may be held for a long time, the load balancing node does not change until the session ends. In such applications, when `statement_level_load_balance` is enabled, it is possible to decide load balancing node per query, not per session.

- Add `auto_failback` (Takuma Hoshiai).

This allows to reattach backend node automatically that is in DOWN status but actually it is running normally.

To use this feature it is required that PostgreSQL is 9.1 or later and new configuration variable `auto_failback` is enabled. Also Pgpool-II must be operating in streaming-replication mode, with `sr_check` and `health_check` are enabled. Pgpool-II calls `pg_stat_replication` on the PostgreSQL primary server to make sure that the standby node in question is running and receiving replication stream from the primary server.

This feature is useful in the case that a standby server fails over due to a temporary network failure.

- Add new `enable_shared_relcache` parameter. (Takuma Hoshiai)

The relation cache were stored in local cache of child processes, so all child processes executed same query to get relation cache. If `enable_shared_relcache` is on, the relation cache is stored in memory cache and all child process share it. It will expect to reduce the load that same query is executed.

- Add new parameter `check_temp_table` to check temporary tables. (Tatsuo Ishii)

Checking temporary tables is slow because it needs to lookup system catalogs. To eliminate the lookup, new method to trace `CREATE TEMP TABLE/DROP TABLE` is added. To use the new method, set `check_temp_table` to `trace`.

Note that it is impossible to trace tables created in functions and triggers. In this case existing method should be used.

- Reduce internal queries against system catalogs. (Tatsuo Ishii)

Currently the relcache module issues 7+ queries to obtain various info from PostgreSQL system catalogs. Some of them are necessary for Pgpool-II to work with multiple version of PostgreSQL. To reduce such internal queries, get PostgreSQL version to know what kind of queries are needed. For

example, we need to know if `pg_namespace` exists and for this purpose we send a query against `pg_class`. But if we know that `pg_namespace` was introduced in PostgreSQL 7.3, we do not need to inquire `pg_class`.

- Performance enhancements for the large INSERT and UPDATE statements. (Muhammad Usama)

`Pgpool-II` only needs very little information, especially for the INSERT and UPDATE statements to decide where it needs to send the query. For example: In master-slave mode, for the INSERT statements `Pgpool-II` only requires the relation name referenced in the statement while it doesn't care much about the column values and other parameters. But since the parser we use in `Pgpool-II` is taken from PostgreSQL source which parses the complete query including the value lists which seems harmless for smaller statements but in case of INSERT and UPDATE with lots of column values and large data in value items, consumes significant time.

So the idea here is to short circuit the INSERT and UPDATE statement parsing as soon as we have the required information. For that purpose, the commit adds the second minimal parser that gets invoked in master-slave mode and tries to extract the performance for large INSERT and UPDATE statements.

Apart from the second parser addition, changes aiming towards the performance enhancements are also part of the commit. See the [commit log](#) for more details.

- Import PostgreSQL 12 beta2 new parser. (Bo Peng)

Major changes of PostgreSQL 12 parser include:

- Add new VACUUM options: `SKIP_LOCKED`, `INDEX_CLEANUP` and `TRUNCATE`.
- Add `COMMIT AND CHAIN` and `ROLLBACK AND CHAIN` commands.
- Add a `WHERE` clause to `COPY FROM`.
- Allow to use `CREATE OR REPLACE AGGREGATE` command.
- Allow to use `mcv` (most-common-value) in `CREATE STATISTICS`.
- `ADD REINDEX` option `CONCURRENTLY`.
- Add `EXPLAIN` option `SETTINGS`.

- Allow to route relcache queries to load balance node. (Tatsuo Ishii)

Queries to build relcache entries were always sent to master (primary) node. This is usually good because we could eliminate the bad effect of replication delay. However if users want to lower the load of master node, it would be nice if we could route the queries to other than master node. This patch introduces new parameter [relcache_query_target](#). If it is set to `load_balance_node`, relcache queries will be routed to load balance node. If it is set to `master`, the queries are routed to master node, which is same as before (this is the default).

- Disable load balance after a SELECT having functions specified in black function list or not specified in white function list. (Bo Peng)

In `Pgpool-II` 4.0 or earlier, if we set [disable_load_balance_on_write](#) = `transaction`, when a write query is issued inside an explicit truncation, subsequent queries should be sent to primary only until the end of this transaction in order to avoid the replication delay. However, the SELECTs having write functions specified in [black_function_list](#) or not specified in [white_function_list](#) are not regarded as a write query and the subsequent read queries are still load balanced. This commit will disable load balance after a SELECT having functions specified in black function list or not specified in white function list.

- Implement new feature to not accept incoming connections. (Tatsuo Ishii)

`Pgpool-II` accepts up to [num_init_children](#) frontends and queues up more connection requests until one of child process becomes free. This mostly works well, but if each session takes long time, the queue grows longer and the whole system does not work smoothly. To overcome the problem, a new way to deal with many connection requests from frontend is implemented: When [reserved_connections](#) is set to 1 or greater, incoming connections from clients are not accepted with error message "Sorry, too many clients already", rather than blocked if the number of current connections from clients is more than $(\text{num_init_children} - \text{reserved_connections})$. This is exactly the same behavior as PostgreSQL.

- Enhance performance by eliminating `select(2)` system calls when they are not necessary. (Tatsuo Ishii,

Jesper Pedersen)

- Enhance performance while sending message to frontend. (Tatsuo Ishii)

SimpleForwardToFrontend(), which is responsible for sending message to frontend, does write buffering only if it is either 'D' (DataRow) or 'd' (CopyData). Other message types were immediately written to socket. But actually this was not necessary. So if the messages are not critical, just write to buffer. With this 10-17% performance enhance was observed.

- Avoid error or notice message analysis if it's not necessary. (Tatsuo Ishii)

After sending query to backend, Pgpool-II always calls pool_extract_error_message() via per_node_error_log(). In the function memory allocation is performed even if error or notice message is returned from backend. To avoid the waste of CPU cycle, check message kind and avoid calling pool_extract_error_message() if it's not error or notice message.

- Enhance performance of CopyData message handling. (Tatsuo Ishii)

When COPY XX FROM STDIN gets executed (typical client is pg_dump), each copy row data is sent from Pgpool-II to frontend using CopyData message. Previously, one CopyData message was followed by a flush, which costed a lot. Instead, now flush is done in subsequent Command Complete, Notice message or Error message. A quick test reveals that this change brings x2.5 speed up.

- Allow to use MD5 hashed password in [health_check_password](#) and [sr_sr_check_password](#). (Tatsuo Ishii)
- Support ECDH key exchange with SSL (Takuma Hoshiai)
- Add backend_application_name to "pgpool show backend" group. (Tatsuo Ishii)
- Deal with PostgreSQL 12. (Tatsuo Ishii)

recovery.conf cannot be used anymore. Standby's recovery configuration is now in postgresql.conf. Also "standby.signal" file is needed in PostgreSQL database cluster directory to start postmaster as a standby server.

HeapTupleGetOid() is not available any more in PostgreSQL 12. Use GETSTRUCT() and refer to oid column of Form_pg_proc.

Change pgpool_adm extension. Now that oid is gone, the signature of CreateTemplateTupleDesc() has been changed.

- Speed up failover when all of backends are down. (Tatsuo Ishii)

Pgpool-II tries to find primary node till [search_primary_node_timeout](#) expires even if all of the backend are in down status. This is not only a waste of time but makes Pgpool-II looked like hanged because while searching primary node failover process is suspended and all of the Pgpool-II child process are in defunct state, thus there's no process which accepts connection requests from clients. Since the default value of searching primary is 300 seconds, typically this keeps on for 300 seconds. This is not comfortable for users.

To fix this immediately give up finding primary node regardless [search_primary_node_timeout](#) and promptly finish the failover process if all of the backend are in down status.

- Resign the master watchdog node from master responsibilities if the primary backend node gets into quarantine state on that. (Muhammad Usama)

By doing this, we could avoid the situation on which there's no primary PostgreSQL server exists. To implement this, make the master/coordinator watchdog node resign from its status if it fails to get the consensus for the quarantined primary node failover, within `FAILOVER_COMMAND_FINISH_TIMEOUT(15)` seconds.

When the watchdog master resigns, because of quarantined primary node its wd_priority is decreased to (-1), so that it should get the least preference in the next election for the master/coordinator node selection. And once the election is concluded the wd_priority for the node gets restored to the original configured value.

In case of failed consensus for standby node failover no action is taken.

- Add parameter [enable_consensus_with_half_votes](#) to configure majority rule calculations. (Muhammad

Usama, Tatsuo Ishii)

Pgpool-II takes the decision of quorum existence and failover consensus after receiving the exact 50% of votes when the watchdog cluster is configured with an even number of nodes. With [enable_consensus_with_half_votes](#) parameter, users can tell Pgpool-II, whether the distributed consensus in an even number of nodes cluster requires $(n/2)$ or $((n/2) + 1)$ votes to decide the majority. Odd number of clusters (3, 5, 7...) are not affected. Extra caution is needed for 2 node watchdog cluster users. See [Section A.3.2](#) for more details.

- Allow to specify absolute path in [pool_passwd](#). (Bo Peng)

Patch is provided by Danylo Hlynskyi.

- Add various sample scripts. (Bo Peng)

Allow failover.sh.sample, follow_master.sh.sample, recovery_1st_stage.sample, recovery_2nd_stage.sample, pgpool_remote_start.sample scripts to be included in distributions.

- Documentation enhancements:
 - Add performance chapter ([Chapter 7](#)). (Tatsuo Ishii)
 - Enhance 'getting started' of 'tutorial' chapter, 'watchdog' of 'tutorial' and some sections of 'server administration'(takuma hoshiai)
 - Update configuration example "Pgpool-II + watchdog setup example". (bo peng)
 - Mention that schema qualifications cannot be used in white/black_function_list. (tatsuo Ishii)
 - Enhance explanation about [failover_command](#) and [follow_master_command](#). (tatsuo ishii)
 - Add note to detach_false_primary configuration parameter. (tatsuo ishii)
 - Add more explanation to follow_master_command. (tatsuo ishii)
 - Enhance watchdog/pgpool-ii example so that it mentions about pg_monitor role. (tatsuo ishii)
 - Mention that multi-statement queries are sent to primary node only. (tatsuo ishii)
 - Add load balancing description. (tatsuo ishii)
 - Add useful link how to create pcp.conf in the pcp reference page. (tatsuo ishii)
 - Add more description to pcp_node_info manual. (tatsuo ishii)
 - Add description to pg_md5 man page how to show pool_passwd ready string. (tatsuo ishii)
 - Enhance client authentication docs. (tatsuo ishii)
 - Enhance watchdog documents regarding quorum failover. (tatsuo ishii)
 - Mention that in raw mode or load_balance_mode = off case for relation cache. (tatsuo ishii)
 - Add general description about failover. (tatsuo ishii)

A.3.4. Bug fixes

- In this release same bug fixes as Pgpool-II 4.0.7 are already applied. See [Section A.6](#) for more details of those fixes.

A.4. Release 4.0.9

Release Date: 2020-05-21

A.4.1. Bug fixes

- Fix build error on some system (Fedora 32). (Tatsuo Ishii)
- Fix pgpool ssl front end accept all ciphers. ([bug 608](#)) (Muhammad Usama)
- Downgrade too verbose authentication logs. (Tatsuo Ishii)
- Fix unnecessary checks. (Tatsuo Ishii)
Patch contributed by sherlockcpp. Discussion: [\[pgpool-general: 7004\]](#)
- Doc: Fix typo. (Tatsuo Ishii)
Patch contributed by Umar Hayat. Discussion: [\[pgpool-hackers: 3587\]](#)
- Fix for segmentation fault in PCP process. ([bug 598](#)) (Muhammad Usama)
- Fix a warning message is never output in watchdog. (Tatsuo Ishii)
Patch provided by sherlockcpp. Discussion: ([\[pgpool-general: 6956\]](#))
- Doc: Fix typo. (Tatsuo Ishii, Bo Peng)
Patch provided by sherlockcpp.
- Fix Pgpool-II hangs when an Execute message is issued right after Sync message and query cache hits. (Tatsuo Ishii)
- Fix problems in watchdog source code processing json data. ([bug 596](#)) (Tatsuo Ishii)
Patch is provided by Greg Nancarrow (Fujitsu Australia).
- Fix SCRAM auth handling bug. ([bug 596](#)) (Tatsuo Ishii)
Patch is provided by Greg Nancarrow (Fujitsu Australia).
- Fix possible data inconsistency in native replication mode. (Tatsuo Ishii)
Discussions: [\[pgpool-general: 6896\]](#) [\[pgpool-hackers: 3540\]](#)
- Fix watchdog ping probes fail with long hostnames due to small buffer. ([bug 516](#)) (Bo Peng)
- Doc: Enhance documents. (Tatsuo Ishii)
- Fix "last status change" timestamp is not set properly. (Tatsuo Ishii)

A.5. Release 4.0.8

Release Date: 2020-02-20

A.5.1. Changes

- Disallowing the quorum aware failover option for the native replication mode. (Muhammad Usama)
In native replication mode, Pgpool-II. is responsible for replicating the data on all backend nodes, and if a node becomes `quarantined` then Pgpool-II stops sending the writes to that node. This is dangerous since it can cause data inconsistency.
So as per the discussion, we reached the conclusion to disallow [failover_when_quorum_exists](#) with the

native replication mode so that backend node should never get into quarantine when Pgpool-II is configured in the replication mode.

- Check if socket file exists at startup and remove them if PID file doesn't exist to avoid bind() failure. (Bo Peng)

A.5.2. Bug fixes

- Fix incorrect query rewriting in native replication mode. ([bug 551](#)) (Bo Peng)
- Doc: Update documentation and fix documentation typos. (Takuma Hoshiai, Tatsuo Ishii, Bo Peng)
- Fix child process segfault after reload if [health_check_database](#) is empty. ([bug 571](#)) (Bo Peng)
- Fix [pgpool_setup](#) to support PostgreSQL 12. (Tatsuo Ishii)
- Doc: Change the sample `follow_master_command` script `%M %H` parameter's order. (Bo Peng)
- Fix occasional regression test failure. (Tatsuo Ishii)
- Fix replication delay worker segfault when `application_name` in `primary_conninfo` is an empty string. ([bug 565](#)) (Tatsuo Ishii)
- Fix [PGPOOL_SHOW](#) command doesn't display `ALWAYS_MASTER`, when `backend_flag` = 'ALWAYS_MASTER'. (Takuma Hoshiai)
- Fix the bug with ignored syslog setting. (Bo Peng)

A.6. Release 4.0.7

Release Date: 2019-10-31

A.6.1. Bug fixes

- Fix incorrect query rewrite in replication mode. (Bo Peng)
- Fix that health check timeout does work in certain case. (Tatsuo Ishii)
Discussion: [\[pgpool-hackers: 3458\]](#), [\[pgpool-hackers: 3459\]](#)
- Doc: add [failover_command](#) description when all standby nodes are down. (Takuma Hoshiai)
- Doc: add note not to assign PostgreSQL servers to [trusted_servers](#). (Tatsuo Ishii)
- Fix for miscellaneous watchdog issues. (Muhammad Usama)

The lost nodes reported by life-check are treated as hard failures even when the node is reachable from the watchdog core.

In the case of network partitioning or partial life-check failure, the kicked out standby node was too aggressive in trying to connect to the master or become a master itself so that potentially put the unnecessary burden on the network and the cluster nodes. Fix is to make the isolated node a bit calm and wait between trying to connect to master or become a master.

See details: ([bug 547](#)), [\[pgpool-general: 6672\]](#)

- Fix assorted ancient v2 protocol bugs. (Tatsuo Ishii)
- Fix problem that [syslog_facility](#) doesn't change by reload. ([bug 548](#)) (Takuma Hoshiai)

- Fix Pgpool-II shutdown failed in certain case. (Tatsuo Ishii)
 - Allow the lost standby node to rejoin the master watchdog node when it gets rediscovered by the lifecheck. ([bug 545](#)) (Muhammad Usama)
 - Overhaul health check debug facility. (Tatsuo Ishii)
 - Fix segfault when executing an erroneous query after DEALLOCATE a named statement. ([bug 546](#)) (Tatsuo Ishii)
 - Doc: clarify that certificate authentication works between only client and Pgpool-II. (Tatsuo Ishii)
 - Doc: update configuration Examples "Pgpool-II + Watchdog Setup Example". (Bo Peng)
 - Doc: mention that VIP will not be brought up if quorum does not exist. (Tatsuo Ishii)
 - Fix `pgpool_setup` to deal with PostgreSQL 9.1. (Tatsuo Ishii)
 - Fix for password authentication does not work when the password for the connecting user is not found in the `pool_passwd` file. ([bug 534](#)) (Muhammad Usama)
 - Add "-l" option to `arping_cmd` command default setting. (Bo Peng)
-

A.7. Release 4.0.6

Release Date: 2019-08-15

A.7.1. Enhancements

- Doc: Update "Pgpool-II + Watchdog Setup Example" to support PostgreSQL 12. (Bo Peng)
 - Import some of memory manager debug facilities from PostgreSQL. (Tatsuo Ishii)
 - Use `pg_get_expr()` instead of `pg_attrdef.adsrc` to support for PostgreSQL 12. (Bo Peng)
 - Enhance shutdown script of `pgpool_setup`. (Tatsuo Ishii)
 - Make `shutdownall` to wait for completion of shutdown of Pgpool-II.
 - If environment variable `CHECK_TIME_WAIT` is set to true, use `netstat` command to confirm usage of the TCP/IP port while executing shutdown script.
 - Deal `pgpool_adm` extension with PostgreSQL 12. (Tatsuo Ishii)
 - Doc: add description to `pg_md5` man page how to show `pool_passwd` hashed string. (Tatsuo Ishii)
 - Doc: add general description about failover. (Tatsuo Ishii)
-

A.7.2. Bug fixes

- Test: Fix test failure of `extended-query-test` when `disable_load_balance_on_write = off/transaction/always`. (Tatsuo Ishii)
- Fix "unable to bind. cannot get parse message" error. ([bug 531](#)) (Tatsuo Ishii)
- Fix online-recovery is blocked after a child process exits abnormally with replication mode and watchdog. ([bug 483](#)) (Muhammad Usama)
- Fix for keep the backend health check running on quarantined nodes. (Muhammad Usama)

Pgpool should keep the backend health check running on quarantined nodes so that when the connectivity resumes, they should automatically get removed from the quarantine.

See [\[pgpool-hackers: 3295\]](#) for more details.

- Fix for no primary on standby pgpool when primary is quarantined on master. (Muhammad Usama)
Master watchdog Pgpool sends the backend status sync message if the primary node is quarantined on it. So standby watchdog pgpool must not update its status when the status of current primary node is not DOWN.
- Fix [watchdog_setup](#) command `mode` option to work correctly. (Takuma Hoshiai)
- Fix [pgpool_setup](#) to produce correct follow master command. (Tatsuo Ishii)
- Fix in native replication mode Pgpool-II rewriting query error when the queries include `GROUPS` and `EXCLUDE` in `frame` clauses. (Bo Peng)
- Fix query cache module so that it checks oid array's bound. (Tatsuo Ishii)
- Fix off-by-one error in query cache module. (Tatsuo Ishii)
- Allow health check process to reload. (Tatsuo Ishii)
- Fix segfault when query cache is enabled. ([bug 525](#)) (Tatsuo Ishii)
- Down grade LOG "checking zapping sent message ..." to DEBUG5. (Tatsuo Ishii)
Discussion: [\[pgpool-general: 6620\]](#)
- Fix segfault when `samenet` is specified in `pool_hba.conf`. (Tatsuo Ishii)
Discussion: [\[pgpool-general: 6601\]](#).
- Doc: Fix documentation mistakes in `follow_master.sh` script and typos. (Bo Peng)
- Fix health check process is not shutting down in certain cases. (Tatsuo Ishii)
- Fix to deal with backslashes according to the config of `standard_conforming_strings` in native replication mode. ([bug 467](#)) (Bo Peng)
- Fix compile error on FreeBSD. ([bug 512](#), [bug 519](#)) (Bo Peng)
- Fix memory leaks. (Tatsuo Ishii)
- Make failover in progress check more aggressively to avoid potential segfault. (Tatsuo Ishii)

A.8. Release 4.0.5

Release Date: 2019-05-16

A.8.1. Enhancements

- Doc: Improve [Reference II, PCP commands](#) document. (Tatsuo Ishii)
- Speed up failover when all of backends are down. (Tatsuo Ishii)

If all of the backend are in down status, immediately give up finding primary node regardless `search_primary_node_timeout` and promptly finish the failover process.

Discussion: [\[pgpool-hackers: 3321\]](#)

- `pgpool-recovery` extension and `pgpool_setup` is now ready for the next major release PostgreSQL 12. (Tatsuo Ishii)
 - Doc: add [restrictions](#) entry. (Takuma Hoshiai)
-

A.8.2. Bug fixes

- Fix the wrong error message "ERROR: connection cache is full", when all backend nodes are down. ([bug 487](#)) (Bo Peng)

When all backend nodes are down, Pgpool-II throws an incorrect error message "ERROR: connection cache is full". Change the error message to "all backend nodes are down, pgpool requires at least one valid node".

- Remove unused `.sgml` file. (Takuma Hoshiai)
- Avoid exit/fork storm of `pool_worker_child` process. (Tatsuo Ishii)

`pool_worker_child` issues query to get WAL position using `do_query()`, which could throw FATAL error. In this case `pool_worker_child` process exits and Pgpool-II parent immediately forks new process. This cycle indefinitely repeats and gives high load to the system. To avoid the exit/fork storm, sleep `sr_check_period`.

- Fix [black_function_list](#)'s broken default value. (Tatsuo Ishii)
- Fix "not enough space in buffer" error. ([bug 499](#)) (Tatsuo Ishii)

The error occurred while processing error message returned from backend and the cause is that the query string in question is too big. Problem is, the buffer is in fixed size (8192 bytes). Eliminate the fixed size buffer and use pallocated buffer instead. This also saves some memory copy work.

- Fix DROP DATABASE failure. (Tatsuo Ishii)
 - Fix wrong variable in `read_status_file()` function. ([bug 493](#)) (Takuma Hoshiai)
 - Add missing `test/watchdog_setup` to EXTRA_DIST. ([bug 470](#)) (Bo Peng)
 - Doc: mention that multi-statement queries are sent to primary node only. ([bug 492](#)) (Tatsuo Ishii)
 - Fix md5 auth broken in raw mode with more than 1 backends. ([bug 491](#)) (Tatsuo Ishii)
 - Test: Fix occasional regression test failure of `014.watchdog_test_quorum_bypass`. (Tatsuo Ishii)
 - Abort session if failover/failback is ongoing to prevent potential segfault. ([bug 481](#), [bug 482](#)) (Tatsuo Ishii)
 - Fix compiler warnings. (Tatsuo Ishii)
 - Fix memory leak in "batch" mode in extended query. ([bug 468](#)) (Tatsuo Ishii)
-

A.9. Release 4.0.4

Release Date: 2019-03-29

A.9.1. Enhancements

- Add new configuration option [ssl_prefer_server_ciphers](#). (Muhammad Usama)

Add the new setting [ssl_prefer_server_ciphers](#) to let users configure if they want client's or server's cipher order to take preference.

The default for this parameter is off, which prioritizes the client's cipher order as usual. However, this is just for keeping backward compatibility, and it is possible that a malicious client uses weak ciphers. For this reason, we recommend to set this parameter to on at all times.

- Allow to set a client cipher list. (Tatsuo Ishii, Yugo Nagata)

For this purpose, a new parameter [ssl_ciphers](#), which specifies the cipher list to be accepted by Pgpool-II, is added. This is already implemented in PostgreSQL and useful to enhance security when SSL is enabled.

A.9.2. Bug fixes

- Fix unnecessary `fsync()` to `pgpool_status` file. (Tatsuo Ishii)

Whenever new connections are created to PostgreSQL backend, `fsync()` was issued to `pgpool_status` file, which could generate excessive I/O in certain conditions. So reduce the chance of issuing `fsync()` so that it is issued only when backend status is changed.

Discussion: [\[pgpool-general: 6436\]](#)

- Doc: add more explanation to [follow_master_command](#). (Tatsuo Ishii)

Add description how [follow_master_command](#) is executed etc.

- Doc: add note to [detach_false_primary](#) configuration parameter. ([bug 469](#)) (Tatsuo Ishii)

To use this feature, [sr_check_user](#) must be super user or in `pg_monitor` group.

A.10. Release 4.0.3

Release Date: 2019-02-21

A.10.1. Changes

- Doc: Update configuration example [Section 8.3](#). (Bo Peng)

A.10.2. Bug fixes

- Skip over "host=" when getting info from `conninfo` string. (Bo Peng)

Patch provided by Nathan Ward.

- Test: Fix old JDBC functions and typos in regression test `068.memqcache_bug`. (Takuma Hoshiai)
- Doc: Fix configuration change timing regarding [memory_cache_enabled](#). (Tatsuo Ishii)
- Fix online recovery failed due to [client_idle_limit_in_recovery](#) in certain cases. ([bug 431](#)) (Tatsuo Ishii)
- Reduce memory usage when large data set is returned from backend. ([bug 462](#)) (Tatsuo Ishii)
- Test: Fix syntax error in extended query test script. (Tatsuo Ishii)
- Fix corner case bug when `strip_quote()` handle an empty query string. ([bug 458](#)) (Tatsuo Ishii)
- Doc: Mention that schema qualifications cannot be used in `white/black_function_list`. (Tatsuo Ishii)
- Fix typo about `wd_priority` in `watchdog_setup`. (Takuma Hoshiai)

- Fixed segfault when `wd_lifecycle_method = 'query'`. ([bug 455](#)) (Muhammad Usama)

The fix was proposed by Muhammad Usama and some adjustments to the patch and testing is done by Yugo Nagata.

- Fix Pgpool child segfault if failover occurs when trying to establish a connection. (Tatsuo Ishii)
- Doc: fix typo in `logdir` description. ([bug 453](#)) (Tatsuo Ishii)
- Fix PAM authentication failed. (Takuma Hoshiai)

See [\[pgpool-general: 6353\]](#) for more details.

- Fix Pgpool-II hang if a client sends a extended query message such as close after sync message but before next simple query. (Tatsuo Ishii)

Discussion: [\[pgpool-hackers: 3164\]](#)

- Fix Pgpool-II hang when `idle_in_transaction_session_timeout = on`. ([bug 448](#)) (Tatsuo Ishii)

A.11. Release 4.0.2

Release Date: 2018-11-22

A.11.1. Bug fixes

- Fix to sort startup packet's parameters sent by client. ([bug 444](#))(Takuma Hoshiai)

If order of startup packet's parameters differ between cached connection pools and connection request, didn't use connection pool and created new connection pool.

- Fix broken authentication for Pgpool-II's internal connections. (Muhammad Usama)

The issue is caused by a mistake in "SCRAM and Certificate authentication support" commit. The problem is while authenticating against backend in `connection_do_auth()`, it returns to caller as soon as backend returns auth ok response. So authentication itself established fine. However `connection_do_auth()` does not proceed until it receives "Ready for query". The fix is to keep processing the data after receiving `auth_ok` response until we get "Ready for query".

Patch provided by Tatsuo Ishii and a tiny modification made by Muhammad Usama.

- Fix compiler warnings with gcc 8.x. (Takuma Hoshiai)
- Fix segmentation fault occurs when a certain Bind message is sent in native replication mode. ([bug 443](#))(Bo Peng)

If the number of parameter format codes is specified to one, but the number of the original query's parameter is zero, `bind_rewrite_timestamp()` will call `memcpy` with a negative value. This causes segmentation fault.

Patch is provided by Yugo Nagata.

- Fix a query passed to relcache so that it uses schema qualified table name. (Tatsuo Ishii)
 - Fix query cache invalidation bug. (Tatsuo Ishii)
 - Fix segfault in extended query + query cache case. (Tatsuo Ishii)
 - Fix memory leak in extended query + query cache enabled. (Tatsuo Ishii)
-

A.12. Release 4.0.1

Release Date: 2018-10-31

A.12.1. Changes

- Allow ssl key files to be owned by users other than Pgpool-II user and root. (Muhammad Usama)
- Allow PCP[attach/detach/promote] commands during failover. (Muhammad Usama)

A.12.2. Bug fixes

- Fix corruption of `pool_passwd` file due to the different lengths of md5, AES and text type passwords. (Muhammad Usama)

The patch was provided by Takuma Hoshiai and modified by Muhammad Usama.

- Fix typo in `child_max_connections` description of `SHOW POOL_STATUS` output. (Tatsuo Ishii)

Patch provided by Phil Ramirez.

- Fix segmentation fault when error query and Sync message are sent in native replication mode. ([bug 434](#)) (Takuma Hoshiai)

In native replication mode, segmentation fault occurs when Sync messages is sent just after a query error.

A.13. Release 4.0

Release Date: 2018-10-19

A.13.1. Overview

This version adds support for SCRAM and CERT authentication, improves load balancing control and import PostgreSQL 11 new SQL parser.

Major enhancements in Pgpool-II 4.0 include:

- Add SCRAM and Certificate authentication support.
- Detecting "false" primary server of PostgreSQL.
- Improvement of load balancing:
 - More load balancing fine control after write queries.
 - Load balancing control for specific queries.
 - Allow to specify load balance weight ratio for load balance parameters.
- Add last state change timestamp to [SHOW POOL NODES](#).
- Import PostgreSQL 11 SQL parser.

- Logging client messages.
-

A.13.2. Migration to Version 4.0

Version 4.0 contains a number of changes that may affect compatibility with previous releases. Observe the following incompatibilities:

- Add 1st/2nd stage online recovery commands parameter to get the node number to be recovered. (Tatsuo Ishii)

Online recovery script now accepts 5 parameters, rather than 4 (the 5th parameter is node number to be recovered). Run `ALTER EXTENSION pgpool_recovery UPDATE TO '1.2'` to update `pgpool_recovery` version. Existing 4-parameter-style recovery scripts can be used if you don't care about information provided by the 5th parameter.

See [recovery_1st_stage_command](#) and [recovery_2nd_stage_command](#) for more details.

- `fail_over_on_backend_error` parameter is renamed to [failover_on_backend_error](#). (Muhammad Usama)

Now we throw a warning message when old config name `fail_over_on_backend_error` is used instead of [failover_on_backend_error](#). Using the old config variable name will have no effect.

- Allow to specify the AES encrypted password in the `pgpool.conf`. (Muhammad Usama)

Since 4.0, you can specify the AES encrypted password in the `pgpool.conf` file for [health_check_password](#), [sr_check_password](#), [wd_lifecycle_password](#) and [recovery_password](#).

To specify the unencrypted clear text password, prefix the password string with `TEXT`. In the absence of a valid prefix, Pgpool-II will consider the string as a plain text password.

The empty password string specified in the `pgpool.conf` file for [health_check_password](#), [sr_check_password](#), [wd_lifecycle_password](#) and [recovery_password](#) will only be used when the [pool_passwd](#) does not contain the password for that specific user. If these parameters are left blank, Pgpool-II will first try to get the password for that specific user from [pool_passwd](#) file before using the empty password.

A.13.3. Major Enhancements

- Add support for `SCRAM` and Certificate based authentication methods. (Muhammad Usama)

- Add support for `SCRAM` authentication method.

`SCRAM` authentication is supported using the `pool_passwd` authentication file.

See [Section 6.2.3](#) for more details.

- Allow to use `CERT` authentication between Pgpool-II and frontend.

To use this authentication method, Pgpool-II will require that the client provide a valid certificate.

See [Section 6.2.4](#) for more details.

- Able to use different auth methods for frontend and backend.

Now it is possible to use different authentication methods between client to Pgpool-II and Pgpool-II to backend.

- Now [pool_passwd](#) can store three format passwords. AES256 encrypted format, plain text format and `md5` format.

Pgpool-II identifies the password format type by its prefix, so each password entry in the `pool_passwd` must be prefixed as per the password format.

`md5` hashed passwords will be prefixed with `md5` and AES256 encrypted password types will be stored using `AES` prefix. To store the password in the plain text format `TEXT` prefix can be used.

In the absence of a valid prefix, Pgpool-II will be considered the string as a plain text password.

For example:

```
username1:AESIFwI86k+ZbVdf6C+t3qpGA==  
username2:md59c6583185ba6a85bdcd1f129ec8cabb4  
username3:TEXTmypassword
```

- Able to use **MD5** and **SCRAM** authentication methods to connect to database without [pool_passwd](#).

A new configuration parameter [allow_clear_text_frontend_auth](#) is added. This parameter enables this config allows the Pgpool-II to use clear-text-password authentication with frontend clients when [pool_passwd](#) file does not contain the password for the connecting user, and use that password (provided by client) to authenticate with the backend using **MD5** and/or **SCRAM** authentication.

- New [pg_enc](#) utility to create encrypted passwords.

A new utility [pg_enc](#) is added to create AES encrypted passwords.

See [Chapter 6](#) for more details.

- Add new parameter [detach_false_primary](#). (Tatsuo Ishii)

If set `detach_false_primary = on`, detach false primary node. The default is off. This parameter is only valid in streaming replication mode and for PostgreSQL 9.6 or after since this feature uses `pg_stat_wal_receiver`. If PostgreSQL 9.5.x or older version is used, no error is raised, just the feature is ignored.

- Add [disable_load_balance_on_write](#) parameter to specify load balance behavior after write queries appear. (Tatsuo Ishii)

This parameter allows to specify the behavior when a write query issued.

- Allow to specify load balance weight ratio for load balance parameters. (Bo Peng)

Add a new feature to allow to specify load balance weight ratio for [database_redirect_preference_list](#) and [app_name_redirect_preference_list](#) parameters.

You can specify the list of "database-name:node id(ratio)" pairs to send SELECT queries to a particular backend node for a particular database connection at a specified load balance ratio.

Also you can specify list of "application-name:node id(ratio)" pairs to send SELECT queries to a particular backend node for a particular client application connection at a specified load balance ratio.

This load balance ratio specifies a value between 0 and 1, and the default is 1.0.

- Add new parameter [black_query_pattern_list](#) to enable specifying SQL patterns lists that should not be load-balanced. (Bo Peng)

Specify a semicolon separated list of SQL patterns that should be sent to primary node only. Regular expression can be used in SQL patterns. Only Master Slave mode is supported.

- Add new parameter [log_client_messages](#) to allow logging client message. (Takuma Hoshiai, Tatsuo Ishii)

Set `log_client_messages = on`, any client messages will be logged without debugging messages.

- Add `last_status_change` column to [SHOW POOL NODES](#) command. (Tatsuo Ishii)

The new column indicates the time when `status` or `role` has been changed.

See [\[pgpool-hackers: 2822\]](#) for the reasoning to add the column.

- Import PostgreSQL 11's SQL parser. (Bo Peng)

Now Pgpool-II can fully understand the newly added SQL syntax in PostgreSQL 11, such as `CREATE/ALTER/DROP PROCEDURE, { RANGE | ROWS | GROUPS } frame_start [frame_exclusion]` etc.

A.13.4. Other Enhancements

- Add "-r" option to `pgpool_setup` to allow use of `pg_rewind`. (Tatsuo Ishii)

With this option, `pgpool_setup` creates `basebackup.sh` which tries `pg_rewind` first. If it fails, falls back to `rsync`.

Also a new environment variable "USE_PG_REWIND" to `pgpool_setup` is added. This brings the same effect as "-r" option is specified.

- Add "-s" option to `pgpool_setup` to support for replication slot. (Tatsuo Ishii)

This eliminates the problem when standby is promoted. When a standby is promoted, it changes the time line in PITR archive, which will stop other standby if any because of shared archive directory.

Also a new environment variable "USE_REPLICATION_SLOT" to `pgpool_setup` is added. This brings the same effect as "-s" option is specified.

If "USE_REPLICATION_SLOT=true", in streaming replication mode, use replication slot instead of archive.

By setting `USE_REPLICATION_SLOT` environment variable, now `pgpool_setup` in all tests uses replication slots. This reduces disk space under `src/test/regression` from 6.3GB to 5,1GB (1.2GB savings).

- Introduce `pgproto` to Pgpool-II. (Takuma Hoshiai)

A new utility `pgproto` is added to test PostgreSQL or any other servers that understand the frontend/backend protocol.

- Allow to display Pgpool-II child process id and PostgreSQL backend id in `pcp_proc_info`. (Tatsuo Ishii)

Add `--all` option to display all child processes and their available connection slots.

- Add `replication_delay` and `last_status_change` to `pcp_node_info`. (Tatsuo Ishii)

- Add `role`, `replication_delay` and `last_status_change` columns to `pgpool_adm`'s `pgpool_adm_pcp_node_info`. (Tatsuo Ishii)

A.13.5. Changes

- Downgrade most of DEBUG1 messages to DEBUG5. (Tatsuo Ishii)

This significantly reduces the size of `pgpool` log when `pgpool` starts with `-d` option (this is equivalent to setting `client_min_messages` to `debug1`).

Per discussion [[pgpool-hackers: 2794](#)].

A.13.6. Bug fixes

- Fix syntax error in native replication, when queries including time functions (`now()`, etc.) and `IN (SELECT ...)` in `WHERE` clause. ([bug 433](#)) (Bo Peng)
- Fix compiler error if `HAVE_ASPRINTF` is not defined. (Tatsuo Ishii)
- Fix `configure.ac` to remove generating `src/sql/pgpool_adm/Makefile.in`. (Tatsuo Ishii)
- Fix `pgpool` main process segfault when PostgreSQL 9.5 is used. (Tatsuo Ishii)

`pgpool_setup -n 3` (or greater) triggers the bug. While recovering node 2, `pgpool` main process tried to retrieve version info from backend #2 even if it's not running. This causes the segfault because connection was not established yet. The reason why PostgreSQL 9.6 or later was not suffered from the bug was, PostgreSQL exited the loop as soon as the server version is higher than 9.5. To fix this, call to `VALID_BACKEND` macro was added.

- Add missing `health_check_timeout` in `pgpool_setup`. (Tatsuo Ishii)

Per node `health_check_timeout` was missing and this should had been there since the per node health check parameter support was added.

- Test: Try to reduce the chance of regression 006.memcache failure. (Tatsuo Ishii)

It seems the occasional failure of the test is caused by replication lag. The script tries to read tables from standby but it returns a table not existing error. So insert `pg_sleep()` after creation of tables.

- Test: Fix regression test `055.backend_all_down` error. (Bo Peng)
- Doc: Enhance online recovery document. (Tatsuo Ishii)

Clarify that 2nd stage command is only required in native replication mode.

- Test: Add new regression test `017.node_0_is_down` for node 0 not being primary. (Tatsuo Ishii)
-

A.14. Release 3.7.14

Release Date: 2020-05-21

A.14.1. Bug fixes

- Fix build error on some system (Fedora 32). (Tatsuo Ishii)
 - Fix unnecessary checks. (Tatsuo Ishii)
Patch contributed by `sherlockcpp`. Discussion: [\[pgpool-general: 7004\]](#)
 - Doc: Fix typo. (Tatsuo Ishii)
Patch contributed by Umar Hayat. Discussion: [\[pgpool-hackers: 35871\]](#)
 - Fix for segmentation fault in PCP process. ([bug 598](#)) (Muhammad Usama)
 - Doc: Fix typo. (Tatsuo Ishii, Bo Peng)
Patch provided by `sherlockcpp`.
 - Fix `Pgpool-II` hangs when an Execute message is issued right after Sync message and query cache hits. (Tatsuo Ishii)
 - Fix problems in watchdog source code processing json data. ([bug 596](#)) (Tatsuo Ishii)
Patch is provided by Greg Nancarrow (Fujitsu Australia).
 - Fix possible data inconsistency in native replication mode. (Tatsuo Ishii)
Discussions: [\[pgpool-general: 6896\]](#) [\[pgpool-hackers: 3540\]](#)
 - Fix watchdog ping probes fail with long hostnames due to small buffer. ([bug 516](#)) (Bo Peng)
-

A.15. Release 3.7.13

Release Date: 2020-02-20

A.15.1. Changes

- Disallowing the quorum aware failover option for the native replication mode. (Muhammad Usama)

In native replication mode, Pgpool-II is responsible for replicating the data on all backend nodes, and if a node becomes `quarantined` then Pgpool-II stops sending the writes to that node. This is dangerous since it can cause data inconsistency.

So as per the discussion, we reached the conclusion to disallow `failover_when_quorum_exists` with the native replication mode so that backend node should never get into quarantine when Pgpool-II is configured in the replication mode.

- Check if socket file exists at startup and remove them if PID file doesn't exist to avoid `bind()` failure. (Bo Peng)

A.15.2. Bug fixes

- Fix incorrect query rewriting in native replication mode. ([bug 551](#)) (Bo Peng)
- Doc: Update documentation and fix documentation typos. (Takuma Hoshiai, Tatsuo Ishii, Bo Peng)
- Fix child process segfault after reload if `health_check_database` is empty. ([bug 571](#)) (Bo Peng)
- Fix `pgpool_setup` to support PostgreSQL 12. (Tatsuo Ishii)
- Fix occasional regression test failure. (Tatsuo Ishii)
- Fix replication delay worker segfault when `application_name` in `primary_conninfo` is an empty string. ([bug 565](#)) (Tatsuo Ishii)
- Fix `PGPOOL_SHOW` command doesn't display `ALWAYS_MASTER`, when `backend_flag` = 'ALWAYS_MASTER'. (Takuma Hoshiai)
- Fix the bug with ignored syslog setting. (Bo Peng)

A.16. Release 3.7.12

Release Date: 2019-10-31

A.16.1. Bug fixes

- Fix incorrect query rewrite in replication mode. (Bo Peng)
- Fix that health check timeout does work in certain case. (Tatsuo Ishii)
Discussion: [\[pgpool-hackers: 3458\]](#), [\[pgpool-hackers: 3459\]](#)
- Doc: add `failover_command` description when all standby nodes are down. (Takuma Hoshiai)
- Doc: add note not to assign PostgreSQL servers to `trusted_servers`. (Tatsuo Ishii)
- Fix for miscellaneous watchdog issues. (Muhammad Usama)

The lost nodes reported by life-check are treated as hard failures even when the node is reachable from the watchdog core.

In the case of network partitioning or partial life-check failure, the kicked out standby node was too aggressive in trying to connect to the master or become a master itself so that potentially put the unnecessary burden on the network and the cluster nodes. Fix is to make the isolated node a bit calm and wait between trying to connect to master or become a master.

See details: ([bug 547](#)), [\[pgpool-general: 6672\]](#)

- Fix assorted ancient v2 protocol bugs. (Tatsuo Ishii)

- Fix problem that [syslog_facility](#) doesn't change by reload. ([bug 548](#)) (Takuma Hoshiai)
 - Fix Pgpool-II shutdown failed in certain case. (Tatsuo Ishii)
 - Allow the lost standby node to rejoin the master watchdog node when it gets rediscovered by the lifecheck. ([bug 545](#)) (Muhammad Usama)
 - Overhaul health check debug facility. (Tatsuo Ishii)
 - Fix segfault when executing an erroneous query after DEALLOCATE a named statement. ([bug 546](#)) (Tatsuo Ishii)
 - Doc: mention that VIP will not be brought up if quorum does not exist. (Tatsuo Ishii)
 - Add "-l" option to [arping_cmd](#) command default setting. (Bo Peng)
-

A.17. Release 3.7.11

Release Date: 2019-08-15

A.17.1. Enhancements

- Import some of memory manager debug facilities from PostgreSQL. (Tatsuo Ishii)
 - Use `pg_get_expr()` instead of `pg_attrdef.adsrc` to support for PostgreSQL 12. (Bo Peng)
 - Enhance shutdown script of [pgpool_setup](#). (Tatsuo Ishii)
 - Make shutdownall to wait for completion of shutdown of Pgpool-II.
 - If environment variable `CHECK_TIME_WAIT` is set to true, use `netstat` command to confirm usage of the TCP/IP port while executing shutdown script.
 - Doc: add description to [pg_md5](#) man page how to show `pool_passwd` hashed string. (Tatsuo Ishii)
 - Doc: add general description about failover. (Tatsuo Ishii)
 - Deal `pgpool_adm` extension with PostgreSQL 12. (Tatsuo Ishii)
-

A.17.2. Bug fixes

- Fix "unable to bind. cannot get parse message" error. ([bug 531](#)) (Tatsuo Ishii)
- Fix online-recovery is blocked after a child process exits abnormally with replication mode and watchdog. ([bug 483](#)) (Muhammad Usama)
- Fix for keep the backend health check running on quarantined nodes. (Muhammad Usama)

Pgpool should keep the backend health check running on quarantined nodes so that when the connectivity resumes, they should automatically get removed from the quarantine.

See [\[pgpool-hackers: 3295\]](#) for more details.

- Fix for no primary on standby pgpool when primary is quarantined on master. (Muhammad Usama)

Master watchdog Pgpool sends the backend status sync message if the primary node is quarantined on it. So standby watchdog pgpool must not update its status when the status of current primary node is not DOWN.
- Fix [watchdog_setup](#) command `mode` option to work correctly. (Takuma Hoshiai)

- Fix [pgpool_setup](#) to produce correct follow master command. (Tatsuo Ishii)
 - Fix query cache module so that it checks oid array's bound. (Tatsuo Ishii)
 - Fix off-by-one error in query cache module. (Tatsuo Ishii)
 - Allow health check process to reload. (Tatsuo Ishii)
 - Fix segfault when query cache is enabled. ([bug 525](#)) (Tatsuo Ishii)
 - Down grade LOG "checking zapping sent message ..." to DEBUG5. (Tatsuo Ishii)
Discussion: [\[pgpool-general: 6620\]](#)
 - Fix segfault when `samenet` is specified in `pool_hba.conf`. (Tatsuo Ishii)
Discussion: [\[pgpool-general: 6601\]](#).
 - Doc: Fix documentation mistakes in `follow_master.sh` script and typos. (Bo Peng)
 - Fix health check process is not shutting down in certain cases. (Tatsuo Ishii)
 - Fix to deal with backslashes according to the config of `standard_conforming_strings` in native replication mode. ([bug 467](#)) (Bo Peng)
 - Fix compile error on FreeBSD. ([bug 512](#), [bug 519](#)) (Bo Peng)
 - Fix memory leaks. (Tatsuo Ishii)
 - Make failover in progress check more aggressively to avoid potential segfault. (Tatsuo Ishii)
-

A.18. Release 3.7.10

Release Date: 2019-05-16

A.18.1. Enhancements

- Doc: Improve [Reference II, PCP commands](#) document. (Tatsuo Ishii)
 - Speed up failover when all of backends are down. (Tatsuo Ishii)
If all of the backend are in down status, immediately give up finding primary node regardless `search_primary_node_timeout` and promptly finish the failover process.
Discussion: [\[pgpool-hackers: 3321\]](#)
 - `pgpool-recovery` extension and `pgpool_setup` is now ready for the next major release PostgreSQL 12. (Tatsuo Ishii)
 - Doc: add [restrictions](#) entry. (Takuma Hoshiai)
-

A.18.2. Bug fixes

- Fix the wrong error message "ERROR: connection cache is full", when all backend nodes are down. ([bug 487](#)) (Bo Peng)

When all backend nodes are down, `Pgpool-II` throws an incorrect error message "ERROR: connection cache is full". Change the error message to "all backend nodes are down, `pgpool` requires at least one valid node".

- Remove unused .sgml file. (Takuma Hoshiai)
- Avoid exit/fork storm of pool_worker_child process. (Tatsuo Ishii)

pool_worker_child issues query to get WAL position using do_query(), which could throw FATAL error. In this case pool_worker_child process exits and Pgpool-II parent immediately forks new process. This cycle indefinitely repeats and gives high load to the system. To avoid the exit/fork storm, sleep sr_check_period.

- Fix [black_function_list](#)'s broken default value. (Tatsuo Ishii)
- Fix "not enough space in buffer" error. ([bug 499](#)) (Tatsuo Ishii)

The error occurred while processing error message returned from backend and the cause is that the query string in question is too big. Problem is, the buffer is in fixed size (8192 bytes). Eliminate the fixed size buffer and use palloced buffer instead. This also saves some memory copy work.

- Fix DROP DATABASE failure. (Tatsuo Ishii)
- Fix wrong variable in read_status_file() function. ([bug 493](#)) (Takuma Hoshiai)
- Add missing test/watchdog_setup to EXTRA_DIST. ([bug 470](#)) (Bo Peng)
- Doc: mention that multi-statement queries are sent to primary node only. ([bug 492](#)) (Tatsuo Ishii)
- Test: Fix occasional regression test failure of 014.watchdog_test_quorum_bypass. (Tatsuo Ishii)
- Abort session if failover/failback is ongoing to prevent potential segfault. ([bug 481](#), [bug 482](#)) (Tatsuo Ishii)
- Fix compiler warnings. (Tatsuo Ishii)
- Fix memory leak in "batch" mode in extended query. ([bug 468](#)) (Tatsuo Ishii)

A.19. Release 3.7.9

Release Date: 2019-03-29

A.19.1. Enhancements

- Add new configuration option [ssl_prefer_server_ciphers](#). (Muhammad Usama)

Add the new setting [ssl_prefer_server_ciphers](#) to let users configure if they want client's or server's cipher order to take preference.

The default for this parameter is off, which prioritizes the client's cipher order as usual. However this is just for keeping backward compatibility, and it is possible that a malicious client uses weak ciphers. For this reason we recommend to set this parameter to on at all times.

- Allow to set a client cipher list. (Tatsuo Ishii, Yugo Nagata)

For this purpose new parameter [ssl_ciphers](#), which specifies the cipher list to be accepted by Pgpool-II, is added. This is already implemented in PostgreSQL and useful to enhance security when SSL is enabled.

A.19.2. Bug fixes

- Fix unnecessary fsync() to pgpool_status file. (Tatsuo Ishii)

Whenever new connections are created to PostgreSQL backend, fsync() was issued to pgpool_status file, which could generate excessive I/O in certain conditions. So reduce the chance of issuing fsync() so that

it is issued only when backend status is changed.

Discussion: [\[pgpool-general: 6436\]](#)

A.20. Release 3.7.8

Release Date: 2019-02-21

A.20.1. Bug fixes

- Test: Fix old JDBC functions and typos in regression test 068.memqcache_bug. (Takuma Hoshiai)
- Doc: Fix configuration change timing regarding [memory_cache_enabled](#). (Tatsuo Ishii)
- Fix online recovery failed due to [client_idle_limit_in_recovery](#) in certain cases. ([bug 431](#)) (Tatsuo Ishii)
- Reduce memory usage when large data set is returned from backend. ([bug 462](#)) (Tatsuo Ishii)
- Test: Fix syntax error in extended query test script. (Tatsuo Ishii)
- Fix corner case bug when `strip_quote()` handle a empty query string. ([bug 458](#)) (Tatsuo Ishii)
- Doc: Mention that schema qualifications cannot be used in `white/black_function_list`. (Tatsuo Ishii)
- Fix typo about `wd_priority` in `watchdog_setup`. (Takuma Hoshiai)
- Fix Pgpool child segfault if failover occurs when trying to establish a connection. (Tatsuo Ishii)
- Doc: fix typo in [logdir](#) description. ([bug 453](#)) (Tatsuo Ishii)
- Fix Pgpool-II hang if a client sends a extended query message such as close after sync message but before next simple query. (Tatsuo Ishii)

Discussion: [\[pgpool-hackers: 3164\]](#)

- Fix Pgpool-II hang when `idle_in_transaction_session_timeout = on`. ([bug 448](#)) (Tatsuo Ishii)
 - Doc: Fix Japanese document typo in [pcp_common_options](#). (Bo Peng)
-

A.21. Release 3.7.7

Release Date: 2018-11-22

A.21.1. Bug fixes

- Fix to sort startup packet's parameters sent by client. ([bug 444](#))(Takuma Hoshiai)
If order of startup packet's parameters differ between cached connection pools and connection request, didn't use connection pool ,and created new connection pool.
- Fix segmentation fault occurs when a certain Bind message is sent in native replication mode. ([bug 443](#))(Bo Peng)

If the number of parameter format codes is specified to one, but the number of the original query's parameter is zero, `bind_rewrite_timestamp()` will call `memcpy` with a negative value. This causes segmentation fault.

Patch is provided by Yugo Nagata.

- Fix a query passed to `relcache` so that it uses schema qualified table name. (Tatsuo Ishii)
 - Fix query cache invalidation bug. (Tatsuo Ishii)
 - Fix segfault in extended query + query cache case. (Tatsuo Ishii)
 - Fix memory leak in extended query + query cache enabled. (Tatsuo Ishii)
-

A.22. Release 3.7.6

Release Date: 2018-10-31

A.22.1. Changes

- Allow `PCP[attach/detach/promote]` commands during failover. (Muhammad Usama)
- Change `pgpool.spec` file to install extension to DB server which supports LLVM JIT. (Bo Peng)
- Doc: Add note to online recovery doc. (Tatsuo Ishii)

This warns that `client_idle_limit_in_recovery` must be smaller than `recovery_timeout`.

- Test: Add regression test for SSL connection. (Tatsuo Ishii)
- Doc: Add notes regarding failover script. (Tatsuo Ishii)

It's not recommended to access against `Pgpool-II` itself from failover/failback scripts.

- Doc: Improve `pg_md5` docs and error message. (Bo Peng)

Patch provided by Jesper Pedersen and modified by me.

- Test: Add definition of `PGLIB` in `regress.sh`. (Bo Peng)

Patch provided by Jesper Pedersen.

A.22.2. Bug fixes

- Fix typo in `child_max_connections` description of `SHOW POOL_STATUS` output. (Tatsuo Ishii)

Patch provided by Phil Ramirez.

- Fix segmentation fault when error query and Sync message are sent in native replication mode. ([bug 434](#)) (Takuma Hoshiai)

In native replication mode, segmentation fault occurs when Sync messages is sent just after a query error.

- Fix syntax error when queries including time functions and `IN (SELECT ...)` in `WHERE` clause in native replication mode. ([bug 433](#)) (Bo Peng)

In native replication mode, queries including time functions (e.g. `now()`, `CURRENT_TIMESTAMP` etc.) are rewritten to a timestamp constant value. However, `Pgpool-II` doesn't support queries including time functions and `IN (SELECT ...)` in `WHERE` clause.

- Fix occasional less data returned to frontend with extended protocol. ([bug432](#)) (Tatsuo Ishii)

The idea for fix is, use pending message data list. It records messages from frontend, and it is expected that we will receive same number of messages.

Initial patch is created by Yugo Nagata and fixed by Tatsuo Ishii.

- Fix memory leak when query cache enabled in streaming replication mode + extended query case. (Tatsuo Ishii)

- Fix memory leak in `trigger_failover_command()`. (Tatsuo Ishii)

- Fix memory leak when `memory_cache_enabled = on` and write SQLs are sent. (Bo Peng)

In a explicit transaction, the `SELECT` results are cached in temporary buffer. If a write SQL is sent which modifies the table, the temporary buffer should be reset.

- Test: Fix occasional failure in regression 065.bug152. (Tatsuo Ishii)

- Test: Add `EXECUTE/DEALLOCATE` regression test. (Takuma Hoshiai)

- Add missing `pgpool_recovery--1.0--1.1.sql` file to update `pgpool_recovery()` function version to 1.1. (Bo Peng)

- Fix kind mismatch error when `DEALLOCATE` statement is issued. (Bo Peng)

`PREPARE` should be add to `pool_add_sent_message`, so that `EXECUTE` and `DEALLOCATE` can be sent to the same node as `PREPARE`.

See [\[pgpool-general: 6226\]](#) for more details.

- Do not update `pool_passwd` if the password length is incorrect. ([bug 419](#)) (Takuma Hoshiai, Tatsuo Ishii)

For Pgpool-II 3.7 or before, the password stored in `pool_passwd` is MD5 password only. So check the correctness of `pool_passwd` by scanning entire file.

- Doc: Change [follow_master_command](#) description "new master" to "new primary". (Bo Peng)

- Fix newer version of gcc warnings. (Tatsuo Ishii)

- Test: Update `clean.sh` which clean up regression test results. (Bo Peng)

Patch provided by Jesper Pedersen.

- Add `.gitignore` files. (Bo Peng)

Patch provided by Jesper Pedersen.

- Fix segfault when node 0 is in down status in case of both health check and [failover_on_backend_error](#) are disabled. (Tatsuo Ishii)

- Doc: Fix typos in documents and scripts. (Tatsuo Ishii)

Patch contributed by Jesper Pedersen.

- Doc: Fix document mistakes of [recovery_1st_stage_command](#) and [recovery_2nd_stage_command](#). (Bo Peng)

A.23. Release 3.7.5

Release Date: 2018-07-31

A.23.1. Bug fixes

- Allow not to use `pool_passwd` in raw mode. ([bug 411](#)) (Tatsuo Ishii)

Since in raw there's only 1 backend is actually involved, there's no need to use `pool_passwd` with `md5` authentication.

- Fix "write on backend 0 failed with error : "Success"" error. ([bug 403](#)) (Tatsuo Ishii)

Don't treated it as an error if `write()` returns 0.

- Fix for 0000409: worker process is not restarted after failover on standby Pgpool-II. ([bug 409](#)) (Muhammad Usama)

Patch contributed by Yugo Nagata.

- Fix for 0000406: failover called with wrong old-primary. ([bug 406](#)) (Muhammad Usama)
- Fixed that the health check process was not started after failed back. ([bug 407](#)) (Tatsuo Ishii)
- Fix memory leaks related to `pool_extract_error_message()`. (Tatsuo Ishii)
- Fix an incorrect declare as `bool`, rather than `int` in `pool_extract_error_message()`. (Tatsuo Ishii)

This led to a segfault issue mentioned on certain platform.

- Fix segfault in `per_node_error_log()` on `armhf` architecture. (Tatsuo Ishii)

Patch provided by Christian Ehrhardt.

- Fix for wrong backend roles on standby after the failover. (Muhammad Usama)
- Doc: Improve documents of "MD5 Password Authentication", "Installing Pgpool-II" and "pg_md5". (Bo Peng)

Patch provided by Takuma Hoshiai.

- Test: Fix 006.memqcache test failure. (Tatsuo Ishii)

A.24. Release 3.7.4

Release Date: 2018-06-12

A.24.1. Bug fixes

- Fix Pgpool-II hung if replication delay is too much, when query cache enabled in extended query mode. (Tatsuo Ishii)

See [\[pgpool-general-jp: 1534\]](#) for more details.

- Doc: Fix document typo of PCP commands option "-U". (Bo Peng)
- Delete some debug code. (Bo Peng)
- In extended query mode, do not set writing tx flag with `SET TRANSACTION READ ONLY`. (Tatsuo Ishii)
- Fix wrong parameter `%P` (old primary node id) passed to failover script in 3.7.3 and 3.7.2. (Tatsuo Ishii)
- Doc: Clarify that [failover_require_consensus](#) requires that health check is enabled. (Tatsuo Ishii)
- Doc: Update outdated [pcp_proc_info](#) manual. (Tatsuo Ishii)
- Test: Fix `test.sh` in `extended_query_test`. (Tatsuo Ishii)

- Add missing `health_check_timeout` in `pgpool_setup`. (Tatsuo Ishii)
- Doc: Enhance online recovery document to Clarify that [recovery_2nd_stage_command](#) is only required in native replication mode. (Tatsuo Ishii)
- Prevent [pcp_recovery_node](#) from recovering "unused" status node. (Tatsuo Ishii)

This allowed to try to recovery a node without configuration data, which leads to variety of problems.

See [\[pgpool-general: 5963\]](#) for more details.

Also I fixed `pgpool_recovery` function so that it quotes an empty string argument with double quotes.

A.25. Release 3.7.3

Release Date: 2018-04-17

A.25.1. Bug fixes

- Disable health check per node parameters by default. (Bo Peng)
- Fix [pcp_detach_node](#) hung when `-g` option is specified. ([bug 391](#)) (Tatsuo Ishii)
- Test: Add new regression test for node 0 is down. (Tatsuo Ishii)
- Make calls to `to_regclass` fully schema qualified. (Tatsuo Ishii)
- Fix `pgpool` child process segfault when `ALWAYS_MASTER` is on. (Tatsuo Ishii)

If following conditions are all met `pgpool` child segfaults:

1. Streaming replication mode.
 2. `fail_over_on_backend_error` is off.
 3. `ALWAYS_MASTER` flag is set to the master (writer) node.
 4. `pgpool_status` file indicates that the node mentioned in #3 is in down status.
- Doc: Improve watchdog documents. (Tatsuo Ishii)
 - Doc: Add a document for adding new config parameter. (Tatsuo Ishii)
 - Test: Improve test script 003.failover. (Bo Peng)
 - Deal with "unable to bind D cannot get parse message "S1" error. (Tatsuo Ishii)
 - Doc: Mention that users can avoid failover using `backend_flag` even PostgreSQL admin shutdown. (Tatsuo Ishii)
 - Doc: Fix document typos. (Bo Peng)
 - Test: Add new regression test for node 0 not being primary. (Tatsuo Ishii)
 - Fix `pgpool_setup` failure in replication mode. (Tatsuo Ishii)
 - Allow to support `pgpool_switch_xlog` PostgreSQL 10. (Tatsuo Ishii)
 - Revert "Fix `pgpool` child process segfault when `ALWAYS_MASTER` is on." (Tatsuo Ishii)

With the [commit](#), write queries are always sent to node 0 even if the primary node is not 0 because `PRIMARY_NODE_ID` macro returns `REAL_MASTER_NODE_ID`, which is usually 0. Thus write queries are

failed with: ERROR: cannot execute INSERT in a read-only transaction

- Test: Enhance extended query test. (Tatsuo Ishii)
 - Doc: Fix [pgpool_admin](#) family functions examples. (Tatsuo Ishii)
-

A.26. Release 3.7.2

Release Date: 2018-02-13

Note: This release fixed the bug with socket writing added in Pgpool-II 3.7.0, 3.6.6 and 3.5.10. Due to this bug, when the network load is high, an illegal message may be sent to the frontend or backend. All users using 3.7.x, 3.6.6 or later, 3.5.10 or later versions of Pgpool-II should update as soon as possible.

A.26.1. Changes

- Allow to build with `libressl`. (Tatsuo Ishii)
See [\[pgpool-hackers: 2714\]](#) for more details. Patch by Sandino Araico Sanchez.
 - Set `TCP_NODELAY` and non blocking to frontend socket. (Tatsuo Ishii)
`TCP_NODELAY` is employed by PostgreSQL, so do we it.
 - Change systemd service file to use `STOP_OPTS="-m fast"`. (Bo Peng)
 - Change `pgpool_setup` to add `restore_command` in `recovery.conf`. (Bo Peng)
-

A.26.2. Bug fixes

- Fix writing transaction flag is accidentally set at commit or rollback. (Tatsuo Ishii)
- Throw a warning message when failover consensus settings on watchdog nodes differs. (Muhammad Usama)
- Doc: Fix document typo. (Bo Peng)
- Fix bug with socket writing. (Tatsuo Ishii)

`pool_write_flush()` is responsible for writing to sockets when pgpool's write buffer is full (this function was introduced in 3.6.6 etc). When network write buffer in kernel is full, it does retrying but it forgot to update the internal buffer pointer. As a result, broken data is written to the socket. This results in variety of problems including too large message length.

- Fix pgpool child process segfault when `ALWAYS_MASTER` is on. (Tatsuo Ishii)

If following conditions are all met pgpool child segfaults:

1. Streaming replication mode.
2. `fail_over_on_backend_error` is off.
3. `ALWAYS_MASTER` flags is set to the master (writer) node.
4. `pgpool_status` file indicates that the node mentioned in #3 is in down status.

See [\[pgpool-hackers: 2687\]](#) and [\[pgpool-general: 5881\]](#) for more details.

- Fix segfault when %a is in `log_line_prefix` and debug message is on. ([bug 376](#)) (Tatsuo Ishii)
- Fix per node health check parameters types. (Tatsuo Ishii)
- Fix queries hanging in `parse_before_bind` with extended protocol and replication + load-balancing. ([bug 377](#)) (Tatsuo Ishii)

A.27. Release 3.7.1

Release Date: 2018-01-09

A.27.1. Bug fixes

- Improve Makefiles. (Bo Peng)

Patch provided by Tomoaki Sato.

- Doc: Fix document typo and mistakes. (Bo Peng)
- Replace `/bin/ed` with `/bin/sed` in [pgpool_setup](#), because `/bin/sed` is included in most distribution's base packages. (Tatsuo Ishii)
- Change the `pgpool.service` and `sysconfig` files to output Pgpool-II log. (Bo Peng)

Removing "Type=forking" and add `OPTS=" -n"` to run Pgpool-II with non-daemon mode, because we need to redirect logs. Using "journalctl" command to see Pgpool-II systemd log.

- Add documentation "[Compiling and installing documents](#)" for SGML document build. (Tatsuo Ishii)
- Fix per node health check parameters ignored. ([bug 371](#)) (Tatsuo Ishii)

Also [pgpool_setup](#) is modified to add appropriate per node health check parameters to `pgpool.conf`.

- Fix health checking process death and forking forever. (Tatsuo Ishii)

When failed to read from backend socket (this could happen when wrong `health_check_user` is specified), the health check process raises a FATAL error in `pool_read()`, which causes death of health check process. And Pgpool-II main forks off a new health check process. This repeats forever.

- Fix timestamp data inconsistency by replication mode. (Bo Peng)

From PostgreSQL10 the column default value such as 'CURRENT_DATE' changes, Pgpool-II didn't rewrite timestamp by the added default values. This caused data inconsistency.

- Doc: Fix [watchdog_setup](#) doc. (Tatsuo Ishii)

It lacked to mention that it supports logical replication mode.

- Downgrade a log message to debug message. (Tatsuo Ishii)

That was mistaken left while last development cycle.

- Test: Add test data for [bug 370](#). (Tatsuo Ishii)

- Fix for re-sync logic in reading packet from backend. (Tatsuo Ishii)

`read_kind_from_backend()`, which reads message kind from backend, re-syncs backend nodes when a ready for query message is received. Unfortunately it forgot to call `pool_pending_message_pull_out()` to delete sync pending message. This leads to random stuck while reading packets from backend. Fix this to call

pool_pending_message_pull_out().

- Fix Pgpool-II hangs. ([bug 370](#)) (Tatsuo Ishii)

If an erroneous query is sent to primary and without a sync message the next query that requires a catalog cache look up is send, Pgpool-II hangs in do_query().

- Fix returning transaction state when "ready for query" message received. (Tatsuo Ishii)

We return primary or master node state of ready for query message to frontend. In most cases this is good. However if other than primary node or master node returns an error state (this could happen if load balance node is other than primary or master node and the query is an erroneous SELECT), this should be returned to frontend, because the frontend already received an error.

- Test: Fix bug with extended-query-test test driver. (Tatsuo Ishii)
- Doc: Enhance document "[Running mode of Pgpool-II](#)". (Tatsuo Ishii)

A.28. Release 3.7

Release Date: 2017-11-22

A.28.1. Overview

This version improves reliability of failover by using new watchdog feature and per node health check. Also this version adapts to changes in PostgreSQL 10: new SQL parser, logical replication and some admin functions name changes.

Major enhancements in Pgpool-II 3.7 include:

- Quorum aware failover feature.
- Allow specifying the hostnames in pool_hba.
- Allow to specify per node health check parameters.
- Support AWS Aurora.
- Import PostgreSQL 10 SQL parser
- Support logical replication.

A.28.2. Major Enhancements

- Quorum and Consensus for backend failover. (Muhammad Usama)

Add ability in the Pgpool-II to considers the existence of quorum and seek the majority node (Pgpool-II nodes part of the watchdog cluster) consensus to validate the backend node failover request. This feature helps make failover decision better and prevent split brain scenarios.

The addition of this feature also made some modification in the execution behavior of the failover (failover, failback, promote-node) command.

Now only the Master node performs the failover, and the failover locks are removed.

Three new configuration parameters to configure the failover behavior from user side: failover_when_quorum_exists, failover_require_consensus, enable_multiple_failover_requests_from_node.

- Allow specifying the hostnames in pool_hba. (Muhammad Usama)

The commit adds the support of hostnames to be used in the address field of pool_hba records, previously only CIDR address was supported.

Along with allowing the hostnames in address field of the HBA record the commit also made the following enhancements in the area.

(1) pool_hba records are now completely parsed at the loading time and we now keep the structured data of records instead of raw record lines, This saves the parsing at every new connection time and however little it may be but its a performance enhancement.

(2) Enhanced parsing now gives the better descriptive error/log messages.

(3) Better handling of auth-options field.

- Supporting per node health check parameters. (Tatsuo Ishii, Muhammad Usama)

Previous implementation of health check is a single serial processing for all of database nodes.

Now pgpool main process forks health check process for each DB node. This commit enables all health-check related parameter to be configured for each individual backend nodes.

For example if we have 3 backend nodes and do following configurations.

```
health_check_period = 10
health_check_period0 = 5
```

Then will set the health_check_period for node 0 to 5 while node-id 1 and 2 will get the value 10

- Import PostgreSQL 10 SQL parser. (Bo Peng)
- Support AWS Aurora. (Tatsuo Ishii)

Add new backend flag "ALWAYS_MASTER" to control the primary node detecting logic. Since we cannot use pg_is_in_recovery() in Aurora, we assign the new flag to a backend which is specified as "writer" in Aurora. Since Aurora always use the same hostname for the master (it's called "writer"), find_primary_node() just returns the node id which has ALWAYS_MASTER flag on.

See more details about ALWAYS_MASTER flag [Table 5-3](#).

Other than that, user can use the streaming replication mode with Aurora. Notice that replication delay cannot be performed in Aurora, sr_check_period should be always 0.

Also add English/Japanese Aurora setting example.

- Support logical replication. (Tatsuo Ishii)

The logical replication mode can be used with PostgreSQL servers operating logical replication. In this mode, PostgreSQL is responsible for synchronizing tables.

Load balancing is possible in the mode. Since logical replication does not replicate all tables, it's user's responsibility to replicate the table which could be load balanced.

The sample configuration file is `$prefix/etc/pgpool.conf.sample-logical`.

And add support for logical replication mode to `pgpool_setup`.

A.28.3. Other Enhancements

- Test: Add some watchdog test cases. (Muhammad Usama)
- Test: Add new test case "node_js.data" to extended-query-test. (Tatsuo Ishii)
- Doc: Documentation updates for pool_hba enhancements. (Muhammad Usama)
- Add "slony mode" to pgpool_setup. (Tatsuo Ishii)
- Deal with OpenSSL 1.1. (Tatsuo Ishii, Muhammad Usama)

- Test: Add new regression test "069.memory_leak_extended". (Tatsuo Ishii)
- Doc: Enhance query cache documents. (Tatsuo Ishii)
- Doc: Add "Tips for Installation" section. (Tatsuo Ishii)
- Test: Add new test 011.watchdog_quorum_failover. (Tatsuo Ishii)
- Test: Add new test suits. (Tatsuo Ishii)

The new test suit "extended-query-test" is intended to test extended queries using pgproto command.

- Add debugging aid to check pending message and backend response. (Tatsuo Ishii)

New function `pool_check_pending_message_and_reply()` added. If pending message kind and backend reply message kind is not inconsistent, it prints a debug message. Currently the only client of the function is `read_kind_from_backend()`.

- Add debugging/testing aid for health check. (Tatsuo Ishii)

A.28.4. Changes

- Add "role" field to `pcp_node_info` command. (Tatsuo Ishii)

Role is the server role: one of "primary", "standby", "master" or "slave".

- Change `master_slave_sub_mode` default to 'stream'. (Tatsuo Ishii)

This is necessary since the default value for the `pool_config_variable.c` is `STREAM_MODE`.

- Allow to fork new health check process when it exits. (Tatsuo Ishii)
- Add new group "health_check" to `PGPOOL SHOW` command doc. (Tatsuo Ishii)
- Remove old html docs and related files. (Tatsuo Ishii)

A.28.5. Bug fix

- Fix Pgpool-II hanging after receiving 'H' (flush) message in streaming replication mode. ([bug 345](#)) (Tatsuo Ishii)

- Fix exit signal handlers to not call `ereport`. (Tatsuo Ishii)

There could be a race condition in the exit signal handlers.

See [\[pgpool-hackers:2545\]](#) for more details.

- Doc: Fix table about `replicate_select` behavior. (Yugo Nagata)
- Test: Fix bug with memory leak tests and add new memory leak test. (Tatsuo Ishii)
- Add log after health check retrying succeeds. (Tatsuo Ishii)

Previously only health check retry logs were produced. So it was not clear if the retry succeeded or not in the end.

- Add necessary setting for health check retry in the Aurora example. (Tatsuo Ishii)

A.29. Release 3.6.21

Release Date: 2020-05-21

A.29.1. Bug fixes

- Fix build error on some system (Fedora 32). (Tatsuo Ishii)
- Fix unnecessary checks. (Tatsuo Ishii)
Patch contributed by sherlockcpp. Discussion: [\[pgpool-general: 7004\]](#)
- Doc: Fix typo. (Tatsuo Ishii)
Patch contributed by Umar Hayat. Discussion: [\[pgpool-hackers: 3587\]](#)
- Fix for segmentation fault in PCP process. ([bug 598](#)) (Muhammad Usama)
- Doc: Fix typo. (Tatsuo Ishii, Bo Peng)
Patch provided by sherlockcpp.
- Fix possible data inconsistency in native replication mode. (Tatsuo Ishii)
Discussions: [\[pgpool-general: 6896\]](#) [\[pgpool-hackers: 3540\]](#)
- Fix watchdog ping probes fail with long hostnames due to small buffer. ([bug 516](#)) (Bo Peng)

A.30. Release 3.6.20

Release Date: 2020-02-20

A.30.1. Changes

- Check if socket file exists at startup and remove them if PID file doesn't exist to avoid bind() failure. (Bo Peng)

A.30.2. Bug fixes

- Fix incorrect query rewriting in native replication mode. ([bug 551](#)) (Bo Peng)
- Doc: Update documentation and fix documentation typos. (Takuma Hoshiai, Tatsuo Ishii, Bo Peng)
- Fix [pgpool_setup](#) to support PostgreSQL 12. (Tatsuo Ishii)
- Fix occasional regression test failure. (Tatsuo Ishii)
- Fix replication delay worker segfault when application_name in primary_conninfo is an empty string. ([bug 565](#)) (Tatsuo Ishii)
- Fix the bug with ignored syslog setting. (Bo Peng)

A.31. Release 3.6.19

Release Date: 2019-10-31

A.31.1. Bug fixes

- Fix incorrect query rewrite in replication mode. (Bo Peng)
 - Doc: add [failover_command](#) description when all standby nodes are down. (Takuma Hoshiai)
 - Doc: add note not to assign PostgreSQL servers to [trusted_servers](#). (Tatsuo Ishii)
 - Fix assorted ancient v2 protocol bugs. (Tatsuo Ishii)
 - Fix problem that [syslog_facility](#) doesn't change by reload. ([bug 548](#)) (Takuma Hoshiai)
 - Fix Pgpool-II shutdown failed in certain case. (Tatsuo Ishii)
 - Allow the lost standby node to rejoin the master watchdog node when it gets rediscovered by the lifecheck. ([bug 545](#)) (Muhammad Usama)
 - Fix segfault when executing an erroneous query after DEALLOCATE a named statement. ([bug 546](#)) (Tatsuo Ishii)
 - Add "-l" option to [arping_cmd](#) command default setting. (Bo Peng)
-

A.32. Release 3.6.18

Release Date: 2019-08-15

A.32.1. Enhancements

- Import some of memory manager debug facilities from PostgreSQL. (Tatsuo Ishii)
 - Use `pg_get_expr()` instead of `pg_attrdef.adsrc` to support for PostgreSQL 12. (Bo Peng)
 - Enhance shutdown script of [pgpool_setup](#). (Tatsuo Ishii)
 - Make shutdownall to wait for completion of shutdown of Pgpool-II.
 - If environment variable `CHECK_TIME_WAIT` is set to true, use `netstat` command to confirm usage of the TCP/IP port while executing shutdown script.
 - Doc: add description to [pg_md5](#) man page how to show `pool_passwd` hashed string. (Tatsuo Ishii)
 - Doc: add general description about failover. (Tatsuo Ishii)
 - Deal `pgpool_adm` extension with PostgreSQL 12. (Tatsuo Ishii)
-

A.32.2. Bug fixes

- Fix "unable to bind. cannot get parse message" error. ([bug 531](#)) (Tatsuo Ishii)
- Fix online-recovery is blocked after a child process exits abnormally with replication mode and watchdog. ([bug 483](#)) (Muhammad Usama)
- Fix [watchdog_setup](#) command `mode` option to work correctly. (Takuma Hoshiai)
- Fix [pgpool_setup](#) to produce correct follow master command. (Tatsuo Ishii)
- Fix query cache module so that it checks oid array's bound. (Tatsuo Ishii)

- Fix off-by-one error in query cache module. (Tatsuo Ishii)
- Allow health check process to reload. (Tatsuo Ishii)
- Fix segfault when query cache is enabled. ([bug 525](#)) (Tatsuo Ishii)
- Down grade LOG "checking zapping sent message ..." to DEBUG5. (Tatsuo Ishii)

Discussion: [\[pgpool-general: 6620\]](#)

- Fix segfault when `samenet` is specified in `pool_hba.conf`. (Tatsuo Ishii)
Discussion: [\[pgpool-general: 6601\]](#).
 - Doc: Fix document typos. (Bo Peng)
 - Fix to deal with backslashes according to the config of `standard_conforming_strings` in native replication mode. ([bug 467](#)) (Bo Peng)
 - Fix compile error on FreeBSD. ([bug 512](#), [bug 519](#)) (Bo Peng)
 - Fix memory leaks. (Tatsuo Ishii)
 - Make failover in progress check more aggressively to avoid potential segfault. (Tatsuo Ishii)
-

A.33. Release 3.6.17

Release Date: 2019-05-16

A.33.1. Enhancements

- Doc: Improve [Reference II, PCP commands](#) document. (Tatsuo Ishii)
- Speed up failover when all of backends are down. (Tatsuo Ishii)

If all of the backend are in down status, immediately give up finding primary node regardless `search_primary_node_timeout` and promptly finish the failover process.

Discussion: [\[pgpool-hackers: 3321\]](#)

- `pgpool-recovery` extension and `pgpool_setup` is now ready for the next major release PostgreSQL 12. (Tatsuo Ishii)
 - Doc: add [restrictions](#) entry. (Takuma Hoshiai)
-

A.33.2. Bug fixes

- Fix the wrong error message "ERROR: connection cache is full", when all backend nodes are down. ([bug 487](#)) (Bo Peng)

When all backend nodes are down, `Pgpool-II` throws an incorrect error message "ERROR: connection cache is full". Change the error message to "all backend nodes are down, `pgpool` requires at least one valid node".

- Remove unused `.sgml` file. (Takuma Hoshiai)
- Avoid exit/fork storm of `pool_worker_child` process. (Tatsuo Ishii)

`pool_worker_child` issues query to get WAL position using `do_query()`, which could throws FATAL error. In this case `pool_worker_child` process exits and `Pgpool-II` parent immediately forks new process. This cycle

indefinitely repeats and gives high load to the system. To avoid the exit/fork storm, sleep `sr_check_period`.

- Fix [black_function_list](#)'s broken default value. (Tatsuo Ishii)
- Fix "not enough space in buffer" error. ([bug 499](#)) (Tatsuo Ishii)

The error occurred while processing error message returned from backend and the cause is that the query string in question is too big. Problem is, the buffer is in fixed size (8192 bytes). Eliminate the fixed size buffer and use pallocated buffer instead. This also saves some memory copy work.

- Fix DROP DATABASE failure. (Tatsuo Ishii)
- Fix wrong variable in `read_status_file()` function. ([bug 493](#)) (Takuma Hoshiai)
- Add missing `test/watchdog_setup` to EXTRA_DIST. ([bug 470](#)) (Bo Peng)
- Doc: mention that multi-statement queries are sent to primary node only. ([bug 492](#)) (Tatsuo Ishii)
- Abort session if failover/fallback is ongoing to prevent potential segfault. ([bug 481](#), [bug 482](#)) (Tatsuo Ishii)
- Fix compiler warnings. (Tatsuo Ishii)
- Fix memory leak in "batch" mode in extended query. ([bug 468](#)) (Tatsuo Ishii)

A.34. Release 3.6.16

Release Date: 2019-03-29

A.34.1. Enhancements

- Add new configuration option [ssl_prefer_server_ciphers](#). (Muhammad Usama)

Add the new setting [ssl_prefer_server_ciphers](#) to let users configure if they want client's or server's cipher order to take preference.

The default for this parameter is off, which prioritize the client's cipher order as usual. However this is just for keeping backward compatibility, and it is possible that a malicious client uses weak ciphers. For this reason we recommend to set this parameter to on at all times.

- Allow to set a client cipher list. (Tatsuo Ishii, Yugo Nagata)

For this purpose new parameter [ssl_ciphers](#), which specifies the cipher list to be accepted by Pgpool-II, is added. This is already implemented in PostgreSQL and useful to enhance security when SSL is enabled.

A.34.2. Bug fixes

- Fix unnecessary `fsync()` to `pgpool_status` file. (Tatsuo Ishii)

Whenever new connections are created to PostgreSQL backend, `fsync()` was issued to `pgpool_status` file, which could generate excessive I/O in certain conditions. So reduce the chance of issuing `fsync()` so that it is issued only when backend status is changed.

Discussion: [\[pgpool-general: 6436\]](#)

A.35. Release 3.6.15

Release Date: 2019-02-21

A.35.1. Bug fixes

- Test: Fix old JDBC functions and typos in regression test 068.memqcache_bug. (Takuma Hoshiai)
- Doc: Fix configuration change timing regarding [memory_cache_enabled](#). (Tatsuo Ishii)
- Fix online recovery failed due to [client_idle_limit_in_recovery](#) in certain cases. ([bug 431](#)) (Tatsuo Ishii)
- Fix corner case bug when `strip_quote()` handle a empty query string. ([bug 458](#)) (Tatsuo Ishii)
- Doc: Mention that schema qualifications cannot be used in `white/black_function_list`. (Tatsuo Ishii)
- Fix typo about `wd_priority` in `watchdog_setup`. (Takuma Hoshiai)
- Fix Pgpool child segfault if failover occurs when trying to establish a connection. (Tatsuo Ishii)
- Doc: fix typo in `logdir` description. ([bug 453](#)) (Tatsuo Ishii)
- Fix Pgpool-II hang if a client sends a extended query message such as close after sync message but before next simple query. (Tatsuo Ishii)

Discussion: [\[pgpool-hackers: 3164\]](#)

- Fix Pgpool-II hang when `idle_in_transaction_session_timeout = on`. ([bug 448](#)) (Tatsuo Ishii)
- Doc: Fix Japanese document typo in [pcp_common_options](#). (Bo Peng)

A.36. Release 3.6.14

Release Date: 2018-11-22

A.36.1. Bug fixes

- Fix to sort startup packet's parameters sent by client. ([bug 444](#))(Takuma Hoshiai)
If order of startup packet's parameters differ between cached connection pools and connection request, didn't use connection pool ,and created new connection pool.
- Fix segmentation fault occurs when a certain Bind message is sent in native replication mode. ([bug 443](#))(Bo Peng)
If the number of parameter format codes is specified to one, but the number of the original query's parameter is zero, `bind_rewrite_timestamp()` will call `memcpy` with a negative value. This causes segmentation fault.
Patch is provided by Yugo Nagata.
- Fix a query passed to `relcache` so that it uses schema qualified table name. (Tatsuo Ishii)
- Fix query cache invalidation bug. (Tatsuo Ishii)
- Fix segfault in extended query + query cache case. (Tatsuo Ishii)
- Fix memory leak in extended query + query cache enabled. (Tatsuo Ishii)

A.37. Release 3.6.13

Release Date: 2018-10-31

A.37.1. Changes

- Allow `PCP[attach/detach/promote]` commands during failover. (Muhammad Usama)
- Change `pgpool.spec` file to install extension to DB server which supports LLVM JIT. (Bo Peng)
- Doc: Add note to online recovery doc. (Tatsuo Ishii)

This warns that `client_idle_limit_in_recovery` must be smaller than `recovery_timeout`.

- Test: Add regression test for SSL connection. (Tatsuo Ishii)
- Doc: Add notes regarding failover script. (Tatsuo Ishii)

It's not recommended to access against `Pgpool-II` itself from failover/failback scripts.

- Doc: Improve `pg_md5` docs and error message. (Bo Peng)

Patch provided by Jesper Pedersen and modified by me.

- Test: Add definition of PGLIB in `regress.sh`. (Bo Peng)

Patch provided by Jesper Pedersen.

A.37.2. Bug fixes

- Fix typo in `child_max_connections` description of `SHOW POOL_STATUS` output. (Tatsuo Ishii)

Patch provided by Phil Ramirez.

- Fix segmentation fault when error query and Sync message are sent in native replication mode. ([bug 434](#)) (Takuma Hoshiai)

In native replication mode, segmentation fault occurs when Sync messages is sent just after a query error.

- Fix syntax error when queries including time functions and `IN (SELECT ...)` in `WHERE` clause in native replication mode. ([bug 433](#)) (Bo Peng)

In native replication mode, queries including time functions (e.g. `now()`, `CURRENT_TIMESTAMP` etc.) are rewritten to a timestamp constant value. However, `Pgpool-II` doesn't support queries including time functions and `IN (SELECT ...)` in `WHERE` clause.

- Fix occasional less data returned to frontend with extended protocol. ([bug432](#)) (Tatsuo Ishii)

The idea for fix is, use pending message data list. It records messages from frontend, and it is expected that we will receive same number of messages.

Initial patch is created by Yugo Nagata and fixed by Tatsuo Ishii.

- Fix memory leak when query cache enabled in streaming replication mode + extended query case. (Tatsuo Ishii)
- Fix memory leak in `trigger_failover_command()`. (Tatsuo Ishii)
- Fix memory leak when `memory_cache_enabled = on` and write SQLs are sent. (Bo Peng)

In an explicit transaction, the `SELECT` results are cached in temporary buffer. If a write `SQL` is sent which modifies the table, the temporary buffer should be reset.

- Fix occasional failure in regression 065.bug152. (Tatsuo Ishii)
- Test: Add `EXECUTE/DEALLOCATE` regression test. (Takuma Hoshiai)
- Add missing `pgpool_recovery--1.0--1.1.sql` file to update `pgpool_recovery()` function version to 1.1. (Bo Peng)
- Fix kind mismatch error when `DEALLOCATE` statement is issued. (Bo Peng)

`PREPARE` should be added to `pool_add_sent_message`, so that `EXECUTE` and `DEALLOCATE` can be sent to the same node as `PREPARE`.

See [\[pgpool-general: 6226\]](#) for more details.

- Do not update `pool_passwd` if the password length is incorrect. ([bug 419](#)) (Takuma Hoshiai, Tatsuo Ishii)

For Pgpool-II 3.7 or before, the password stored in `pool_passwd` is MD5 password only. So check the correctness of `pool_passwd` by scanning entire file.

- Doc: Change `follow_master_command` description "new master" to "new primary". (Bo Peng)
- Fix newer version of gcc warnings. (Tatsuo Ishii)
- Test: Update `clean.sh` which clean up regression test results. (Bo Peng)

Patch provided by Jesper Pedersen.

- Add `.gitignore` files. (Bo Peng)

Patch provided by Jesper Pedersen.

- Fix segfault when node 0 is in down status in case of both health check and `failover_on_backend_error` are disabled. (Tatsuo Ishii)
- Doc: Fix typos in documents and scripts. (Tatsuo Ishii)

Patch contributed by Jesper Pedersen.

- Doc: Fix document mistakes of `recovery_1st_stage_command` and `recovery_2nd_stage_command`. (Bo Peng)

A.38. Release 3.6.12

Release Date: 2018-07-31

A.38.1. Bug fixes

- Fix "write on backend 0 failed with error :\"Success\"" error. ([bug 403](#)) (Tatsuo Ishii)

Don't treat it as an error if `write()` returns 0.

- Fix for 0000409: worker process is not restarted after failover on standby. ([bug 409](#)) (Muhammad Usama)

Patch contributed by Yugo Nagata.

- Fix memory leaks related to `pool_extract_error_message()`. (Tatsuo Ishii)
- Fix an incorrect declare as `bool`, rather than `int` in `pool_extract_error_message()`. (Tatsuo Ishii)

This led to a segfault issue mentioned on certain platform.

- Fix segfault in `per_node_error_log()` on armhf architecture. (Tatsuo Ishii)

Patch provided by Christian Ehrhardt.

- Doc: Improve documents of "MD5 Password Authentication", "Installing Pgpool-II" and "pg_md5".(Bo Peng)

Patch provided by Takuma Hoshiai.

- Test: Fix 006.memqcache test failure. (Tatsuo Ishii)
-

A.39. Release 3.6.11

Release Date: 2018-06-12

A.39.1. Bug fixes

- Fix Pgpool-II hung if replication delay is too much, when query cache enabled in extended query mode. (Tatsuo Ishii)

See [\[pgpool-general-jp: 1534\]](#) for more details.

- Doc: Fix document typo of PCP commands option "-U". (Bo Peng)
- Delete some debug code. (Bo Peng)
- In extended query mode, do not set writing tx flag with SET TRANSACTION READ ONLY. (Tatsuo Ishii)
- Doc: Update outdated [pcp_proc_info](#) manual. (Tatsuo Ishii)
- Doc: Enhance online recovery document to Clarify that [recovery_2nd_stage_command](#) is only required in native replication mode. (Tatsuo Ishii)
- Prevent [pcp_recovery_node](#) from recovering "unused" status node. (Tatsuo Ishii)

This allowed to try to recovery a node without configuration data, which leads to variety of problems.

See [\[pgpool-general: 5963\]](#) for more details.

Also I fixed `pgpool_recovery` function so that it quotes an empty string argument with double quotes.

A.40. Release 3.6.10

Release Date: 2018-04-17

A.40.1. Bug fixes

- Test: Add new regression test for node 0 is down. (Tatsuo Ishii)
- Make calls to `to_regclass` fully schema qualified. (Tatsuo Ishii)

- Doc: Improve watchdog documents. (Tatsuo Ishii)
 - Test: Improve test script 003.failover. (Bo Peng)
 - Deal with "unable to bind D cannot get parse message "S1" error. (Tatsuo Ishii)
 - Doc: Mention that users can avoid failover using backend_flag even PostgreSQL admin shutdown. (Tatsuo Ishii)
 - Doc: Fix document typos. (Bo Peng)
 - Fix `pgpool_setup` failure in replication mode. (Tatsuo Ishii)
 - Allow to support `pgpool_switch_xlog` PostgreSQL 10. (Tatsuo Ishii)
 - Doc: Fix `pgpool_adm` family functions examples. (Tatsuo Ishii)
-

A.41. Release 3.6.9

Release Date: 2018-02-13

Note: This release fixed the bug with socket writing added in Pgpool-II 3.7.0, 3.6.6 and 3.5.10. Due to this bug, when the network load is high, an illegal message may be sent to the frontend or backend. All users using 3.7.x, 3.6.6 or later, 3.5.10 or later versions of Pgpool-II should update as soon as possible.

A.41.1. Changes

- Allow to build with `libressl`. (Tatsuo Ishii)
See [\[pgpool-hackers: 2714\]](#) for more details. Patch by Sandino Araico Sanchez.
 - Set `TCP_NODELAY` and non blocking to frontend socket. (Tatsuo Ishii)
`TCP_NODELAY` is employed by PostgreSQL, so do we it.
 - Change systemd service file to use `STOP_OPTS="-m fast"`. (Bo Peng)
 - Change `pgpool_setup` to add `restore_command` in `recovery.conf`. (Bo Peng)
-

A.41.2. Bug fixes

- Fix writing transaction flag is accidentally set at commit or rollback. (Tatsuo Ishii)
- Doc: Fix document typos. (Bo Peng)
- Fix bug with socket writing. (Tatsuo Ishii)
`pool_write_flush()` is responsible for writing to sockets when `pgpool`'s write buffer is full (this function was introduced in 3.6.6 etc). When network write buffer in kernel is full, it does retrying but it forgot to update the internal buffer pointer. As a result, broken data is written to the socket. This results in variety of problems including too large message length.
- Fix segfault when `%a` is in `log_line_prefix` and debug message is on. ([bug 376](#)) (Tatsuo Ishii)
- Fix queries hanging in `parse_before_bind` with extended protocol and replication + load-balancing. ([bug](#)

A.42. Release 3.6.8

Release Date: 2018-01-09

A.42.1. Bug fixes

- Doc: Fix document typo and mistakes. (Bo Peng)
- Replace `/bin/ed` with `/bin/sed` in [pgpool_setup](#), because `/bin/sed` is included in most distribution's base packages. (Tatsuo Ishii)
- Change the `pgpool.service` and `sysconfig` files to output `Pgpool-II` log. (Bo Peng)

Removing "Type=forking" and add `OPTS="-n"` to run `Pgpool-II` with non-daemon mode, because we need to redirect logs. Using `journalctl` command to see `Pgpool-II` systemd log.

- Add documentation "[Compiling and installing documents](#)" for SGML document build. (Tatsuo Ishii)
- Fix timestamp data inconsistency by replication mode. (Bo Peng)

From PostgreSQL10 the column default value such as 'CURRENT_DATE' changes, `Pgpool-II` didn't rewrite timestamp by the added default values. This caused data inconsistency.

- Doc: Fix [watchdog_setup](#) doc. (Tatsuo Ishii)

It lacked to mention that it supports logical replication mode.

- Downgrade a log message to debug message. (Tatsuo Ishii)

That was mistaken left while last development cycle.

- Fix for re-sync logic in reading packet from backend. (Tatsuo Ishii)

`read_kind_from_backend()`, which reads message kind from backend, re-syncs backend nodes when a ready for query message is received. Unfortunately it forgot to call `pool_pending_message_pull_out()` to delete sync pending message. This leads to random stuck while reading packets from backend. Fix this to call `pool_pending_message_pull_out()`.

- Fix `Pgpool-II` hangs. ([bug 370](#)) (Tatsuo Ishii)

If an erroneous query is sent to primary and without a sync message the next query that requires a catalog cache look up is `send`, `Pgpool-II` hangs in `do_query()`.

- Add `SL_MODE` macro for upper compatibility with `Pgpool-II` 3.7 or later. (Tatsuo Ishii)
- Fix returning transaction state when "ready for query" message received. (Tatsuo Ishii)

We return primary or master node state of ready for query message to frontend. In most cases this is good. However if other than primary node or master node returns an error state (this could happen if load balance node is other than primary or master node and the query is an erroneous `SELECT`), this should be returned to frontend, because the frontend already received an error.

- Fix `pgpool` start message printed multiple times. (Tatsuo Ishii)
- Add an execute permission bit to the start/stop script in [watchdog_setup](#). (Tatsuo Ishii)

A.43. Release 3.6.7

Release Date: 2017-11-01

A.43.1. Bug fixes

- Add different `pgpool.sysconfig` file for RHEL6 and RHEL7. ([bug 343](#)) (Bo Peng)

In RHEL6, the "-n" option is needed to redirect log.

- Fixing an issue in the handling of `pg_terminate_backend()`. (Muhammad Usama)

In some cases `pg_terminate_backend()` can cause failover even when the call is properly issued through Pgpool-II.

- Fix for bug in watchdog where sometime failover is not reliably performed. (Muhammad Usama)

Currently watchdog process only considers the node's watchdog state before deciding if it can handle the failover and failover-locking requests.

But the problem with this technique is that, for the instance when the node has announced itself as a master/coordinator of the cluster and is waiting for the standby nodes to recognise it as a Master node. For that period of time the watchdog state of the node is Master/coordinator, but it is yet not fully capable of handling the failover and failover-locking requests.

So sometimes this leads to a situation where a failover is not reliably performed in case when the failover request arrives while watchdog cluster is in the process of electing a leader.

The fix for the above situation is to make sure that the node has fully acquired the Master status before accepting the failover and failover-locking requests.

- Fix bug with handling of 'H' (flush) message in streaming replication mode. ([bug 345](#)) (Tatsuo Ishii)

If user expects to read response right after 'H', Pgpool-II hangs. The cause was, when 'H' received, extended query mode was reset and pending message was not used.

- Doc: Fix `pcp_node_info` documents. (Tatsuo Ishii)

- Fix bug mistakenly overriding global backend status right after failover. (Tatsuo Ishii)

See [\[pgpool-general: 5728\]](#) for more details.

- Fix exit signal handlers to not call `ereport`. (Tatsuo Ishii)

See [\[pgpool-hackers: 2545\]](#) for more details.

- Deal with OpenSSL 1.1. (Tatsuo Ishii, Muhammad Usama)

- Doc: Fix table about `replicate_select` behavior (Yugo Nagata)

A.44. Release 3.6.6

Release Date: 2017-09-05

A.44.1. Bug fixes

- Fix Pgpool-II hanging when error occurs in streaming replication mode and extended query. (Tatsuo

Ishii)

If backend returns ERROR, Pgpool-II reads message from frontend until a `sync` message is sent. Previous code assumed next message is `sync`. However it is possible that more message coming before the `sync` message, it's a low probability though. Fix it to continue reading messages until the `sync` message is read.

- Fix `wd_authkey` bug in that a request to add new node to the cluster is rejected by master. (Yugo Nagata)

This is a bug due to the implementation of 3.5.6 and 3.6.3.

This changed the definition of `tv_sec` that is used to check `wd_authkey` so that this was affected by the clock of OS. So, if there is a lag between two nodes' clocks, the `wd_authkey` check fails.

- Test: Fix load balance test driver. (Tatsuo Ishii)

Some tests only for native replication mode was executed in streaming replication mode as well.

- Fix not working `slony` mode in extended query. (Tatsuo Ishii)

When response returned from backend, in progress flag was not reset in `slony` mode, which cause waiting for next message from backend in vain.

- Fix ancient bug of stream write modules. (Tatsuo Ishii)

Fix bug with `pool_write_noerror()` when requested length exceeds remaining write buffer size. This could lead to a buffer overrun problem.

When write buffer is full, `pool_flush_it()` is called, which could write data to socket in the middle of message. To fix the problem directly write requested data if the write buffer is going to be full.

Enhance performance of `pool_unread()`.

- Test: Some miscellanies small fixes in regression test scripts. (Muhammad Usama)

- Doc: Fix documentation about load-balancing. (Yugo Nagata)

- Fix core dump and mishandling of temp tables. (Tatsuo Ishii)

- Fix ancient bug of `pool_unread()`. (Tatsuo Ishii)

When `realloc()` is called in `pool_unread()`, it did not update the buffer size. This could cause variety of memory corruption and unexpected data reading from backend. The reason why we did not found that is, probably recently Pgpool-II starts extensively to use `pool_unread()`.

- Fix handling of empty queries. ([bug 328](#)) (Tatsuo Ishii)

When empty query (empty string or all comment query) is sent, command complete message was returned to frontend. This is not correct. An empty query response should be returned to frontend.

- Fix query cache bug with streaming replication mode and extended query case. (Tatsuo Ishii)

- Fix memory leak with streaming replication mode/extended query case. ([bug 324](#)) (Tatsuo Ishii)

- Test: Fix Java program in 005.regression test. (Tatsuo Ishii)

- Fix for when failover is triggered by worker process, it is possible that wrong DB node could failover. ([bug 303](#)) (Tatsuo Ishii)

This is due to the `db_node_id` member in the `POLL_CONNECTION` structure is not initialized in the process (in child process the member is properly initialized). To solve the problem, add new function `pool_set_db_node_id()` to set the structure member variable and call it inside `make_persistent_db_connection()`.

- Fix starting unnecessary transaction when `SET` command is issued. (Tatsuo Ishii)

- Fix for [\[pgpool-general: 5621\]](#) Failover() function should be executed with health check alarm disabled. (Muhammad Usama)

- Fix Pgpool-II hung up bug or other errors in error case in extended query in replication mode. (Tatsuo Ishii)

And other fixes in this commit.

- 1) Do not send intended error query to backend in streaming replication mode in `ErrorResponse3()`.
- 2) Fix `pool_virtual_master_db_node_id()` to return the `virtual_master_node_id` only when query is in progress and query context exists.

- Doc: Fix Pgpool-II document typo. (Bo Peng)
 - Allow make dist to include `pgpool.service`. (Yugo Nagata)
-

A.44.2. Enhancements

- Doc: Add new English and Japanese documents of [Pgpool-II + Watchdog Setup Example](#). (Bo Peng)
 - Test: Add more memory leak regression tests. (Tatsuo Ishii)
-

A.45. Release 3.6.5

Release Date: 2017-07-11

A.45.1. Bug fixes

- Fix for [\[pgpool-hackers: 2400\]](#) Garbage output (Muhammad Usama)
Mostly the log messages fixes and few code cleanups.
- Importing the latest changes in the `MemoryManager API` from PostgreSQL code. (Muhammad Usama)
- Fixing 0000306: Pgpool steals back MASTER status. ([bug 306](#)) (Muhammad Usama)
- Fixing [\[pgpool-hackers: 2390\]](#) Problems with the relative paths in daemon mode (Muhammad Usama)
- Adjust function name change in PostgreSQL 10 dev head. (Tatsuo Ishii)

```
pg_current_wal_location -> pg_current_wal_lsn  
pg_last_wal_replay_location -> pg_last_wal_replay_lsn
```

- Fix a possible hang with streaming replication and extended protocol. (Yugo Nagata)

This hang occurred under a certain condition. The following is an example.

- `pgpool.conf` is configured so that all read queries are sent to the standby.
- First, issue a writing query in a transaction block
- After committing the transaction, issue a select query.
- When processing the query, send Describe (statement) message just after Parse.

Without using JDBC, we can reproduce the problem by `pgproto` with the following messages.

```

'Q' "DROP TABLE IF EXISTS test_tbl"
'Y'
'Q' "CREATE TABLE test_tbl(i int)"
'Y'
'Q' "INSERT INTO test_tbl VALUES(1)"
'Y'

'P' "" "BEGIN" 0
'B' "" "" 0 0 0
'E' "" 0
'S'
'Y'

'P' "" "INSERT INTO test_tbl VALUES(1)" 0
'B' "" "" 0 0 0
'E' "" 0
'S'
'Y'

'P' "" "COMMIT" 0
'B' "" "" 0 0 0
'E' "" 0
'S'
'Y'

'P' "S_1" "SELECT * FROM test_tbl" 0
'D' 'S' "S_1"
'B' "C_1" "S_1" 0 0 0
'E' "C_1" 0
'S'
'Y'

'X'

```

To fix it, `parse_before_bind()` should be called only if we are in a transaction block so that we can send `Bind` and `Execute` to the right backend.

- Fix `Pgpool-II` hang when used by erlang applications. (Tatsuo Ishii)

`Erlang` client sends "Describe" message followed by "Flush". So the backend returns "Row description." However `Pgpool-II` forgets to reset the query in progress flag upon receiving "Row description" message, then `Pgpool-II` keeps on waiting for response from backend. This is the cause of erlang client hanging.

Fix is, just reset the query in progress flag upon receiving "Row description" message. Same thing can be said to "no data" message.

See [\[pgpool-general: 5555\]](#) for more details.

- Fix bug with sending bind message to wrong target node. ([bug 314](#)) (Tatsuo Ishii)
- Fix query cache hang when used by node.js. (Tatsuo Ishii)

See [\[pgpool-general: 5511\]](#) for more details.

- Deal with PostgreSQL 10 in streaming replication delay checking. (Tatsuo Ishii)
- Fix query cache memory leak. (Tatsuo Ishii)

Clearing cache buffers in case of no oid queries (like `BEGIN`, `CHECKPOINT`, `VACUUM`, etc) should have been done, but it did not.

- Fix extended query hang in certain case. (Tatsuo Ishii)

`erlang` PostgreSQL API produces `Parse` ('P'), `Describe` ('D'), `Flush` ('H'), `Bind` ('B'), and `Execute` ('E'). Notice the 'H' message (this does not happen in `JDBC`. I suspect that's the reason why this problem is not popular before). After that, `Pgpool-II` dropped the extended query mode, it failed to find which backend to read data. Thus `Pgpool-II` simply tries to read all of backend which causes hang because it may have not send a message to some of backends.

Solution is, after receiving the flush message set doing extended query flag.

- Fix for [\[pgpool-hackers: 2354\]](#) segfault with `pg_md5`. (Muhammad Usama)
- Fix descriptions of `white/black_memcache_table_list`. (Tatsuo Ishii)

They are far from actual implementations.

See [\[pgpool-general: 5479\]](#) for more details.

- Fix corner case bug in `Pgpool-II` starting up. (Tatsuo Ishii)

It is possible that a failover request is accepted before primary node is searched. This leads `Pgpool-II` to a strange state: there's no primary node if the failed node was a primary node (even if new primary node exists as a result of promotion of existing standby).

See [\[pgpool-hackers: 2321\]](#) for more details.

A.46. Release 3.6.4

Release Date: 2017-05-11

A.46.1. Bug fixes

- Fixing a few corner cases in the failover request handling of the watchdog. (Muhammad Usama)
 - Tightening up the watchdog cluster membership criteria. (Muhammad Usama)
 - Enhance document for load balancing. (Tatsuo Ishii)
 - Add node 0 failover test. (Tatsuo Ishii)
 - Fix `Pgpool-II` child process segfault reported in [\[pgpool-hackers: 2312\]](#). (Tatsuo Ishii)
-

A.47. Release 3.6.3

Release Date: 2017-04-28

A.47.1. Bug fixes

- Fix "show pool_cache" segfault when memcached is used. ([Bug 301](#)) (Tatsuo Ishii)
- Fix for some more code warnings. (Muhammad Usama)
- Fixing some annoying compiler warnings. (Muhammad Usama)
- Removing the function defined but not used warnings from `pool_config_variable.c` (Muhammad Usama)
- Removing the references of obsolete `debug_level` configuration parameter. (Muhammad Usama)
- Fixing a mistake in the watchdog code. (Muhammad Usama)
commit also adds some debug messages in the watchdog code.
- Fix for 0000299: Errors on the reloading of configuration. ([Bug 299](#)) (Muhammad Usama)

- Add pgpool_adm English and Japanese docs. (Tatsuo Ishii)
- Fix document indentation. (Tatsuo Ishii)
- Fix for 0000289: Inconsistent backend state. ([Bug 289](#)) (Muhammad Usama)
- Enhancing the handling of split-brain scenario by the watchdog. (Muhammad Usama)

Previously, the watchdog cluster was used to call for re-election of the master/coordinator node whenever the split-brain situation was detected. And consequently every node was required to rejoin the watchdog network, Which was essentially similar to the re-booting of the whole watchdog cluster.

The candidate for the master/coordinator node is selected on the following criteria.

1-- When two watchdog nodes are claiming to be the cluster master, the master node that has performed the escalation keeps the master status and the other node is asked to step down.

2-- If the conflict could not be resolved by the escalation status of the nodes, The node which holds the quorum remains the master/coordinator.

3-- If the quorum status of both contenders is also same. The node with higher number of connected alive nodes get the preference.

4-- Finally, if all above three yields no winner, the older master (The node that has the coordinator status for longer duration) remains the master.

- Enhancing the watchdog internal command mechanism to handle multiple concurrent commands. (Muhammad Usama)
- Fix compiler warnings. (Tatsuo Ishii)
- Comment out unsupported Java method in new JDBC drivers to prevent regression failure. (Tatsuo Ishii)
- Downgrade parse before bind log message to debug1. (Tatsuo Ishii)
- Fix coverity warnings. (Tatsuo Ishii, Muhammad Usama)
- Fix for [\[pgpool-general: 5396\]](#) pam ldap failure. (Muhammad Usama)
- Mention that SQL type commands cannot be used in extended query mode. (Tatsuo Ishii)
- Consider SHOW command as kind of a read query. (Tatsuo Ishii)

In streaming replication mode, if SHOW is issued then subsequent SELECTs are sent to the primary node in an explicit transaction. This is not a reasonable and unnecessary limitation. Also fix hang when parse command returns error.

- Fix memory leak problem caused by commit adcb636. (Tatsuo Ishii)

Commit adcb636 introduces "pending message queue". When a message arrives, the info is added to the queue and a copy of object is created at the same time, but forgot to free the object. Fix is, creating a new function `pool_pending_message_free_pending_message()` and call it after `pool_pending_message_add()`, `pool_pending_message_get()` and `pool_pending_message_pull_out()`. Problem reported by Sergey Kim.

- Mega patch to fix "kind mismatch" (or derived) errors in streaming replication mode. ([Bug 271](#)) (Tatsuo Ishii)

The errors are caused by wrong prediction in which (or both) database node will send response to Pgpool-II. Previous implementation using "sync map" are weak and sometimes fail in the prediction.

This patch introduces new implementation using "pending message queue", which records all sent message to backends. The element of the queue stores info regarding messages types (parse/bind/execute/describe/close/sync), to which database node the message was sent and so on. It's a simple FIFO queue. When a message arrives from backend, by looking at the head of the "pending message queue", it is possible to reliably predict what kind of message and from which database node it will arrive. After receiving the message, the element is removed from the queue.

I would like to thank to Sergey Kim, who has been helping me in testing series of patches.

See [Bug 271](#) and discussion in pgpool-hackers mailing list [[pgpool-hackers: 2043](#)] and [[pgpool-hackers: 2140](#)] for more details.

- Fix for 0000296: Pgpool v3.6.2 terminated by systemd because the service Type has been set to 'forking'. ([Bug 296](#)) (Muhammad Usama)

A.48. Release 3.6.2

Release Date: 2017-03-17

A.48.1. Bug fixes

- Add "Wants=network.target" to pgpool.service file. ([bug 294](#)) (Bo Peng)
- Fix [pcp_promote_node](#) bug that fails promoting node 0. (Yugo Nagata)

The master node could not be promoted by `pcp_promote_node` with the following error;

```
FATAL: invalid pgpool mode for process recovery request
DETAIL: specified node is already primary node, can't promote node id 0
```

In streaming replication mode, there is a case that Pgpool-II regards the status of primary node as "standby" for some reasons, for example, when `pg_ctl promote` is executed manually during Pgpool-II is running, in which case, it seems to Pgpool-II that the primary node doesn't exist.

This status mismatch should be fixed by `pcp_promote_node`, but when the node is the master node (the first alive node), it fails as mentioned above.

The reason is as following. before changing the status, `pcp_promote_node` checks if the specified node is already primary or not by comparing the node id with `PRIMARY_NODE_ID`. However, if the primary doesn't exist from Pgpool-II's view, `PRIMARY_NODE_ID` is set to 0, which is same as `MASTER_NODE_ID`. Hence, when the master node is specified to be promoted, `pcp_promote_node` is confused that this node is already primary and doesn't have to be promoted, and it exits with the error.

To fix this, `pcp_promote_node` should check the node id by using `REAL_PRIMARY_NODE_ID`, which is set -1 when the primary doesn't exist, rather than `PRIMARY_NODE_ID`.

- Fix document error. (Tatsuo Ishii, Bo Peng)
- Pgpool-II should not perform ping test after bringing down the VIP. (Muhammad Usama)

This issue was reported by the reporter of bug:[[pgpool-II 0000249](#)]: watchdog sometimes fails de-escalation

- Fix to release shared memory segments when Pgpool-II exits. ([bug 272](#)) (Tatsuo Ishii)
- Fix for [[pgpool-general: 5315](#)] `pg_terminate_backend` (Muhammad Usama)
- Adding the missing `ExecStop` and `ExecReload` commands to the systemd service configuration file. (Muhammad Usama)
- Fix for 281: "segmentation fault" when execute [pcp_attach_node](#). ([bug 281](#)) (Muhammad Usama)
- Fix load balancing bug in streaming replication mode. (Tatsuo Ishii)

In an explicit transaction, any `SELECT` will be load balanced until write query is sent. After writing query is sent, any `SELECT` should be sent to the primary node. However if a `SELECT` is sent before a sync message is sent, this does not work since the treatment of writing query is done after ready for query message arrives.

Solution is, the treatment for writing query is done in executing the writing query as well.

The bug has been there since V3.5.

- Fix yet another kind mismatch error in streaming replication mode. (Tatsuo Ishii)
- Fix `do_query()` hangs after close message. (Tatsuo Ishii)
- Fixing stack smashing detected. ([bug 280](#)) (Muhammad Usama)

It was a buffer overflow in `wd_get_cmd` function

- Fixing the issue with the watchdog process restart. (Muhammad Usama)

When the watchdog process gets abnormally terminated because of some problem (e.g. Segmentation fault) the new spawned watchdog process fails to start and produces an error "bind on ... failed with reason: Address already in use".

Reason is the abnormally terminating watchdog process never gets the time to clean-up the socket it uses for IPC and the new process gets an error because the socket address is already occupied.

Fix is, the Pgpool main process sets the flag in shared memory to mark the watchdog process was abnormally terminated and at startup when the watchdog process see that the flag is set, it performs the clean up of the socket file and also performs the de-escalation (If the watchdog process was crashed when it was master/coordinator node) if required before initializing itself.

- Fix query cache bug reported in [pgpool-general-jp:1441](#). (Tatsuo Ishii)

In streaming replication mode with query cache enabled, SELECT hangs in the following scenario:

- 1) a SELECT hits query cache and returns rows from the query cache.
- 2) following SELECT needs to search meta data and it hangs.

In #1, while returning the cached result, it misses to call `pool_unset_pending_response()`, which leave the `pending_response` flag be set. In #2, `do_query()` checks the flag and tries to read pending response from backend. Since there's no pending data in backend, it hangs in reading data from backend.

Fix is, just call `pool_unset_pending_response()` in #1 to reset the flag.

Bug report and fix provided by Nobuyuki Nagai. New regression test item (068) added by me.

- Remove `elog/ereport` calls from signal handlers. (Tatsuo Ishii)

See [\[pgpool-hackers: 1950\]](#) for details.

- Fix bug failed to create INET domain socket in FreeBSD if `listen_addresses = '*'`. ([bug 202](#)) (Bo Peng)
- Fix for 0000249: watchdog sometimes fails de-escalation. ([bug 249](#)) (Muhammad Usama)

The solution is to use the `waitpid()` system call without `WNOHANG` option.

- Fix `connection_life_time` broken by `authentication_timeout`. (Yugo Nagata)
- Fix authentication timeout that can occur right after client connections. (Yugo Nagata)

A.49. Release 3.6.1

Release Date: 2016-12-26

A.49.1. Bug fixes

- Tightening up the watchdog security. (Muhammad Usama)

Now wd_authkey uses the HMAC SHA-256 hashing.

- Add pgpool_adm extension in Pgpool-II RPM. (Bo Peng)
- Fix occasional segfault when query cache is enabled. (bug 263) (Tatsuo Ishii)
- Fix packet kind does not match error in extended protocol. (bug 231) (Tatsuo Ishii)

According to the bug231, the bug seem to bite you if all of following conditions are met:

- Streaming replication mode
- Load balance node is not node 0
- Extended protocol is used
- SELECT is executed, the statement is closed, then a transaction command is executed

The sequence of how the problem bites is:

1. SELECT executes on statement S1 on the load balance node 1
2. Frontend send Close statement
3. Pgpool-II forward it to backend 1
4. Frontend sends Parse, Bind, Execute of COMMIT
5. Pgpool-II forward it to backend 0 & 1
6. Frontend sends sync message
7. Pgpool-II forward it to backend 0 & 1
8. Backend 0 replies back Parse complete ("1"), while backend 1 replies back close complete ("3") because of #3.
9. Kind mismatch occurs

The solution is, in #3, let Pgpool-II wait for response from backend 1, but do not read the response message. Later on Pgpool-II's state machine will read the response from it before the sync message is sent in #6. With this, backend 1 will reply back "1" in #8, and the kind mismatch error does not occur.

Also, fix not calling pool_set_doing_extended_query_message() when receives Close message. (I don't know why it was missed).

New regression test "067.bug231" was added.

- Fix a race condition in a signal handler. (bug 265) (Tatsuo Ishii)

In child.c there's signal handler which calls elog. Since the signal handler is not blocked against other signals while processing, deadlock could occur in the system calls in the pgpool shutdown sequence. To fix the problem, now the signal handler is blocked by using POOL_SETMASK.

Ideally we should avoid calling elog in signal handlers though.

- Fix wrong minimum configuration value for client_idle_limit_in_recovery. (bug 264) (Tatsuo Ishii)
- Allow to execute "make xslthtml" under doc.ja. (Tatsuo Ishii)

A.50. Release 3.6

Release Date: 2016-11-21

A.50.1. Overview

Major enhancements in Pgpool-II 3.6 include:

- Improve the behavior of failover. In the streaming replication mode, client sessions will not be disconnected when a failover occurs any more if the session does not use the failed standby server. If the primary server goes down, still all sessions will be disconnected. Also it is possible to connect to Pgpool-II even if it is doing health checking retries. Before all attempt of connecting to Pgpool-II failed while doing health checking retries.
- New PGPOOL SET command has been introduced. Certain configuration parameters now can be changed on the fly in a session.
- Watchdog is significantly enhanced. It becomes more reliable than previous releases.
- Handling of extended query protocol (e.g. used by Java applications) in streaming replication mode speeds up if many rows are returned in a result set.
- Import parser of PostgreSQL 9.6.
- In some cases `pg_terminate_backend()` now does not trigger a failover.
- Change documentation format from raw HTML to SGML.

The above items are explained in more detail in the sections below.

A.50.2. Major Enhancements

- Improve the behavior of failover. (Tatsuo Ishii)

In the streaming replication mode, client sessions will not be disconnected when a failover occurs any more if the session does not use the failed standby server. If the primary server goes down, still all sessions will be disconnected. Health check timeout case will also cause the full session disconnection. Other health check error, including retry over case does not trigger full session disconnection.

For user's convenience, "show pool_nodes" command shows the session local load balance node info since this is important for users in case of failover. If the load balance node is not the failed node, the session will not be affected by failover.

Also now it is possible to connect to Pgpool-II even if it is doing health checking retries. Before all attempt of connecting to Pgpool-II failed while doing health checking retries. Before any attempt to connect to Pgpool-II fails if it is doing a health check against failed node even if [failover_on_backend_error](#) is off because Pgpool-II child first tries to connect to all backend including the failed one and exits if it fails to connect to a backend (of course it fails). This is a temporary situation and will be resolved once pgpool executes failover. However if the health check is retrying, the temporary situation keeps longer depending on the setting of [health_check_max_retries](#) and [health_check_retry_delay](#). This is not good. Attached patch tries to mitigate the problem:

When an attempt to connect to backend fails, give up connecting to the failed node and skip to other node, rather than exiting the process if operating in streaming replication mode and the node is not primary node.

Mark the local status of the failed node to "down". This will let the primary node be selected as a load balance node and every queries will be sent to the primary node. If there's other healthy standby nodes, one of them will be chosen as the load balance node.

After the session is over, the child process will suicide to not retain the local status.

- Add [PGPOOL SHOW](#), [PGPOOL SET](#) and [PGPOOL RESET](#) commands. (Muhammad Usama)

These are similar to the PostgreSQL's SET and SHOW commands for GUC variables, adding the functionality in Pgpool-II to set and reset the value of config parameters for the current session, and for that it adds a new syntax in Pgpool-II which is similar to PostgreSQL's SET and RESET variable syntax with an addition of PGPOOL keyword at the start.

Currently supported configuration parameters by PGPOOL SHOW/SET/RESET are: [log_statement](#), [log_per_node_statement](#), [check_temp_table](#), [check_unlogged_table](#), [allow_sql_comments](#), [client_idle_limit](#), [log_error_verbosity](#), [client_min_messages](#), [log_min_messages](#), [client_idle_limit_in_recovery](#).

- Sync inconsistent status of PostgreSQL nodes in Pgpool-II instances after restart. (bug 218) (Muhammad Usama)

Watchdog does not synchronize status.

- Enhance performance of SELECT when lots of rows involved. (Tatsuo Ishii)

Pgpool-II flushes data to network (calling write(2)) every time it sends a row data ("Data Row" message) to frontend. For example, if 10,000 rows needed to be transfer, 10,000 times write()s are issued. This is pretty expensive. Since after repeating to send row data, "Command Complete" message is sent, it's enough to issue a write() with the command complete message. Also there are unnecessary flushing are in handling the command complete message.

[Quick testing](#) showed that from 47% to 62% performance enhancements were achieved in some cases.

Unfortunately, performance in workloads where transferring few rows, will not be enhanced since such rows are needed to flush to network anyway.

- Import PostgreSQL 9.6's SQL parser. (Bo Peng)

This allows Pgpool-II to fully understand the newly added SQL syntax such as `COPY INSERT RETURNING`.

- In some cases `pg_terminate_backend()` now does not trigger a failover. (Muhammad Usama)

Because PostgreSQL returns exactly the same error code as postmaster down case and `pg_terminate_backend()` case, using `pg_terminate_backend()` raises a failover which user might not want. To fix this, now Pgpool-II finds a pid of backend which is the target of `pg_terminate_backend()` and does not trigger failover if so.

This functions is limited to the case of simple protocol and the pid is given to `pg_terminate_backend()` as a constant. So if you call `pg_terminate_backend()` via extended protocol (e.g. Java) still `pg_terminate_backend()` triggers a failover.

- HTML documents are now generated from SGML documents. (Muhammad Usama, Tatsuo Ishii, Bo Peng)

It is intended to have better construction, contents and maintainability. Also man pages are now generated from SGML. However, still there's tremendous room to enhance the SGML documents. Please help us!

A.50.3. Other Enhancements

- Make authentication error message more user friendly. (Tatsuo Ishii)

When attempt to connect to backend (including health checking), emit error messages from backend something like "sorry, too many clients already" instead of "invalid authentication message response type, Expecting 'R' and received '%c'"

- Tighten up health check timer expired condition in `pool_check_fd()`. (Muhammad Usama)

- Add new script called "watchdog_setup". (Tatsuo Ishii)

[watchdog_setup](#) is a command to create a temporary installation of Pgpool-II clusters with watchdog for mainly testings.

- Add "-pg" option to `pgpool_setup`. (Tatsuo Ishii)

This is useful when you want to assign specific port numbers to PostgreSQL while using [pgpool_setup](#). Also now `pgpool_setup` is installed in the standard bin directory which is same as `pgpool`.

- Add "replication delay" column to "show pool_nodes". (Tatsuo Ishii)

This column shows the [replication delay](#) value in bytes if operated in streaming replication mode.

- Do not update status file if all backend nodes are in down status. (Chris Pacey, Tatsuo Ishii)

This commit tries to remove the data inconsistency in replication mode found in [\[pgpool-general: 3918\]](#) by not recording the status file when all backend nodes are in down status. This surprisingly simple but smart solution was provided by Chris Pacey.

- Allow to use multiple SSL cipher protocols. (Muhammad Usama)

By replacing `TLSv1_method()` with `SSLv23_method()` while initializing the SSL session, we can use more protocols than TLSv1 protocol.

- Allow to use arbitrary number of items in the `black_function_list/white_function_list`. (Muhammad Usama)

Previously there were fixed limits for those.

- Properly process empty queries (all comments). (Tatsuo Ishii)

`Pgpool-II` now recognizes an empty query consisted of all comments (for example `"/* DBD::Pg ping test v3.5.3 */"`) (note that no `;`) as an empty query.

Before such that query was recognized an error.

- Add some warning messages for `wd_authkey` hash calculation failure. (Yugo Nagata)

Sometimes `wd_authkey` calculation fails for some reason other than authkey mismatch. The additional messages make these distinguishable for each other.

A.50.4. Changes

- Fix the broken `log_destination = syslog` functionality. (Muhammad Usama)

Fixing the logging to the syslog destination, which got broken by the `PGPOOL SET/SHOW` command commit, and also enhancing the `log_destination` configuration parameter to be assigned with the comma separated list of multiple destinations for the `Pgpool-II` log. Now, after this commit `log_destination` can be set to any combination of `'syslog'` and `'stderr'` log destinations.

- Change the default value of `search_primary_node_timeout` from 10 to 300. (Tatsuo Ishii)

Prior default value 10 seconds is sometimes too short for a standby to be promoted.

- Change the `Makefile` under directory `src/sql/`, that is proposed by [\[pgpool-hackers: 1611\]](#). (Bo Peng)
- Change the PID length of `pcp_proc_count` command output to 6 characters long. (Bo Peng)

If the `Pgpool-II` process ID are over 5 characters, the 6th character of each process ID will be removed. This commit changes the process ID length of `pcp_proc_count` command output to 6 characters long.

- Redirect all user queries to primary server. (Tatsuo Ishii)

Up to now some user queries are sent to other than the primary server even if `load_balance_mode = off`. This commit changes the behavior: if `load_balance_mode = off` in streaming replication mode, now all the user queries are sent to the primary server only.

A.50.5. Bug fixes

- Fixing a potential crash in `pool_stream` functions. (Muhammad Usama)

`POOL_CONNECTION->con_info` should be checked for null value before de-referencing when read or write fails on backend socket.

- Fixing the design of failover command propagation on watchdog cluster. (Muhammad Usama)

Overhauling the design of how failover, failback and promote node commands are propagated to the watchdog nodes. Previously the watchdog on `pgpool-II` node that needs to perform the node command (failover, failback or promote node) used to broadcast the failover command to all attached `pgpool-II`

nodes. And this sometimes makes the synchronization issues, especially when the watchdog cluster contains a large number of nodes and consequently the failover command sometimes gets executed by more than one `Pgpool-II`.

Now with this commit all the node commands are forwarded to the master/coordinator watchdog, which in turn propagates to all standby nodes. Apart from above the commit also changes the failover command interlocking mechanism and now only the master/coordinator node can become the lock holder so the failover commands will only get executed on the master/coordinator node.

- Fix the case when all backends are down then 1 node attached. (Tatsuo Ishii)

When all backends are down, no connection is accepted. Then 1 `PostgreSQL` becomes up, and attach the node using `pcp_attach_node`. It successfully finishes. However, when a new connection arrives, still the connection is refused because `Pgpool-II` child process looks into the cached status, in which the recovered node is still in down status if mode is streaming replication mode (native replication and other modes are fine). Solution is, if all nodes are down, force to restart all `pgpool` child.

- Fix for avoiding downtime when `Pgpool-II` changes require a restart. (Muhammad Usama)

To fix this, the verification mechanism of configuration parameter values is reversed, previously the standby nodes used to verify their parameter values against the respective values on the master `Pgpool-II` node and when the inconsistency was found the FATAL error was thrown, now with this commit the verification responsibility is delegated to the master `Pgpool-II` node. Now the master node will verify the configuration parameter values of each joining standby node against its local values and will produce a WARNING message instead of an error in case of a difference. This way the nodes having the different configurations will also be allowed to join the watchdog cluster and the user has to manually look out for the configuration inconsistency warnings in the master `Pgpool-II` log to avoid the surprises at the time of `Pgpool-II` master switch over.

- Fix a problem with the watchdog [failover_command](#) locking mechanism. (Muhammad Usama)

- Add compiler flag "-fno-strict-aliasing" in `configure.ac` to fix compiler error. (Tatsuo Ishii)

- Do not use `random()` while generating MD5 salt. (Tatsuo Ishii)

`random()` should not be used in security related applications. To replace `random()`, import `PostmasterRandom()` from `PostgreSQL`. Also store current time at the start up of `Pgpool-II` main process for later use.

- Don't ignore sync message from frontend when query cache is enabled. (Tatsuo Ishii)
- Fix bug that `Pgpool-II` fails to start if [listen_addresses](#) is empty string. (bug 237) (Muhammad Usama)

The socket descriptor array (`fds[]`) was not getting the array end marker when TCP listen addresses are not used.

- Create regression log directory if it does not exist yet. (Tatsuo Ishii)
- Fixing the error messages when the socket operation fails. (Muhammad Usama)
- Update regression test 003.failover to reflect the changes made to [show_pool_nodes](#). (Tatsuo Ishii)
- Fix hang when portal suspend received. (bug 230) (Tatsuo Ishii)

- Fix `pgpool` doesn't de-escalate IP in case network restored. (bug 228) (Muhammad Usama)

`set_state` function is made to de-escalate, when it is changing the local node's state from the coordinator state to some other state.

- SIGUSR1 signal handler should be installed before watchdog initialization. (Muhammad Usama)

Since there can be a case where a failover request from other watchdog nodes arrive at the same time when the watchdog has just been initialized, and if we wait any longer to install a SIGUSR1 signal handler, it can result in a potential crash

- Fix `Pgpool-II` doesn't escalate ip in case of another node unavailability. (bug 215) (Muhammad Usama)

The heartbeat receiver fails to identify the heartbeat sender watchdog node when the heartbeat destination is specified in terms of an IP address while `wd_hostname` is configured as a hostname string or vice versa.

- Fixing a coding mistake in watchdog code. (Muhammad Usama)

`wd_issue_failover_lock_command()` function is supposed to forward command type passed in as an argument to the `wd_send_failover_sync_command()` function instead it was passing the `NODE_FAILBACK_CMD` command type.

The commit also contains some log message enhancements.

- Display human readable output for backend node status. (Muhammad Usama)

Changed the output of `pcp_node_info` utility and show commands display human readable backend status string instead of internal status code.

- Replace "MAJOR" macro to prevent occasional failure. (Tatsuo Ishii)

The macro calls `pool_virtual_master_db_node_id()` and then access `backend->slots[id]->con` using the node id returned. In rare cases, it could point to 0 (in case when the DB node is not connected), which gives access to `con->major`, then it causes a segfault.

- Fix "kind mismatch" error message in Pgpool-II. (Muhammad Usama)

Many of "kind mismatch..." errors are caused by notice/warning messages produced by one or more of the DB nodes. In this case now Pgpool-II forwards the messages to frontend, rather than throwing the "kind mismatch..." error. This would reduce the chance of "kind mismatch..." errors.

- Fix handling of `pcp_listen_addresses` config parameter. (Muhammad Usama)

- Save and restore `errno` in each signal handler. (Tatsuo Ishii)

- Fix usage of `wait(2)` in pgpool main process. (Tatsuo Ishii)

The usage of `wait(2)` in Pgpool-II main could cause infinite wait in the system call. Solution is, to use `waitpid(2)` instead of `wait(2)`.

- Fix that `pool_read()` does not emit error messages when `read(2)` returns -1 if `failover_on_backend_error` is off. (Tatsuo Ishii)

- Fix buffer over run problem in "show pool_nodes". (Tatsuo Ishii)

While processing "show pool_nodes", the buffer for hostname was too short. It should be same size as the buffer used for `pgpool.conf`. Problem reported by a twitter user who is using pgpool on AWS (which could have very long hostname).

- Fix [\[pgpool-hackers: 1638\]](#) pgpool-II does not use default configuration. (Muhammad Usama)

Configuration file not found should just throw a WARNING message instead of ERROR or FATAL.

- Fix bug with load balance node id info on shmem. (Tatsuo Ishii)

There are few places where the load balance node was mistakenly put on wrong place. It should be placed on:

```
ConnectionInfo *con_info[child id, connection pool_id, backend id].load_balancing_node].
```

In fact it was placed on:

```
*con_info[child id, connection pool_id, 0].load_balancing_node].
```

As long as the backend id in question is 0, it is ok. However while testing Pgpool-II 3.6's enhancement regarding failover, if primary node is 1 (which is the load balance node) and standby is 0, a client connecting to node 1 is disconnected when failover happens on node 0. This is unexpected and the bug was revealed.

It seems the bug was there since long time ago but it had not found until today by the reason above.

- Fix for bug that pgpool hangs connections to database. (bug 197) (Muhammad Usama)

The client connection was getting stuck when backend node and remote Pgpool-II node becomes unavailable at the same time. The reason was a missing command timeout handling in the function that sends the IPC commands to watchdog.

- Fix a possible hang during health checking. (bug 204) (Yugo Nagata)

Health checking was hang when any data wasn't sent from backend after `connect(2)` succeeded. To fix this, `pool_check_fd()` returns 1 when `select(2)` exits with EINTR due to SIGALRM while health checking is performed.

- Deal with the case when the primary is not node 0 in streaming replication mode. (Tatsuo Ishii)

<http://www.pgpool.net/mantisbt/view.php?id=194#c837> reported that if primary is not node 0, then statement timeout could occur even after bug194-3.3.diff was applied. After some investigation, it appeared that MASTER macro could return other than primary or load balance node, which was not supposed to happen, thus `do_query()` sends queries to wrong node (this is not clear from the report but I confirmed it in my investigation).

`pool_virtual_master_db_node_id()`, which is called in MASTER macro returns `query_context->virtual_master_node_id` if query context exists. This could return wrong node if the variable has not been set yet. To fix this, the function is modified: if the variable is not either load balance node or primary node, the primary node id is returned.

- If statement timeout is enabled on backend and `do_query()` sends a query to primary node, and all of following user queries are sent to standby, it is possible that the next command, for example END, could cause a statement timeout error on the primary, and a kind mismatch error on pgpool-II is raised. (bug 194) (Tatsuo Ishii)

This fix tries to mitigate the problem by sending sync message instead of flush message in `do_query()`, expecting that the sync message reset the statement timeout timer if we are in an explicit transaction. We cannot use this technique for implicit transaction case, because the sync message removes the unnamed portal if there's any.

Plus, `pg_stat_statement` will no longer show the query issued by `do_query()` as "running".

Plus, `pg_stat_statement` will no longer show the query issued by `do_query()` as "running".

- Fix extended protocol handling in raw mode. (Tatsuo Ishii)

Bug152 reveals that extended protocol handling in raw mode (actually other than in stream mode) was wrong in `Describe()` and `Close()`. Unlike stream mode, they should wait for backend response.

- Fix confusing comments in `pgpool.conf`. (Tatsuo Ishii)
- Fix Japanese and Chinese documentation bug about raw mode. (Yugo Nagata, Bo Peng)

Connection pool is available in raw mode.

- Fix `is_set_transaction_serializable()` when `SET default_transaction_isolation TO 'serializable'`. (bug 191) (Bo Peng)

`SET default_transaction_isolation TO 'serializable'` is sent to not only primary but also to standby server in streaming replication mode, and this causes an error. Fix is, in streaming replication mode, `SET default_transaction_isolation TO 'serializable'` is sent only to the primary server.

- Fix extended protocol hang with empty query. (bug 190) (Tatsuo Ishii)

The fixes related to extended protocol cases in 3.5.1 broke the case of empty query. In this case backend replies with "empty query response" which is same meaning as a command complete message. Problem is, when empty query response is received, pgpool does not reset the query in progress flag thus keeps on waiting for backend. However, backend will not send the ready for query message until it receives a sync message. Fix is, resetting the in progress flag after receiving the empty query response and reads from frontend expecting it sends a sync message.

- Fix for [\[pgpool-general: 4569\]](#) segfault during `trusted_servers` check. (Muhammad Usama)

PostgreSQL's memory and exception manager APIs adopted by the Pgpool-II 3.4 are not thread safe and are causing the segmentation fault in the watchdog lifecheck process, as it uses the threads to ping configured trusted hosts for checking the upstream connections. Fix is to remove threads and use the

child process approach instead.

- Validating the PCP packet length. (Muhammad Usama)

Without the validation check, a malformed PCP packet can crash the PCP child and/or can run the server out of memory by sending the packet with a very large data size.

- Fix [pgpool_setup](#) to not confuse log output. (Tatsuo Ishii)

Before it simply redirects the stdout and stderr of pgpool process to a log file. This could cause log contents being garbled or even missed because of race condition caused by multiple process being writing concurrently. I and Usama found this while investigating the regression failure of 004.watchdog. To fix this, [pgpool_setup](#) now generates startall script so that pgpool now sends stdout/stderr to cat command and cat writes to the log file (It seems the race condition does not occur when writing to a pipe).

- Fix for [\[pgpool-general: 4519\]](#) Worker Processes Exit and Are Not Re-spawned. (Muhammad Usama)

The problem was due to a logical mistake in the code for checking the exiting child process type when the watchdog is enabled. I have also changed the severity of the message from FATAL to LOG, emitted for child exits due to max connection reached.

- Fix pgpool hung after receiving error state from backend. (bug #169) (Tatsuo Ishii)

This could happen if we execute an extended protocol query and it fails.

- Fix query stack problems in extended protocol case. (bug 167, 168) (Tatsuo Ishii)

- Fix [\[pgpool-hackers: 1440\]](#) yet another reset query stuck problem. (Tatsuo Ishii)

After receiving X message from frontend, if Pgpool-II detects EOF on the connection before sending reset query, Pgpool-II could wait for backend which had not received the reset query. To fix this, if EOF received, treat this as FRONTEND_ERROR, rather than ERROR.

- Fix for [\[pgpool-general: 4265\]](#) another reset query stuck problem. (Muhammad Usama)

The solution is to report FRONTEND_ERROR instead of simple ERROR when pool_flush on front-end socket fails.

- Fixing pgpool-recovery module compilation issue with PostgreSQL 9.6. (Muhammad Usama)

Incorporating the change of function signature for GetConfigOption() functions in PostgreSQL 9.6

- Fix compile issue on FreeBSD. (Muhammad Usama)

Add missing include files. The patch is contributed by the bug reporter and enhanced a little by me.

- Fix regression test to check timeout of each test. (Yugo Nagata)

- Add some warning messages for [wd_authkey](#) hash calculation failure. (Yugo Nagata)

Sometimes [wd_authkey](#) calculation fails for some reason other than authkey mismatch. The additional messages make these distinguishable for each other.

A.51. Release 3.5.25

Release Date: 2020-05-21

A.51.1. Bug fixes

- Fix build error on some system (Fedora 32). (Tatsuo Ishii)

- Fix unnecessary checks. (Tatsuo Ishii)
Patch contributed by sherlockcpp. Discussion: [\[pgpool-general: 7004\]](#)
 - Fix watchdog ping probes fail with long hostnames due to small buffer. ([bug 516](#)) (Bo Peng)
-

A.52. Release 3.5.24

Release Date: 2020-02-20

A.52.1. Changes

- Check if socket file exists at startup and remove them if PID file doesn't exist to avoid bind() failure. (Bo Peng)
-

A.52.2. Bug fixes

- Fix incorrect query rewriting in native replication mode. ([bug 551](#)) (Bo Peng)
 - Fix `pgpool_setup` to support PostgreSQL 12. (Tatsuo Ishii)
 - Fix occasional regression test failure. (Tatsuo Ishii)
 - Fix replication delay worker segfault when `application_name` in `primary_conninfo` is an empty string. ([bug 565](#)) (Tatsuo Ishii)
 - Fix the bug with ignored syslog setting. (Bo Peng)
-

A.53. Release 3.5.23

Release Date: 2019-10-31

A.53.1. Bug fixes

- Fix incorrect query rewrite in replication mode. (Bo Peng)
 - Fix assorted ancient v2 protocol bugs. (Tatsuo Ishii)
 - Fix problem that `syslog_facility` doesn't change by reload. ([bug 548](#)) (Takuma Hoshiai)
 - Fix Pgpool-II shutdown failed in certain case. (Tatsuo Ishii)
 - Fix segfault when executing an erroneous query after DEALLOCATE a named statement. ([bug 546](#)) (Tatsuo Ishii)
 - Add "-l" option to `arping_cmd` command default setting. (Bo Peng)
-

A.54. Release 3.5.22

Release Date: 2019-08-15

A.54.1. Enhancements

- Import some of memory manager debug facilities from PostgreSQL. (Tatsuo Ishii)
- Use `pg_get_expr()` instead of `pg_attrdef.adsrc` to support for PostgreSQL 12. (Bo Peng)
- Enhance shutdown script of `pgpool_setup`. (Tatsuo Ishii)
 - Make `shutdownall` to wait for completion of shutdown of Pgpool-II.
 - If environment variable `CHECK_TIME_WAIT` is set to true, use `netstat` command to confirm usage of the TCP/IP port while executing shutdown script.
- Deal `pgpool_adm` extension with PostgreSQL 12. (Tatsuo Ishii)

A.54.2. Bug fixes

- Fix "unable to bind. cannot get parse message" error. ([bug 531](#)) (Tatsuo Ishii)
- Fix online-recovery is blocked after a child process exits abnormally with replication mode and watchdog. ([bug 483](#)) (Muhammad Usama)
- Fix `pgpool_setup` to produce correct follow master command. (Tatsuo Ishii)
- Fix query cache module so that it checks oid array's bound. (Tatsuo Ishii)
- Fix off-by-one error in query cache module. (Tatsuo Ishii)
- Fix segfault when query cache is enabled. ([bug 525](#)) (Tatsuo Ishii)
- Down grade LOG "checking zapping sent message ..." to DEBUG5. (Tatsuo Ishii)
Discussion: [\[pgpool-general: 6620\]](#)
- Fix segfault when `samenet` is specified in `pool_hba.conf`. (Tatsuo Ishii)
Discussion: [\[pgpool-general: 6601\]](#).
- Fix to deal with backslashes according to the config of `standard_conforming_strings` in native replication mode. ([bug 467](#)) (Bo Peng)
- Fix memory leaks. (Tatsuo Ishii)
- Make failover in progress check more aggressively to avoid potential segfault. (Tatsuo Ishii)

A.55. Release 3.5.21

Release Date: 2019-05-16

A.55.1. Enhancements

- Speed up failover when all of backends are down. (Tatsuo Ishii)

If all of the backend are in down status, immediately give up finding primary node regardless `search_primary_node_timeout` and promptly finish the failover process.

Discussion: [\[pgpool-hackers: 3321\]](#)

- `pgpool-recovery` extension and `pgpool_setup` is now ready for the next major release PostgreSQL 12. (Tatsuo Ishii)

A.55.2. Bug fixes

- Fix the wrong error message "ERROR: connection cache is full", when all backend nodes are down. ([bug 487](#)) (Bo Peng)

When all backend nodes are down, `Pgpool-II` throws an incorrect error message "ERROR: connection cache is full". Change the error message to "all backend nodes are down, `pgpool` requires at least one valid node".

- Avoid exit/fork storm of `pool_worker_child` process. (Tatsuo Ishii)

`pool_worker_child` issues query to get WAL position using `do_query()`, which could throws FATAL error. In this case `pool_worker_child` process exits and `Pgpool-II` parent immediately forks new process. This cycle indefinitely repeats and gives high load to the system. To avoid the exit/fork storm, sleep `sr_check_period`.

- Fix `black_function_list`'s broken default value. (Tatsuo Ishii)
- Fix "not enough space in buffer" error. ([bug 499](#)) (Tatsuo Ishii)

The error occurred while processing error message returned from backend and the cause is that the query string in question is too big. Problem is, the buffer is in fixed size (8192 bytes). Eliminate the fixed size buffer and use pallocated buffer instead. This also saves some memory copy work.

- Fix DROP DATABASE failure. (Tatsuo Ishii)
- Fix wrong variable in `read_status_file()` function. ([bug 493](#)) (Takuma Hoshiai)
- Abort session if failover/failback is ongoing to prevent potential segfault. ([bug 481](#), [bug 482](#)) (Tatsuo Ishii)
- Fix compiler warnings. (Tatsuo Ishii)
- Fix memory leak in "batch" mode in extended query. ([bug 468](#)) (Tatsuo Ishii)

A.56. Release 3.5.20

Release Date: 2019-03-29

A.56.1. Enhancements

- Add new configuration option `ssl_prefer_server_ciphers`. (Muhammad Usama)

Add the new setting `ssl_prefer_server_ciphers` to let users configure if they want client's or server's cipher order to take preference.

The default for this parameter is off, which prioritize the client's cipher order as usual. However this is just for keeping backward compatibility, and it is possible that a malicious client uses weak ciphers. For this reason we recommend to set this parameter to on at all times.

- Allow to set a client cipher list. (Tatsuo Ishii)

For this purpose new parameter [ssl_ciphers](#), which specifies the cipher list to be accepted by Pgpool-II, is added. This is already implemented in PostgreSQL and useful to enhance security when SSL is enabled.

A.56.2. Bug fixes

- Fix unnecessary `fsync()` to `pgpool_status` file. (Tatsuo Ishii)

Whenever new connections are created to PostgreSQL backend, `fsync()` was issued to `pgpool_status` file, which could generate excessive I/O in certain conditions. So reduce the chance of issuing `fsync()` so that it is issued only when backend status is changed.

Discussion: [\[pgpool-general: 6436\]](#)

A.57. Release 3.5.19

Release Date: 2019-02-21

A.57.1. Bug fixes

- Test: Fix old JDBC functions and typos in regression test `068.memqcache_bug`. (Takuma Hoshiai)
- Doc: Fix configuration change timing regarding [memory_cache_enabled](#). (Tatsuo Ishii)
- Fix online recovery failed due to [client_idle_limit_in_recovery](#) in certain cases. ([bug 431](#)) (Tatsuo Ishii)
- Fix corner case bug when `strip_quote()` handle a empty query string. ([bug 458](#)) (Tatsuo Ishii)
- Fix Pgpool child segfault if failover occurs when trying to establish a connection. (Tatsuo Ishii)

See [\[pgpool-hackers: 3214\]](#) for discussion.

- Fix Pgpool-II hang if a client sends a extended query message such as close after sync message but before next simple query. (Tatsuo Ishii)

Discussion: [\[pgpool-hackers: 3164\]](#)

- Fix Pgpool-II hang when `idle_in_transaction_session_timeout = on`. ([bug 448](#)) (Tatsuo Ishii)
-

A.58. Release 3.5.18

Release Date: 2018-11-22

A.58.1. Bug fixes

- Fix to sort startup packet's parameters sent by client. ([bug 444](#))(Takuma Hoshiai)

If order of startup packet's parameters differ between cached connection pools and connection request, didn't use connection pool ,and created new connection pool.

- Fix segmentation fault occurs when a certain Bind message is sent in native replication mode. ([bug](#)

[443](#)(Bo Peng)

If the number of parameter format codes is specified to one, but the number of the original query's parameter is zero, `bind_rewrite_timestamp()` will call `memcpy` with a negative value. This causes segmentation fault.

Patch is provided by Yugo Nagata.

- Fix a query passed to `relcache` so that it uses schema qualified table name. (Tatsuo Ishii)
- Fix query cache invalidation bug. (Tatsuo Ishii)
- Fix segfault in extended query + query cache case. (Tatsuo Ishii)
- Fix memory leak in extended query + query cache enabled. (Tatsuo Ishii)

A.59. Release 3.5.17

Release Date: 2018-10-31

A.59.1. Changes

- Allow `PCP[attach/detach/promote]` commands during failover. (Muhammad Usama)
- Change `pgpool.spec` file to install extension to DB server which supports LLVM JIT. (Bo Peng)
- Test: Add regression test for SSL connection. (Tatsuo Ishii)
- Test: Add definition of PGLIB in `regress.sh`. (Bo Peng)

Patch provided by Jesper Pedersen.

A.59.2. Bug fixes

- Fix typo in `child_max_connections` description of `SHOW POOL_STATUS` output. (Tatsuo Ishii)

Patch provided by Phil Ramirez.

- Fix segmentation fault when error query and Sync message are sent in native replication mode. ([bug 434](#)) (Takuma Hoshiai)

In native replication mode, segmentation fault occurs when Sync messages is sent just after a query error.

- Fix syntax error when queries including time functions and `IN (SELECT ...)` in `WHERE` clause in native replication mode. ([bug 433](#)) (Bo Peng)

In native replication mode, queries including time functions (e.g. `now()`, `CURRENT_TIMESTAMP` etc.) are rewritten to a timestamp constant value. However, `Pgpool-II` doesn't support queries including time functions and `IN (SELECT ...)` in `WHERE` clause.

- Fix occasional less data returned to frontend with extended protocol. ([bug432](#)) (Tatsuo Ishii)

The idea for fix is, use pending message data list. It records messages from frontend, and it is expected that we will receive same number of messages.

Initial patch is created by Yugo Nagata and fixed by Tatsuo Ishii.

- Fix memory leak in `trigger_failover_command()`. (Tatsuo Ishii)

- Fix memory leak when query cache enabled in streaming replication mode + extended query case. (Tatsuo Ishii)
- Fix memory leak when `memory_cache_enabled = on` and write SQLs are sent. (Bo Peng)

In a explicit transaction, the `SELECT` results are cached in temporary buffer. If a write `SQL` is sent which modifies the table, the temporary buffer should be reset.
- Fix occasional failure in regression 065.bug152. (Tatsuo Ishii)
- Test: Add `EXECUTE/DEALLOCATE` regression test. (Takuma Hoshiai)
- Add missing `pgpool_recovery--1.0--1.1.sql` file to update `pgpool_recovery()` function version to 1.1. (Bo Peng)
- Fix kind mismatch error when `DEALLOCATE` statement is issued. (Bo Peng)

`PREPARE` should be add to `pool_add_sent_message`, so that `EXECUTE` and `DEALLOCATE` can be sent to the same node as `PREPARE`.

See [\[pgpool-general: 6226\]](#) for more details.

- Do not update `pool_passwd` if the password length is incorrect. ([bug 419](#)) (Takuma Hoshiai, Tatsuo Ishii)

For Pgpool-II 3.7 or before, the password stored in `pool_passwd` is MD5 password only. So check the correctness of `pool_passwd` by scanning entire file.
- Test: Update `clean.sh` which clean up regression test results. (Bo Peng)

Patch provided by Jesper Pedersen.
- Add `.gitignore` files. (Bo Peng)

Patch provided by Jesper Pedersen.
- Fix segfault when node 0 is in down status in case of both health check and [failover_on_backend_error](#) are disabled. (Tatsuo Ishii)

A.60. Release 3.5.16

Release Date: 2018-07-31

A.60.1. Bug fixes

- Fix "write on backend 0 failed with error : "Success"" error. ([bug 403](#)) (Tatsuo Ishii)

Don't treated it as an error if `write()` returns 0.
- Fix memory leaks related to `pool_extract_error_message()`. (Tatsuo Ishii)
- Fix an incorrect declare as `bool`, rather than `int` in `pool_extract_error_message()`. (Tatsuo Ishii)

This led to a segfault issue mentioned on certain platform.
- Fix segfault in `per_node_error_log()` on `armhf` architecture. (Tatsuo Ishii)

Patch provided by Christian Ehrhardt.
- Test: Fix 006.memqcache test failure. (Tatsuo Ishii)

A.61. Release 3.5.15

Release Date: 2018-06-12

A.61.1. Bug fixes

- Fix Pgpool-II hung if replication delay is too much, when query cache enabled in extended query mode. (Tatsuo Ishii)

See [\[pgpool-general-jp: 1534\]](#) for more details.

- In extended query mode, do not set writing tx flag with SET TRANSACTION READ ONLY. (Tatsuo Ishii)
- Prevent [pcp_recovery_node](#) from recovering "unused" status node. (Tatsuo Ishii)

This allowed to try to recovery a node without configuration data, which leads to variety of problems.

See [\[pgpool-general: 5963\]](#) for more details.

Also I fixed `pgpool_recovery` function so that it quotes an empty string argument with double quotes.

A.62. Release 3.5.14

Release Date: 2018-04-17

A.62.1. Bug fixes

- Test: Add new regression test for node 0 is down. (Tatsuo Ishii)
 - Make calls to `to_regclass` fully schema qualified. (Tatsuo Ishii)
 - Test: Improve test script 003.failover. (Bo Peng)
 - Deal with "unable to bind D cannot get parse message "S1" error. (Tatsuo Ishii)
 - Fix `pgpool_setup` failure in replication mode. (Tatsuo Ishii)
 - Allow to support `pgpool_switch_xlog` PostgreSQL 10. (Tatsuo Ishii)
-

A.63. Release 3.5.13

Release Date: 2018-02-13

Note: This release fixed the bug with socket writing added in Pgpool-II 3.7.0, 3.6.6 and 3.5.10. Due to this bug, when the network load is high, an illegal message may be sent to the frontend or backend. All users using 3.7.x, 3.6.6 or later, 3.5.10 or later versions of Pgpool-II should update as soon as possible.

A.63.1. Changes

- Allow to build with `libressl`. (Tatsuo Ishii)
See [\[pgpool-hackers: 2714\]](#) for more details. Patch by Sandino Araico Sanchez.
- Set `TCP_NODELAY` and non blocking to frontend socket. (Tatsuo Ishii)
`TCP_NODELAY` is employed by PostgreSQL, so do we it.
- Change systemd service file to use `STOP_OPTS="-m fast"`. (Bo Peng)
- Change `pgpool_setup` to add `restore_command` in `recovery.conf`. (Bo Peng)

A.63.2. Bug fixes

- Fix writing transaction flag is accidentally set at commit or rollback. (Tatsuo Ishii)
- Fix bug with socket writing. (Tatsuo Ishii)
`pool_write_flush()` is responsible for writing to sockets when `pgpool`'s write buffer is full (this function was introduced in 3.6.6 etc). When network write buffer in kernel is full, it does retrying but it forgot to update the internal buffer pointer. As a result, broken data is written to the socket. This results in variety of problems including too large message length.
- Fix segfault when `%a` is in `log_line_prefix` and debug message is on. ([bug 376](#)) (Tatsuo Ishii)
- Fix queries hanging in `parse_before_bind` with extended protocol and replication + load-balancing. ([bug 377](#)) (Tatsuo Ishii)

A.64. Release 3.5.12

Release Date: 2018-01-09

A.64.1. Bug fixes

- Replace `/bin/ed` with `/bin/sed` in `pgpool_setup`, because `/bin/sed` is included in most distribution's base packages. (Tatsuo Ishii)
- Change the `pgpool.service` and `sysconfig` files to output `Pgpool-II` log. (Bo Peng)
Removing "Type=forking" and add `OPTS="-n"` to run `Pgpool-II` with non-daemon mode, because we need to redirect logs. Using "journalctl" command to see `Pgpool-II` systemd log.
- Fix timestamp data inconsistency by replication mode. (Bo Peng)
From PostgreSQL10 the column default value such as 'CURRENT_DATE' changes, `Pgpool-II` didn't rewrite timestamp by the added default values. This caused data inconsistency.
- Downgrade a log message to debug message. (Tatsuo Ishii)
That was mistaken left while last development cycle.
- Fix for re-sync logic in reading packet from backend. (Tatsuo Ishii)
`read_kind_from_backend()`, which reads message kind from backend, re-syncs backend nodes when a ready for query message is received. Unfortunately it forgot to call `pool_pending_message_pull_out()` to delete sync pending message. This leads to random stuck while reading packets from backend. Fix this to call

pool_pending_message_pull_out().

- Fix Pgpool-II hangs. ([bug 370](#)) (Tatsuo Ishii)

If an erroneous query is sent to primary and without a sync message the next query that requires a catalog cache look up is send, Pgpool-II hangs in do_query().

- Add SL_MODE macro for upper compatibility with Pgpool-II 3.7 or later. (Tatsuo Ishii)
- Fix returning transaction state when "ready for query" message received. (Tatsuo Ishii)

We return primary or master node state of ready for query message to frontend. In most cases this is good. However if other than primary node or master node returns an error state (this could happen if load balance node is other than primary or master node and the query is an erroneous SELECT), this should be returned to frontend, because the frontend already received an error.

- Fix pgpool start message printed multiple times. (Tatsuo Ishii)
- Add an execute permission bit to the start/stop script in [watchdog_setup](#). (Tatsuo Ishii)

A.65. Release 3.5.11

Release Date: 2017-11-01

A.65.1. Bug fixes

- Add different pgpool.sysconfig file for RHEL6 and RHEL7. ([bug 343](#)) (Bo Peng)

In RHEL6, the "-n" option is needed to redirect log.

- Fix for bug in watchdog where sometime failover is not reliably performed. (Muhammad Usama)

Currently watchdog process only considers the node's watchdog state before deciding if it can handle the failover and failover-locking requests.

But the problem with this technique is that, for the instance when the node has announced itself as a master/coordinator of the cluster and is waiting for the standby nodes to recognise it as a Master node. For that period of time the watchdog state of the node is Master/coordinator, but it is yet not fully capable of handling the failover and failover-locking requests.

So sometimes this leads to a situation where a failover is not reliably performed in case when the failover request arrives while watchdog cluster is in the process of electing a leader.

The fix for the above situation is to make sure that the node has fully acquired the Master status before accepting the failover and failover-locking requests.

- Fix bug with handling of 'H' (flush) message in streaming replication mode. ([bug 345](#)) (Tatsuo Ishii)

If user expects to read response right after 'H', Pgpool-II hangs. The cause was, when 'H' received, extended query mode was reset and pending message was not used.

- Doc: Fix `pcp_node_info` documents. (Tatsuo Ishii)
- Fix bug mistakenly overriding global backend status right after failover. (Tatsuo Ishii)

See [\[pgpool-general: 5728\]](#) for more details.

- Fix exit signal handlers to not call ereport. (Tatsuo Ishii)

See [\[pgpool-hackers: 2545\]](#) for more details.

- Deal with OpenSSL 1.1. (Tatsuo Ishii, Muhammad Usama)

A.66. Release 3.5.10

Release Date: 2017-09-05

A.66.1. Bug fixes

- Fix Pgpool-II hanging when error occurs in streaming replication mode and extended query. (Tatsuo Ishii)
If backend returns ERROR, Pgpool-II reads message from frontend until a `sync` message is sent. Previous code assumed next message is `sync`. However it is possible that more message coming before the `sync` message, it's a low probability though. Fix it to continue reading messages until the `sync` message is read.
- Fix `wd_authkey` bug in that a request to add new node to the cluster is rejected by master. (Yugo Nagata)
This is a bug due to the implementation of 3.5.6 and 3.6.3.
This changed the definition of `tv_sec` that is used to check `wd_authkey` so that this was affected by the clock of OS. So, if there is a lag between two nodes' clocks, the `wd_authkey` check fails.
- Test: Fix load balance test driver. (Tatsuo Ishii)
Some tests only for native replication mode was executed in streaming replication mode as well.
- Fix not working `slony` mode in extended query. (Tatsuo Ishii)
When response returned from backend, in progress flag was not reset in `slony` mode, which cause waiting for next message from backend in vain.
- Fix ancient bug of stream write modules. (Tatsuo Ishii)
Fix bug with `pool_write_noerror()` when requested length exceeds remaining write buffer size. This could lead to a buffer overrun problem.
When write buffer is full, `pool_flush_it()` is called, which could write data to socket in the middle of message. To fix the problem directly write requested data if the write buffer is going to be full.
Enhance performance of `pool_unread()`.
- Test: Some miscellaneous small fixes in regression test scripts. (Muhammad Usama)
- Doc: Fix documentation about load-balancing. (Yugo Nagata)
- Fix core dump and mishandling of temp tables. (Tatsuo Ishii)
- Fix ancient bug of `pool_unread()`. (Tatsuo Ishii)
When `realloc()` is called in `pool_unread()`, it did not update the buffer size. This could cause variety of memory corruption and unexpected data reading from backend. The reason why we did not found that is, probably recently Pgpool-II starts extensively to use `pool_unread()`.
- Fix handling of empty queries. ([bug 328](#)) (Tatsuo Ishii)
When empty query (empty string or all comment query) is sent, command complete message was returned to frontend. This is not correct. An empty query response should be returned to frontend.
- Fix query cache bug with streaming replication mode and extended query case. (Tatsuo Ishii)
- Fix memory leak with streaming replication mode/extended query case. ([bug 324](#)) (Tatsuo Ishii)
- Test: Fix Java program in 005.regression test. (Tatsuo Ishii)

- Fix for when failover is triggered by worker process, it is possible that wrong DB node could failover. ([bug 303](#)) (Tatsuo Ishii)

This is due to the `db_node_id` member in the `POLL_CONNECTION` structure is not initialized in the process (in child process the member is properly initialized). To solve the problem, add new function `pool_set_db_node_id()` to set the structure member variable and call it inside `make_persistent_db_connection()`.

- Fix starting unnecessary transaction when `SET` command is issued. (Tatsuo Ishii)
- Fix for [\[pgpool-general: 5621\]](#) `Failover()` function should be executed with health check alarm disabled. (Muhammad Usama)
- Fix `Pgpool-II` hung up bug or other errors in error case in extended query in replication mode. (Tatsuo Ishii)

And other fixes in this commit.

1) Do not send intended error query to backend in streaming replication mode in `ErrorResponse3()`.

2) Fix `pool_virtual_master_db_node_id()` to return the `virtual_master_node_id` only when query is in progress and query context exists.

- Allow `make dist` to include `pgpool.service`. (Yugo Nagata)

A.66.2. Enhancements

- Test: Redirect build log to a log file. (Tatsuo Ishii)
- Test: Add more memory leak regression tests. (Tatsuo Ishii)

A.67. Release 3.5.9

Release Date: 2017-07-11

A.67.1. Bug fixes

- Fix for [\[pgpool-hackers: 2400\]](#) Garbage output (Muhammad Usama)
Mostly the log messages fixes and few code cleanups.
- Importing the latest changes in the `MemoryManager API` from PostgreSQL code. (Muhammad Usama)
- Fixing [\[pgpool-hackers: 2390\]](#) Problems with the relative paths in daemon mode (Muhammad Usama)
- Adjust function name change in PostgreSQL 10 dev head. (Tatsuo Ishii)

```
pg_current_wal_location  -> pg_current_wal_lsn
pg_last_wal_replay_location -> pg_last_wal_replay_lsn
```

- Fix a possible hang with streaming replication and extended protocol (Yugo Nagata)

This hang occurred under a certain condition. The following is an example.

- pgpool.conf is configured so that all read queries are sent to the standby.
- First, issue a writing query in a transaction block
- After committing the transaction, issue a select query.
- When processing the query, send Describe (statement) message just after Parse.

Without using JDBC, we can reproduce the problem by pgproto with the following messages.

```
'Q' "DROP TABLE IF EXISTS test_tbl"
'Y'
'Q' "CREATE TABLE test_tbl(i int)"
'Y'
'Q' "INSERT INTO test_tbl VALUES(1)"
'Y'

'P' "" "BEGIN" 0
'B' "" "" 0 0 0
'E' "" 0
'S'
'Y'

'P' "" "INSERT INTO test_tbl VALUES(1)" 0
'B' "" "" 0 0 0
'E' "" 0
'S'
'Y'

'P' "" "COMMIT" 0
'B' "" "" 0 0 0
'E' "" 0
'S'
'Y'

'P' "S_1" "SELECT * FROM test_tbl" 0
'D' 'S' "S_1"
'B' "C_1" "S_1" 0 0 0
'E' "C_1" 0
'S'
'Y'

'X'
```

To fix it, `parse_before_bind()` should be called only if we are in a transaction block so that we can send `Bind` and `Execute` to the right backend.

- Fix Pgpool-II hang when used by erlang applications. (Tatsuo Ishii)

Erlang client sends "Describe" message followed by "Flush". So the backend returns "Row description." However Pgpool-II forgets to reset the query in progress flag upon receiving "Row description" message, then Pgpool-II keeps on waiting for response from backend. This is the cause of erlang client hanging.

Fix is, just reset the query in progress flag upon receiving "Row description" message. Same thing can be said to "no data" message.

See [\[pgpool-general: 5555\]](#) for more details.

- Fix bug with sending bind message to wrong target node. ([bug 314](#)) (Tatsuo Ishii)
- Fix query cache hang when used by node.js. (Tatsuo Ishii)

See [\[pgpool-general: 5511\]](#) for more details.

- Deal with PostgreSQL 10 in streaming replication delay checking. (Tatsuo Ishii)
- Fix query cache memory leak. (Tatsuo Ishii)

Clearing cache buffers in case of no oid queries (like `BEGIN`, `CHECKPOINT`, `VACUUM`, etc) should have been done, but it did not.

- Fix extended query hang in certain case. (Tatsuo Ishii)

erlang PostgreSQL API produces Parse ('P'), Describe ('D'), Flush ('H'), Bind ('B'), and Execute ('E'). Notice the 'H' message (this does not happen in JDBC. I suspect that's the reason why this problem is not popular before). After that, Pgpool-II dropped the extended query mode, it failed to find which backend to read data. Thus Pgpool-II simply tries to read all of backend which causes hang because it may have not send a message to some of backends.

Solution is, after receiving the flush message set doing extended query flag.

- Fix corner case bug in Pgpool-II starting up. (Tatsuo Ishii)

It is possible that a failover request is accepted before primary node is searched. This leads Pgpool-II to a strange state: there's no primary node if the failed node was a primary node (even if new primary node exists as a result of promotion of existing standby).

See [\[pgpool-hackers: 2321\]](#) for more details.

A.68. Release 3.5.8

Release Date: 2017-05-11

A.68.1. Bug fixes

- Add node 0 failover test. (Tatsuo Ishii)
- Fix Pgpool-II child process segfault reported in [\[pgpool-hackers: 2312\]](#). (Tatsuo Ishii)

A.69. Release 3.5.7

Release Date: 2017-04-28

A.69.1. Bug fixes

- Fixing a mistake in the watchdog code. (Muhammad Usama)
commit also adds some debug messages in the watchdog code.
- Fix for 0000299: Errors on the reloading of configuration. ([Bug 299](#)) (Muhammad Usama)
- Fix for 0000289: Inconsistent backend state. ([Bug 289](#)) (Muhammad Usama)
- Enhancing the handling of split-brain scenario by the watchdog. (Muhammad Usama)

Previously, the watchdog cluster was used to call for re-election of the master/coordinator node whenever the split-brain situation was detected. And consequently every node was required to rejoin the watchdog network, Which was essentially similar to the re-booting of the whole watchdog cluster.

The candidate for the master/coordinator node is selected on the following criteria.

1-- When two watchdog nodes are claiming to be the cluster master, the master node that has performed the escalation keeps the master status and the other node is asked to step down.

2-- If the conflict could not be resolved by the escalation status of the nodes, The node which holds the

quorum remains the master/coordinator.

3-- If the quorum status of both contenders is also same. The node with higher number of connected alive nodes get the preference.

4-- Finally, if all above three yields no winner, the older master (The node that has the coordinator status for longer duration) remains the master.

- Enhancing the watchdog internal command mechanism to handle multiple concurrent commands. (Muhammad Usama)
- Add bool encode and decode functions to JSON framework for code compatibility.(Muhammad Usama)
- Comment out unsupported Java method in new JDBC drivers to prevent regression failure. (Tatsuo Ishii)
- Downgrade parse before bind log message to debug1. (Tatsuo Ishii)
- Fix coverity warnings. (Tatsuo Ishii, Muhammad Usama)
- Fix for [\[pgpool-general: 5396\]](#) pam ldap failure. (Muhammad Usama)
- Consider SHOW command as kind of a read query. (Tatsuo Ishii)

In streaming replication mode, if SHOW is issued then subsequent SELECTs are sent to the primary node in an explicit transaction. This is not a reasonable and unnecessary limitation. Also fix hang when parse command returns error.

- Fix memory leak problem caused by commit adcb636. (Tatsuo Ishii)

Commit adcb636 introduces "pending message queue". When a message arrives, the info is added to the queue and a copy of object is created at the same time, but forgot to free the object. Fix is, creating a new function `pool_pending_message_free_pending_message()` and call it after `pool_pending_message_add()`, `pool_pending_message_get()` and `pool_pending_message_pull_out()`. Problem reported by Sergey Kim.

- Mega patch to fix "kind mismatch" (or derived) errors in streaming replication mode. ([Bug 271](#)) (Tatsuo Ishii)

The errors are caused by wrong prediction in which (or both) database node will send response to Pgpool-II. Previous implementation using "sync map" are weak and sometimes fail in the prediction.

This patch introduces new implementation using "pending message queue", which records all sent message to backends. The element of the queue stores info regarding messages types (parse/bind/execute/describe/close/sync), to which database node the message was sent and so on. It's a simple FIFO queue. When a message arrives from backend, by looking at the head of the "pending message queue", it is possible to reliably predict what kind of message and from which database node it will arrive. After receiving the message, the element is removed from the queue.

I would like to thank to Sergey Kim, who has been helping me in testing series of patches.

See [Bug 271](#) and discussion in pgpool-hackers mailing list [\[pgpool-hackers: 2043\]](#) and [\[pgpool-hackers: 2140\]](#) for more details.

- Fix for 0000296: Pgpool v3.6.2 terminated by systemd because the service Type has been set to 'forking'. ([Bug 296](#)) (Muhammad Usama)

A.70. Release 3.5.6

Release Date: 2017-03-17

A.70.1. Bug fixes

- Add "Wants=network.target" to pgpool.service file. ([bug 294](#)) (Bo Peng)
- Fix [pcp_promote_node](#) bug that fails promoting node 0. (Yugo Nagata)

The master node could not be promoted by `pcp_promote_node` with the following error;

```
FATAL: invalid pgpool mode for process recovery request
DETAIL: specified node is already primary node, can't promote node id 0
```

In streaming replication mode, there is a case that Pgpool-II regards the status of primary node as "standby" for some reasons, for example, when `pg_ctl promote` is executed manually during Pgpool-II is running, in which case, it seems to Pgpool-II that the primary node doesn't exist.

This status mismatch should be fixed by `pcp_promote_node`, but when the node is the master node (the first alive node), it fails as mentioned above.

The reason is as following. before changing the status, `pcp_promote_node` checks if the specified node is already primary or not by comparing the node id with `PRIMARY_NODE_ID`. However, if the primary doesn't exist from Pgpool-II's view, `PRIMARY_NODE_ID` is set to 0, which is same as `MASTER_NODE_ID`. Hence, when the master node is specified to be promoted, `pcp_promote_node` is confused that this node is already primary and doesn't have to be promoted, and it exits with the error.

To fix this, `pcp_promote_node` should check the node id by using `REAL_PRIMARY_NODE_ID`, which is set -1 when the primary doesn't exist, rather than `PRIMARY_NODE_ID`.

- Add the latest release note link to README file.(Bo Peng)
- Pgpool-II should not perform ping test after bringing down the VIP. (Muhammad Usama)

This issue was reported by the reporter of bug:[pgpool-II 0000249]: watchdog sometimes fails de-escalation

- Fix to release shared memory segments when Pgpool-II exits. ([bug 272](#)) (Tatsuo Ishii)
- Fix for [\[pgpool-general: 5315\]](#) `pg_terminate_backend` (Muhammad Usama)
- Adding the missing `ExecStop` and `ExecReload` commands to the `systemd` service configuration file. (Muhammad Usama)
- Fix for 281: "segmentation fault" when execute [pcp_attach_node](#). ([bug 281](#)) (Muhammad Usama)
- Fix load balancing bug in streaming replication mode. (Tatsuo Ishii)

In an explicit transaction, any `SELECT` will be load balanced until write query is sent. After writing query is sent, any `SELECT` should be sent to the primary node. However if a `SELECT` is sent before a `sync` message is sent, this does not work since the treatment of writing query is done after ready for query message arrives.

Solution is, the treatment for writing query is done in executing the writing query as well.

The bug has been there since V3.5.

- Fix yet another kind mismatch error in streaming replication mode. (Tatsuo Ishii)
- Fix `do_query()` hangs after close message. (Tatsuo Ishii)
- Fixing stack smashing detected. ([bug 280](#)) (Muhammad Usama)

It was a buffer overflow in `wd_get_cmd` function

- Fixing the issue with the watchdog process restart. (Muhammad Usama)

When the watchdog process gets abnormally terminated because of some problem (e.g. Segmentation fault) the new spawned watchdog process fails to start and produces an error "bind on ... failed with reason: Address already in use".

Reason is the abnormally terminating watchdog process never gets the time to clean-up the socket it uses for IPC and the new process gets an error because the socket address is already occupied.

Fix is, the Pgpool main process sets the flag in shared memory to mark the watchdog process was abnormally terminated and at startup when the watchdog process see that the flag is set, it performs the clean up of the socket file and also performs the de-escalation (If the watchdog process was crashed when it was master/coordinator node) if required before initializing itself.

- Fix query cache bug reported in [pgpool-general-jp:1441](#). (Tatsuo Ishii)

In streaming replication mode with query cache enabled, SELECT hangs in the following scenario:

- 1) a SELECT hits query cache and returns rows from the query cache.
- 2) following SELECT needs to search meta data and it hangs.

In #1, while returning the cached result, it misses to call `pool_unset_pending_response()`, which leave the `pending_response` flag be set. In #2, `do_query()` checks the flag and tries to read pending response from backend. Since there's no pending data in backend, it hangs in reading data from backend.

Fix is, just call `pool_unset_pending_response()` in #1 to reset the flag.

Bug report and fix provided by Nobuyuki Nagai. New regression test item (068) added by me.

- Remove `elog/ereport` calls from signal handlers. (Tatsuo Ishii)

See [\[pgpool-hackers: 1950\]](#) for details.

- Fix bug failed to create INET domain socket in FreeBSD if `listen_addresses = '*'`. ([bug 202](#)) (Bo Peng)
- Fix for 0000249: watchdog sometimes fails de-escalation. ([bug 249](#)) (Muhammad Usama)

The solution is to use the `waitpid()` system call without `WNOHANG` option.

- Fix `connection_life_time` broken by `authentication_timeout`. (Yugo Nagata)
- Fix authentication timeout that can occur right after client connections. (Yugo Nagata)

A.71. Release 3.5.5

Release Date: 2016-12-26

A.71.1. Bug fixes

- Tightening up the watchdog security. (Muhammad Usama)

Now `wd_authkey` uses the HMAC SHA-256 hashing.

- Add `pgpool_adm` extension in Pgpool-II RPM. (Bo Peng)
- Fix occasional segfault when query cache is enabled. ([bug 263](#)) (Tatsuo Ishii)
- Fix packet kind does not match error in extended protocol. ([bug 231](#)) (Tatsuo Ishii)

According to the [bug231](#), the bug seem to bite you if all of following conditions are met:

- Streaming replication mode
- Load balance node is not node 0
- Extended protocol is used
- SELECT is executed, the statement is closed, then a transaction command is executed

The sequence of how the problem bites is:

1. SELECT executes on statement S1 on the load balance node 1
2. Frontend send Close statement
3. Pgpool-II forward it to backend 1
4. Frontend sends Parse, Bind, Execute of COMMIT
5. Pgpool-II forward it to backend 0 & 1
6. Frontend sends sync message
7. Pgpool-II forward it to backend 0 & 1
8. Backend 0 replies back Parse complete ("1"), while backend 1 replies back close complete ("3") because of #3.
9. Kind mismatch occurs

The solution is, in #3, let Pgpool-II wait for response from backend 1, but do not read the response message. Later on Pgpool-II's state machine will read the response from it before the sync message is sent in #6. With this, backend 1 will reply back "1" in #8, and the kind mismatch error does not occur.

Also, fix not calling `pool_set_doing_extended_query_message()` when receives Close message. (I don't know why it was missed).

New regression test "067.bug231" was added.

- Fix a race condition in a signal handler per bug 265. (Tatsuo Ishii)

In `child.c` there's signal handler which calls `elog`. Since the signal handler is not blocked against other signals while processing, deadlock could occur in the system calls in the pgpool shutdown sequence. To fix the problem, now the signal handler is blocked by using `POOL_SETMASK`.

Ideally we should avoid calling `elog` in signal handlers though.

- Back porting the improved failover command propagation mechanism from Pgpool-II 3.6 (Muhammad Usama)

Overhauling the design of how failover, failback and promote node commands are propagated to the watchdog nodes. Previously the watchdog on pgpool-II node that needs to perform the node command (failover, failback or promote node) used to broadcast the failover command to all attached pgpool-II nodes. And this sometimes makes the synchronization issues, especially when the watchdog cluster contains a large number of nodes and consequently the failover command sometimes gets executed by more than one pgpool-II.

Now with this commit all the node commands are forwarded to the master/coordinator watchdog, which in turn propagates to all standby nodes. Apart from above the commit also changes the failover command interlocking mechanism and now only the master/coordinator node can become the lock holder so the failover commands will only get executed on the master/coordinator node.

- Do not cancel a query when the query resulted in an error other than in native replication mode. (Tatsuo Ishii)

It was intended to keep the consistency, but there's no point in other than native replication mode.

- Remove obsoleted option "-c" in pgpool command. (Tatsuo Ishii)

Also fix typo in the help message.

- Fix authentication failed error when PCP command is cancelled. (bug 252) (Muhammad Usama)
- Change the default value of `search_primary_node_timeout` from 10 to 300. (Tatsuo Ishii)

Prior default value 10 seconds is sometimes too short for a standby to be promoted.

- Fix the case when all backends are down then 1 node attached. (bug 248) (Tatsuo Ishii)

When all backends are down, no connection is accepted. Then 1 PostgreSQL becomes up, and attach the node using `pcp_attach_node`. It successfully finishes. However, when a new connection arrives, still the connection is refused because `pgpool` child process looks into the cached status, in which the recovered node is still in down status if mode is streaming replication mode (native replication and other modes are fine). Solution is, if all nodes are down, force to restart all `pgpool` child.

- Fix for: [pgpool-general: 4997] Avoiding downtime when `pgpool` changes require a restart (Muhammad Usama)

To fix this, The verification mechanism of configuration parameter values is reversed, previously the standby nodes used to verify their parameter values against the respective values on the master `pgpool-II` node and when the inconsistency was found the FATAL error was thrown, now with this commit the verification responsibility is delegated to the master `pgpool-II` node. Now the master node will verify the configuration parameter values of each joining standby node against its local values and will produce a WARNING message instead of an error in case of a difference. This way the nodes having the different configurations will also be allowed to join the watchdog cluster and the user has to manually look out for the configuration inconsistency warnings in the master `pgpool-II` log to avoid the surprises at the time of `pgpool-II` master switch over.

- Add compiler flag "-fno-strict-aliasing" in `configure.ac` to fix compiler error. (Tatsuo Ishii)
- Do not use `random()` while generating MD5 salt. (Tatsuo Ishii)

`random()` should not be used in security related applications. To replace `random()`, import `PostmasterRandom()` from PostgreSQL. Also store current time at the start up of `Pgpool-II` main process for later use.

- Don't ignore sync message from frontend when query cache is enabled. (Tatsuo Ishii)

A.72. Release 3.4.26

Release Date: 2019-10-31

A.72.1. Bug fixes

- Fix incorrect query rewrite in replication mode. (Bo Peng)
- Fix assorted ancient v2 protocol bugs. (Tatsuo Ishii)
- Fix problem that `syslog_facility` doesn't change by reload. ([bug 548](#)) (Takuma Hoshiai)
- Fix `Pgpool-II` shutdown failed in certain case. (Tatsuo Ishii)
- Fix segfault when executing an erroneous query after `DEALLOCATE` a named statement. ([bug 546](#)) (Tatsuo Ishii)

A.73. Release 3.4.25

Release Date: 2019-08-15

A.73.1. Enhancements

- Import some of memory manager debug facilities from PostgreSQL. (Tatsuo Ishii)

- Use `pg_get_expr()` instead of `pg_attrdef.adsrc` to support for PostgreSQL 12. (Bo Peng)
 - Enhance shutdown script of `pgpool_setup`. (Tatsuo Ishii)
 - Make `shutdownall` to wait for completion of shutdown of Pgpool-II.
 - If environment variable `CHECK_TIME_WAIT` is set to true, use `netstat` command to confirm usage of the TCP/IP port while executing shutdown script.
 - Deal `pgpool_adm` extension with PostgreSQL 12. (Tatsuo Ishii)
-

A.73.2. Bug fixes

- Fix `pgpool_setup` to produce correct follow master command. (Tatsuo Ishii)
 - Fix query cache module so that it checks oid array's bound. (Tatsuo Ishii)
 - Fix off-by-one error in query cache module. (Tatsuo Ishii)
 - Fix segfault when `samenet` is specified in `pool_hba.conf`. (Tatsuo Ishii)
- Discussion: [\[pgpool-general: 6601\]](#).
- Fix to deal with backslashes according to the config of `standard_conforming_strings` in native replication mode. ([bug 467](#)) (Bo Peng)
 - Fix memory leaks. (Tatsuo Ishii)
-

A.74. Release 3.4.24

Release Date: 2019-05-16

A.74.1. Enhancements

- Speed up failover when all of backends are down. (Tatsuo Ishii)
- If all of the backend are in down status, immediately give up finding primary node regardless `search_primary_node_timeout` and promptly finish the failover process.
- Discussion: [\[pgpool-hackers: 3321\]](#)
- `pgpool-recovery` extension and `pgpool_setup` is now ready for the next major release PostgreSQL 12. (Tatsuo Ishii)
-

A.74.2. Bug fixes

- Fix the wrong error message "ERROR: connection cache is full", when all backend nodes are down. ([bug 487](#)) (Bo Peng)
- When all backend nodes are down, Pgpool-II throws an incorrect error message "ERROR: connection cache is full". Change the error message to "all backend nodes are down, pgpool requires at least one valid node".
- Avoid exit/fork storm of `pool_worker_child` process. (Tatsuo Ishii)
- `pool_worker_child` issues query to get WAL position using `do_query()`, which could throws FATAL error. In this case `pool_worker_child` process exits and Pgpool-II parent immediately forks new process. This cycle indefinitely repeats and gives high load to the system. To avoid the exit/fork storm, sleep `sr_check_period`.

- Fix [black_function_list](#)'s broken default value. (Tatsuo Ishii)
- Fix "not enough space in buffer" error. ([bug 499](#)) (Tatsuo Ishii)

The error occurred while processing error message returned from backend and the cause is that the query string in question is too big. Problem is, the buffer is in fixed size (8192 bytes). Eliminate the fixed size buffer and use pallocated buffer instead. This also saves some memory copy work.

- Fix DROP DATABASE failure. (Tatsuo Ishii)
 - Fix wrong variable in read_status_file() function. ([bug 493](#)) (Takuma Hoshiai)
 - Fix compiler warnings. (Tatsuo Ishii)
-

A.75. Release 3.4.23

Release Date: 2019-03-29

A.75.1. Enhancements

- Add new configuration option [ssl_prefer_server_ciphers](#). (Muhammad Usama)

Add the new setting [ssl_prefer_server_ciphers](#) to let users configure if they want client's or server's cipher order to take preference.

The default for this parameter is off, which prioritize the client's cipher order as usual. However this is just for keeping backward compatibility, and it is possible that a malicious client uses weak ciphers. For this reason we recommend to set this parameter to on at all times.

- Allow to set a client cipher list. (Tatsuo Ishii)

For this purpose new parameter [ssl_ciphers](#), which specifies the cipher list to be accepted by Pgpool-II, is added. This is already implemented in PostgreSQL and useful to enhance security when SSL is enabled.

A.75.2. Bug fixes

- Fix unnecessary fsync() to pgpool_status file. (Tatsuo Ishii)

Whenever new connections are created to PostgreSQL backend, fsync() was issued to pgpool_status file, which could generate excessive I/O in certain conditions. So reduce the chance of issuing fsync() so that it is issued only when backend status is changed.

Discussion: [\[pgpool-general: 6436\]](#)

A.76. Release 3.4.22

Release Date: 2019-02-21

A.76.1. Bug fixes

- Doc: Fix configuration change timing regarding [memory_cache_enabled](#). (Tatsuo Ishii)
 - Fix online recovery failed due to [client_idle_limit_in_recovery](#) in certain cases. ([bug 431](#)) (Tatsuo Ishii)
 - Fix corner case bug when `strip_quote()` handle a empty query string. ([bug 458](#)) (Tatsuo Ishii)
 - Fix Pgpool child segfault if failover occurs when trying to establish a connection. (Tatsuo Ishii)
See [\[pgpool-hackers: 3214\]](#) for discussion.
 - Fix Pgpool-II hang when `idle_in_transaction_session_timeout = on`. ([bug 448](#)) (Tatsuo Ishii)
-

A.77. Release 3.4.21

Release Date: 2018-11-22

A.77.1. Bug fixes

- Fix to sort startup packet's parameters sent by client. ([bug 444](#))(Takuma Hoshiai)

If order of startup packet's parameters differ between cached connection pools and connection request, didn't use connection pool ,and created new connection pool.

- Fix segmentation fault occurs when a certain Bind message is sent in native replication mode. ([bug 443](#))(Bo Peng)

If the number of parameter format codes is specified to one, but the number of the original query's parameter is zero, `bind_rewrite_timestamp()` will call `memcpy` with a negative value. This causes segmentation fault.

Patch is provided by Yugo Nagata.

- Fix a query passed to `relcache` so that it uses schema qualified table name. (Tatsuo Ishii)
 - Fix query cache invalidation bug. (Tatsuo Ishii)
 - Fix memory leak in extended query + query cache enabled. (Tatsuo Ishii)
-

A.78. Release 3.4.20

Release Date: 2018-10-31

A.78.1. Changes

- Change `pgpool.spec` file to install extension to DB server which supports LLVM JIT. (Bo Peng)
- Test: Add regression test for SSL connection. (Tatsuo Ishii)
- Test: Add definition of PGLIB in `regress.sh`. (Bo Peng)

Patch provided by Jesper Pedersen.

A.78.2. Bug fixes

- Fix typo in `child_max_connections` description of `SHOW POOL_STATUS` output. (Tatsuo Ishii)

Patch provided by Phil Ramirez.

- Fix segmentation fault when error query and Sync message are sent in native replication mode. ([bug 434](#)) (Takuma Hoshiai)

In native replication mode, segmentation fault occurs when Sync messages is sent just after a query error.

- Fix syntax error when queries including time functions and `IN (SELECT ...)` in `WHERE` clause in native replication mode. ([bug 433](#)) (Bo Peng)

In native replication mode, queries including time functions (e.g. `now()`, `CURRENT_TIMESTAMP` etc.) are rewritten to a timestamp constant value. However, `Pgpool-II` doesn't support queries including time functions and `IN (SELECT ...)` in `WHERE` clause.

- Fix memory leak in `trigger_failover_command()`. (Tatsuo Ishii)
- Fix memory leak when `memory_cache_enabled = on` and write SQLs are sent. (Bo Peng)

In a explicit transaction, the `SELECT` results are cached in temporary buffer. If a write SQL is sent which modifies the table, the temporary buffer should be reset.

- Test: Fix occasional failure in regression 065.bug152. (Tatsuo Ishii)
- Add missing `pgpool_recovery--1.0--1.1.sql` file to update `pgpool_recovery()` function version to 1.1. (Bo Peng) Add missing `pgpool_recovery--1.0--1.1.sql` file to update `pgpool_recovery()` function version to 1.1. (Bo Peng)
- Do not update `pool_passwd` if the password length is incorrect. ([bug 419](#)) (Takuma Hoshiai, Tatsuo Ishii)

For `Pgpool-II` 3.7 or before, the password stored in `pool_passwd` is MD5 password only. So check the correctness of `pool_passwd` by scanning entire file.

- Test: Update `clean.sh` which clean up regression test results. (Bo Peng)

Patch provided by Jesper Pedersen.

- Add `.gitignore` files. (Bo Peng)

Patch provided by Jesper Pedersen.

- Fix segfault when node 0 is in down status in case of both health check and [failover_on_backend_error](#) are disabled. (Tatsuo Ishii)

A.79. Release 3.4.19

Release Date: 2018-07-31

A.79.1. Bug fixes

- Fix "write on backend 0 failed with error : "Success"" error. ([bug 403](#)) (Tatsuo Ishii)

Don't treated it as an error if `write()` returns 0.

- Fix memory leaks related to `pool_extract_error_message()`. (Tatsuo Ishii)
- Fix an incorrect declare as `bool`, rather than `int` in `pool_extract_error_message()`. (Tatsuo Ishii)

This led to a segfault issue mentioned on certain platform.

- Fix segfault in `per_node_error_log()` on armhf architecture. (Tatsuo Ishii)

Patch provided by Christian Ehrhardt.

- Test: Fix 006.memqcache test failure. (Tatsuo Ishii)
-

A.80. Release 3.4.18

Release Date: 2018-06-12

A.80.1. Bug fixes

- Prevent [pcp_recovery_node](#) from recovering "unused" status node. (Tatsuo Ishii)

This allowed to try to recovery a node without configuration data, which leads to variety of problems.

See [\[pgpool-general: 5963\]](#) for more details.

Also I fixed `pgpool_recovery` function so that it quotes an empty string argument with double quotes.

A.81. Release 3.4.17

Release Date: 2018-04-17

A.81.1. Bug fixes

- Test: Add new regression test for node 0 is down. (Tatsuo Ishii)
 - Make calls to `to_regclass` fully schema qualified. (Tatsuo Ishii)
 - Test: Improve the test script 003.failover. (Bo Peng)
 - Allow to support `pgpool_switch_xlog` PostgreSQL 10. (Tatsuo Ishii)
 - Fix `pgpool_setup` failure in replication mode. (Tatsuo Ishii)
-

A.82. Release 3.4.16

Release Date: 2018-02-13

A.82.1. Changes

- Set TCP_NODELAY and non blocking to frontend socket. (Tatsuo Ishii)
TCP_NODELAY is employed by PostgreSQL, so do we it.
 - Change systemd service file to use STOP_OPTS="-m fast". (Bo Peng)
 - Change pgpool_setup to add restore_command in recovery.conf. (Bo Peng)
-

A.82.2. Bug fixes

- Fix segfault when %a is in log_line_prefix and debug message is on. ([bug 376](#)) (Tatsuo Ishii)
 - Fix queries hanging in parse_before_bind with extended protocol and replication + load-balancing. ([bug 377](#)) (Tatsuo Ishii)
-

A.83. Release 3.4.15

Release Date: 2018-01-09

A.83.1. Bug fixes

- Replace /bin/ed with /bin/sed in [pgpool_setup](#), because /bin/sed is included in most distribution's base packages. (Tatsuo Ishii)
- Change the pgpool.service and sysconfig files to output Pgpool-II log. (Bo Peng)

Removing "Type=forking" and add OPTS="-n" to run Pgpool-II with non-daemon mode, because we need to redirect logs. Using "journalctl" command to see Pgpool-II systemd log.

- Fix timestamp data inconsistency by replication mode. (Bo Peng)

From PostgreSQL10 the column default value such as 'CURRENT_DATE' changes, Pgpool-II didn't rewrite timestamp by the added default values. This caused data inconsistency.

- Fix returning transaction state when "ready for query" message received. (Tatsuo Ishii)

We return primary or master node state of ready for query message to frontend. In most cases this is good. However if other than primary node or master node returns an error state (this could happen if load balance node is other than primary or master node and the query is an erroneous SELECT), this should be returned to frontend, because the frontend already received an error.

- Fix pgpool start message printed multiple times. (Tatsuo Ishii)
-

A.84. Release 3.4.14

Release Date: 2017-11-01

A.84.1. Bug fixes

- Add different pgpool.sysconfig file for RHEL6 and RHEL7. ([bug 343](#)) (Bo Peng)

In RHEL6, the "-n" option is needed to redirect log.

- Fix finding primary node is not working in 3.4.12, 3.4.13. (Tatsuo Ishii)
- Fix bug mistakenly overriding global backend status right after failover. (Tatsuo Ishii)

See [\[pgpool-general: 5728\]](#) for more details.

- Deal with OpenSSL 1.1. (Tatsuo Ishii, Muhammad Usama)

A.85. Release 3.4.13

Release Date: 2017-09-05

A.85.1. Bug fixes

- Doc: Fix documentation about load-balancing. (Yugo Nagata)
- Fix ancient bug of `pool_unread()`. (Tatsuo Ishii)

When `realloc()` is called in `pool_unread()`, it did not update the buffer size. This could cause variety of memory corruption and unexpected data reading from backend. The reason why we did not found that is, probably recently `Pgpool-II` starts extensively to use `pool_unread()`.

- Test: Fix Java program in 005.regression test. (Tatsuo Ishii)
- Fix for when failover is triggered by worker process, it is possible that wrong DB node could failover. ([bug 303](#)) (Tatsuo Ishii)

This is due to the `db_node_id` member in the `POLL_CONNECTION` structure is not initialized in the process (in child process the member is properly initialized). To solve the problem, add new function `pool_set_db_node_id()` to set the structure member variable and call it inside `make_persistent_db_connection()`.

- Fix starting unnecessary transaction when `SET` command is issued. (Tatsuo Ishii)
- Fix for [\[pgpool-general: 5621\]](#) `Failover()` function should be executed with health check alarm disabled. (Muhammad Usama)
- Allow make dist to include `pgpool.service`. (Yugo Nagata)

A.86. Release 3.4.12

Release Date: 2017-07-11

A.86.1. Bug fixes

- Importing the latest changes in the `MemoryManager` API from PostgreSQL code. (Muhammad Usama)
- Fixing [\[pgpool-hackers: 2390\]](#) Problems with the relative paths in daemon mode (Muhammad Usama)
- Adjust function name change in PostgreSQL 10 dev head. (Tatsuo Ishii)

```
pg_current_wal_location -> pg_current_wal_lsn
pg_last_wal_replay_location -> pg_last_wal_replay_lsn
```

- Fix query cache hang when used by node.js. (Tatsuo Ishii)

See [\[pgpool-general: 5511\]](#) for more details.

- Deal with PostgreSQL 10 in streaming replication delay checking. (Tatsuo Ishii)
- Fix query cache memory leak. (Tatsuo Ishii)

Clearing cache buffers in case of no oid queries (like `BEGIN`, `CHECKPOINT`, `VACUUM`, etc) should have been done, but it did not.

- Fix corner case bug in Pgpool-II starting up. (Tatsuo Ishii)

It is possible that a failover request is accepted before primary node is searched. This leads Pgpool-II to a strange state: there's no primary node if the failed node was a primary node (even if new primary node exists as a result of promotion of existing standby).

See [\[pgpool-hackers: 2321\]](#) for more details.

A.87. Release 3.4.11

Release Date: 2017-04-28

A.87.1. Bug fixes

- Fix for 0000299: Errors on the reloading of configuration. ([Bug 299](#)) (Muhammad Usama)
- Fix coverity warnings. (Muhammad Usama)
- Fix for [\[pgpool-general: 5396\]](#) pam ldap failure. (Muhammad Usama)
- Fix for 0000296: Pgpool v3.6.2 terminated by systemd because the service Type has been set to 'forking'. ([Bug 296](#)) (Muhammad Usama)

A.88. Release 3.4.10

Release Date: 2017-03-17

A.88.1. Bug fixes

- Add "Wants=network.target" to pgpool.service file. ([bug 294](#)) (Bo Peng)
- Fix [pcp_promote_node](#) bug that fails promoting node 0. (Yugo Nagata)

The master node could not be promoted by `pcp_promote_node` with the following error;

```
FATAL: invalid pgpool mode for process recovery request
DETAIL: specified node is already primary node, can't promote node id 0
```

In streaming replication mode, there is a case that Pgpool-II regards the status of primary node as "standby" for some reasons, for example, when `pg_ctl promote` is executed manually during Pgpool-II is running, in which case, it seems to Pgpool-II that the primary node doesn't exist.

This status mismatch should be fixed by `pcp_promote_node`, but when the node is the master node (the first alive node), it fails as mentioned above.

The reason is as following. before changing the status, `pcp_promote_node` checks if the specified node is already primary or not by comparing the node id with `PRIMARY_NODE_ID`. However, if the primary doesn't exist from Pgpool-II's view, `PRIMARY_NODE_ID` is set to 0, which is same as `MASTER_NODE_ID`. Hence, when the master node is specified to be promoted, `pcp_promote_node` is confused that this node is already primary and doesn't have to be promoted, and it exits with the error.

To fix this, `pcp_promote_node` should check the node id by using `REAL_PRIMARY_NODE_ID`, which is set -1 when the primary doesn't exist, rather than `PRIMARY_NODE_ID`.

- Add the latest release note link to README file.(Bo Peng)
- Pgpool-II should not perform ping test after bringing down the VIP. (Muhammad Usama)

This issue was reported by the reporter of bug:[pgpool-II 0000249]: watchdog sometimes fails de-escalation

- Fix to release shared memory segments when Pgpool-II exits. ([bug 272](#)) (Tatsuo Ishii)
- Fix for [[pgpool-general: 5315](#)] `pg_terminate_backend` (Muhammad Usama)
- Adding the missing `ExecStop` and `ExecReload` commands to the systemd service configuration file. (Muhammad Usama)
- Fix for 281: "segmentation fault" when execute `pcp_attach_node`. ([bug 281](#)) (Muhammad Usama)
- Fix load balancing bug in streaming replication mode. (Tatsuo Ishii)

In an explicit transaction, any `SELECT` will be load balanced until write query is sent. After writing query is sent, any `SELECT` should be sent to the primary node. However if a `SELECT` is sent before a sync message is sent, this does not work since the treatment of writing query is done after ready for query message arrives.

Solution is, the treatment for writing query is done in executing the writing query as well.

The bug has been there since V3.5.

- Fix yet another kind mismatch error in streaming replication mode. (Tatsuo Ishii)
- Fix `do_query()` hangs after close message. (Tatsuo Ishii)
- Fixing stack smashing detected. ([bug 280](#)) (Muhammad Usama)

It was a buffer overflow in `wd_get_cmd` function

- Remove `elog/ereport` calls from signal handlers. (Tatsuo Ishii)

See [[pgpool-hackers: 1950](#)] for details.

- Fix bug failed to create INET domain socket in FreeBSD if `listen_addresses = '*'`. ([bug 202](#)) (Bo Peng)
- Fix for 0000249: watchdog sometimes fails de-escalation. ([bug 249](#)) (Muhammad Usama)

The solution is to use the `waitpid()` system call without `WNOHANG` option.

- Fix `connection_life_time` broken by `authentication_timeout`. (Yugo Nagata)
- Fix authentication timeout that can occur right after client connections. (Yugo Nagata)

A.89. Release 3.4.9

Release Date: 2016-12-26

A.89.1. Bug fixes

- Tightening up the watchdog security. (Muhammad Usama)

Now `wd_authkey` uses the HMAC SHA-256 hashing.

- Add `pgpool_adm` extension in `Pgpool-IIRPM`. (Bo Peng)
- Fix occasional segfault when query cache is enabled. (bug 263) (Tatsuo Ishii)
- Do not cancel a query when the query resulted in an error other than in native replication mode. (Tatsuo Ishii)

It was intended to keep the consistency, but there's no point in other than native replication mode.

- Remove obsoleted option "-c" in `pgpool` command. (Tatsuo Ishii)

Also fix typo in the help message.

- Change the default value of `search_primary_node_timeout` from 10 to 300. (Tatsuo Ishii)

Prior default value 10 seconds is sometimes too short for a standby to be promoted.

Per [pgpool-general: 5026].

- Fix the case when all backends are down then 1 node attached. (bug 248) (Tatsuo Ishii)

When all backends are down, no connection is accepted. Then 1 PostgreSQL becomes up, and attach the node using `pcp_attach_node`. It successfully finishes. However, when a new connection arrives, still the connection is refused because `pgpool` child process looks into the cached status, in which the recovered node is still in down status if mode is streaming replication mode (native replication and other modes are fine). Solution is, if all nodes are down, force to restart all `pgpool` child.

- Do not use `random()` while generating MD5 salt. (Tatsuo Ishii)

`random()` should not be used in security related applications. To replace `random()`, import `PostmasterRandom()` from PostgreSQL. Also store current time at the start up of `Pgpool-II` main process for later use.

- Don't ignore sync message from frontend when query cache is enabled. (Tatsuo Ishii)

A.90. Release 3.3.22

Release Date: 2018-07-31

A.90.1. Bug fixes

- Fix "write on backend 0 failed with error :\"Success\"" error. ([bug 403](#)) (Tatsuo Ishii)

Don't treated it as an error if `write()` returns 0.

- Fix segfault in `per_node_error_log()` on armhf architecture. (Tatsuo Ishii)
Patch provided by Christian Ehrhardt.
-

A.91. Release 3.3.21

Release Date: 2018-04-17

A.91.1. Bug fixes

- Make calls to `to_regclass` fully schema qualified. (Tatsuo Ishii)
-

A.92. Release 3.3.20

Release Date: 2018-02-13

A.92.1. Changes

- Change systemd service file to use `STOP_OPTS="-m fast"`. (Bo Peng)
 - Change `pgpool_setup` to add `restore_command` in `recovery.conf`. (Bo Peng)
-

A.92.2. Bug fixes

- Fix queries hanging in `parse_before_bind` with extended protocol and replication + load-balancing. ([bug 377](#)) (Tatsuo Ishii)
-

A.93. Release 3.3.19

Release Date: 2018-01-09

A.93.1. Bug fixes

- Change the `pgpool.service` and `sysconfig` files to output Pgpool-II log. (Bo Peng)

Removing "Type=forking" and add `OPTS="-n"` to run Pgpool-II with non-daemon mode, because we need to redirect logs. Using "journalctl" command to see Pgpool-II systemd log.

- Fix timestamp data inconsistency by replication mode. (Bo Peng)

From PostgreSQL10 the column default value such as 'CURRENT_DATE' changes, Pgpool-II didn't rewrite timestamp by the added default values. This caused data inconsistency.

- Fix returning transaction state when "ready for query" message received. (Tatsuo Ishii)

We return primary or master node state of ready for query message to frontend. In most cases this is good. However if other than primary node or master node returns an error state (this could happen if load balance node is other than primary or master node and the query is an erroneous SELECT), this should be returned to frontend, because the frontend already received an error.

A.94. Release 3.3.18

Release Date: 2017-11-01

A.94.1. Bug fixes

- Add different pgpool.sysconfig file for RHEL6 and RHEL7. ([bug 343](#)) (Bo Peng)
In RHEL6, the "-n" option is needed to redirect log.
 - Fix bug mistakenly overriding global backend status right after failover. (Tatsuo Ishii)
See [\[pgpool-general: 5728\]](#) for more details.
 - Deal with OpenSSL 1.1. (Tatsuo Ishii, Muhammad Usama)
-

A.95. Release 3.3.17

Release Date: 2017-09-05

A.95.1. Bug fixes

- Doc: Fix documentation about load-balancing. (Yugo Nagata)
 - Fix ancient bug of pool_unread(). (Tatsuo Ishii)
When realloc() is called in pool_unread(), it did not update the buffer size. This could cause variety of memory corruption and unexpected data reading from backend. The reason why we did not found that is, probably recently Pgpool-II starts extensively to use pool_unread().
 - Test: Fix Java program in 005.regression test. (Tatsuo Ishii)
 - Fix for when failover is triggered by worker process, it is possible that wrong DB node could failover. ([bug 303](#)) (Tatsuo Ishii)
This is due to the db_node_id member in the POLL_CONNECTION structure is not initialized in the process (in child process the member is properly initialized). To solve the problem, add new function pool_set_db_node_id() to set the structure member variable and call it inside make_persistent_db_connection().
 - Fix starting unnecessary transaction when SET command is issued. (Tatsuo Ishii)
 - Fix for [\[pgpool-general: 5621\]](#) Failover() function should be executed with health check alarm disabled. (Muhammad Usama)
-

A.96. Release 3.3.16

Release Date: 2017-07-11

A.96.1. Bug fixes

- Fixing [\[pgpool-hackers: 2390\]](#) Problems with the relative paths in daemon mode (Muhammad Usama)
- Adjust function name change in PostgreSQL 10 dev head. (Tatsuo Ishii)

```
pg_current_wal_location  -> pg_current_wal_lsn  
pg_last_wal_replay_location -> pg_last_wal_replay_lsn
```

- Fix query cache hang when used by node.js. (Tatsuo Ishii)
See [\[pgpool-general: 5511\]](#) for more details.
- Deal with PostgreSQL 10 in streaming replication delay checking. (Tatsuo Ishii)
- Fix query cache memory leak. (Tatsuo Ishii)

Clearing cache buffers in case of no oid queries (like `BEGIN`, `CHECKPOINT`, `VACUUM`, etc) should have been done, but it did not.

A.97. Release 3.3.15

Release Date: 2017-04-28

A.97.1. Bug fixes

- Fix for 0000299: Errors on the reloading of configuration. ([Bug 299](#)) (Muhammad Usama)
- Fix coverity warnings. (Muhammad Usama)
- Fix for 0000296: Pgpool v3.6.2 terminated by systemd because the service Type has been set to 'forking'. ([Bug 296](#)) (Muhammad Usama)

A.98. Release 3.3.14

Release Date: 2017-03-17

A.98.1. Bug fixes

- Add "Wants=network.target" to pgpool.service file. ([bug 294](#)) (Bo Peng)
- Fix [pcp_promote_node](#) bug that fails promoting node 0. (Yugo Nagata)

The master node could not be promoted by `pcp_promote_node` with the following error;

```
FATAL: invalid pgpool mode for process recovery request
DETAIL: specified node is already primary node, can't promote node id 0
```

In streaming replication mode, there is a case that Pgpool-II regards the status of primary node as "standby" for some reasons, for example, when `pg_ctl promote` is executed manually during Pgpool-II is running, in which case, it seems to Pgpool-II that the primary node doesn't exist.

This status mismatch should be fixed by `pcp_promote_node`, but when the node is the master node (the first alive node), it fails as mentioned above.

The reason is as following. before changing the status, `pcp_promote_node` checks if the specified node is already primary or not by comparing the node id with `PRIMARY_NODE_ID`. However, if the primary doesn't exist from Pgpool-II's view, `PRIMARY_NODE_ID` is set to 0, which is same as `MASTER_NODE_ID`. Hence, when the master node is specified to be promoted, `pcp_promote_node` is confused that this node is already primary and doesn't have to be promoted, and it exits with the error.

To fix this, `pcp_promote_node` should check the node id by using `REAL_PRIMARY_NODE_ID`, which is set -1 when the primary doesn't exist, rather than `PRIMARY_NODE_ID`.

- Add the latest release note link to README file.(Bo Peng)
- Fix to release shared memory segments when Pgpool-II exits. ([bug 272](#)) (Tatsuo Ishii)
- Fix for [\[pgpool-general: 5315\]](#) `pg_terminate_backend` (Muhammad Usama)
- Adding the missing `ExecStop` and `ExecReload` commands to the systemd service configuration file. (Muhammad Usama)
- Fixing stack smashing detected. ([bug 280](#)) (Muhammad Usama)

It was a buffer overflow in `wd_get_cmd` function

- Remove `pool_log/pool_error` calls from signal handlers. (Tatsuo Ishii)
- See [\[pgpool-hackers: 1950\]](#) for details.
- Fix for 0000249: watchdog sometimes fails de-escalation. ([bug 249](#)) (Muhammad Usama)
- The solution is to use the `waitpid()` system call without `WNOHANG` option.
- Fix `connection_life_time` broken by `authentication_timeout`. (Yugo Nagata)
- Fix authentication timeout that can occur right after client connections. (Yugo Nagata)

A.99. Release 3.3.13

Release Date: 2016-12-26

A.99.1. Bug fixes

- Tightening up the watchdog security. (Muhammad Usama)
- Now `wd_authkey` uses the HMAC SHA-256 hashing.

- Add `pgpool_adm` extension in **Pgpool-II RPM**. (Bo Peng)
- Fix occasional segfault when query cache is enabled. (bug 263) (Tatsuo Ishii)
- Do not cancel a query when the query resulted in an error other than in native replication mode. (Tatsuo Ishii)

It was intended to keep the consistency, but there's no point in other than native replication mode.

- Change the default value of `search_primary_node_timeout` from 10 to 300. (Tatsuo Ishii)

Prior default value 10 seconds is sometimes too short for a standby to be promoted.

Per `[pgpool-general: 5026]`.

- Fix the case when all backends are down then 1 node attached. (bug 248) (Tatsuo Ishii)

When all backends are down, no connection is accepted. Then 1 PostgreSQL becomes up, and attach the node using `pcp_attach_node`. It successfully finishes. However, when a new connection arrives, still the connection is refused because `pgpool` child process looks into the cached status, in which the recovered node is still in down status if mode is streaming replication mode (native replication and other modes are fine). Solution is, if all nodes are down, force to restart all `pgpool` child.

- Do not use `random()` while generating MD5 salt. (Tatsuo Ishii)

`random()` should not be used in security related applications. To replace `random()`, import `PostmasterRandom()` from PostgreSQL. Also store current time at the start up of `Pgpool-II` main process for later use.

- Don't ignore sync message from frontend when query cache is enabled. (Tatsuo Ishii)

A.100. Release 3.2.22

Release Date: 2017-09-05

A.100.1. Bug fixes

- Doc: Fix documentation about load-balancing. (Yugo Nagata)
- Fix ancient bug of `pool_unread()`. (Tatsuo Ishii)

When `realloc()` is called in `pool_unread()`, it did not update the buffer size. This could cause variety of memory corruption and unexpected data reading from backend. The reason why we did not found that is, probably recently `Pgpool-II` starts extensively to use `pool_unread()`.

- Fix for when failover is triggered by worker process, it is possible that wrong DB node could failover. ([bug 303](#)) (Tatsuo Ishii)

This is due to the `db_node_id` member in the `POLL_CONNECTION` structure is not initialized in the process (in child process the member is properly initialized). To solve the problem, add new function `pool_set_db_node_id()` to set the structure member variable and call it inside `make_persistent_db_connection()`.

- Fix starting unnecessary transaction when `SET` command is issued. (Tatsuo Ishii)
- Fix for [\[pgpool-general: 5621\]](#) `Failover()` function should be executed with health check alarm disabled. (Muhammad Usama)

A.101. Release 3.2.21

Release Date: 2017-07-11

A.101.1. Bug fixes

- Fixing [\[pgpool-hackers: 2390\]](#) Problems with the relative paths in daemon mode (Muhammad Usama)
- Adjust function name change in PostgreSQL 10 dev head. (Tatsuo Ishii)

```
pg_current_wal_location  -> pg_current_wal_lsn  
pg_last_wal_replay_location -> pg_last_wal_replay_lsn
```

- Fix query cache hang when used by node.js. (Tatsuo Ishii)
See [\[pgpool-general: 5511\]](#) for more details.
- Deal with PostgreSQL 10 in streaming replication delay checking. (Tatsuo Ishii)
- Fix query cache memory leak. (Tatsuo Ishii)

Clearing cache buffers in case of no oid queries (like `BEGIN`, `CHECKPOINT`, `VACUUM`, etc) should have been done, but it did not. Fix query cache memory leak. (Tatsuo Ishii)

A.102. Release 3.2.20

Release Date: 2017-04-28

A.102.1. Bug fixes

- Fix for 0000299: Errors on the reloading of configuration. ([Bug 299](#)) (Muhammad Usama)

A.103. Release 3.2.19

Release Date: 2017-03-17

A.103.1. Bug fixes

- Fix [pcp_promote_node](#) bug that fails promoting node 0. (Yugo Nagata)

The master node could not be promoted by `pcp_promote_node` with the following error;

```
FATAL: invalid pgpool mode for process recovery request  
DETAIL: specified node is already primary node, can't promote node id 0
```

In streaming replication mode, there is a case that Pgpool-II regards the status of primary node as "standby" for some reasons, for example, when `pg_ctl promote` is executed manually during Pgpool-II is running, in which case, it seems to Pgpool-II that the primary node doesn't exist.

This status mismatch should be fixed by `pcp_promote_node`, but when the node is the master node (the first alive node), it fails as mentioned above.

The reason is as following. before changing the status, `pcp_promote_node` checks if the specified node is already primary or not by comparing the node id with `PRIMARY_NODE_ID`. However, if the primary doesn't exist from Pgpool-II's view, `PRIMARY_NODE_ID` is set to 0, which is same as `MASTER_NODE_ID`. Hence, when the master node is specified to be promoted, `pcp_promote_node` is confused that this node is already primary and doesn't have to be promoted, and it exits with the error.

To fix this, `pcp_promote_node` should check the node id by using `REAL_PRIMARY_NODE_ID`, which is set -1 when the primary doesn't exist, rather than `PRIMARY_NODE_ID`.

- Add the latest release note link to README file.(Bo Peng)
- Fix to release shared memory segments when Pgpool-II exits. ([bug 272](#)) (Tatsuo Ishii)
- Remove `pool_log/pool_error` calls from signal handlers. (Tatsuo Ishii)

See [[pgpool-hackers: 1950](#)] for details.

- Fix for 0000249: watchdog sometimes fails de-escalation. ([bug 249](#)) (Muhammad Usama)

The solution is to use the `waitpid()` system call without `WNOHANG` option.

- Fix `connection_life_time` broken by `authentication_timeout`. (Yugo Nagata)
- Fix authentication timeout that can occur right after client connections. (Yugo Nagata)

A.104. Release 3.2.18

Release Date: 2016-12-26

A.104.1. Bug fixes

- Fix occasional segfault when query cache is enabled. (Tatsuo Ishii)

Per bug 263.

- Do not cancel a query when the query resulted in an error other than in native replication mode. (Tatsuo Ishii)

It was intended to keep the consistency, but there's no point in other than native replication mode.

- Do not use `random()` while generating MD5 salt. (Tatsuo Ishii)

`random()` should not be used in security related applications. To replace `random()`, import `PostmasterRandom()` from PostgreSQL. Also store current time at the start up of Pgpool-II main process for later use.

- Don't ignore sync message from frontend when query cache is enabled. (Tatsuo Ishii)

A.105. Release 3.1.21

Release Date: 2016-12-26

A.105.1. Bug fixes

- Do not cancel a query when the query resulted in an error other than in native replication mode. (Tatsuo Ishii)

It was intended to keep the consistency, but there's no point in other than native replication mode.

- Do not use random() while generating MD5 salt. (Tatsuo Ishii)

random() should not be used in security related applications. To replace random(), import PostmasterRandom() from PostgreSQL. Also store current time at the start up of Pgpool-II main process for later use.

- Don't ignore sync message from frontend when query cache is enabled. (Tatsuo Ishii)
-

Index

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [H](#) | [I](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [W](#) | [Y](#)

A

allow_clear_text_frontend_auth configuration parameter, [Authentication Settings](#)
allow_multiple_failover_requests_from_node configuration parameter, [Controlling the Failover behavior](#)
allow_sql_comments configuration parameter, [Load Balancing Settings](#)
app_name_redirect_preference_list configuration parameter, [Load Balancing Settings](#)
arping_cmd configuration parameter, [Virtual IP control](#)
arping_path configuration parameter, [Virtual IP control](#)
AUTH, [Using different methods for frontend and backend authentication](#), [Using AES256 encrypted passwords in](#)
authentication_timeout configuration parameter, [Authentication Settings](#)
auto_failback configuration parameter, [Failover and Failback Settings](#)
auto_failback_interval configuration parameter, [Failover and Failback Settings](#)

B

backend_application_name configuration parameter, [Backend Data Settings](#)
backend_data_directory configuration parameter, [Backend Data Settings](#)
backend_flag configuration parameter, [Backend Data Settings](#)
backend_hostname configuration parameter, [Backend Connection Settings](#)
backend_port configuration parameter, [Backend Connection Settings](#)
backend_weight configuration parameter, [Backend Connection Settings](#)
bison, [Requirements](#)
black_function_list configuration parameter, [Load Balancing Settings](#)
black_memqcache_table_list configuration parameter, [Common configurations](#)
black_query_pattern_list configuration parameter, [Load Balancing Settings](#)

C

Certificate, [Certificate Authentication](#)
check_temp_table configuration parameter, [Misc Configuration Parameters](#)
check_unlogged_table configuration parameter, [Misc Configuration Parameters](#)
child_life_time configuration parameter, [Connection Pooling Settings](#)
child_max_connections configuration parameter, [Connection Pooling Settings](#)
clear_memqcache_on_escalation configuration parameter, [Behaviour on escalation and de-escalation](#)
client authentication, [Client Authentication](#)
client_idle_limit configuration parameter, [Connection Pooling Settings](#)
client_idle_limit_in_recovery configuration parameter, [Online Recovery](#)
client_min_messages configuration parameter, [When To Log](#)
configuration

of the server, [Server Configuration](#)

configuring watchdog, [Watchdog](#)

connection_cache configuration parameter, [Connection Pooling Settings](#)

connection_life_time configuration parameter, [Connection Pooling Settings](#)

connect_timeout configuration parameter, [Health Check](#)

D

database_redirect_preference_list configuration parameter, [Load Balancing Settings](#)

delay_threshold configuration parameter, [Streaming Replication Check](#)

delegate_IP configuration parameter, [Virtual IP control](#)

detach_false_primary configuration parameter, [Failover and Failback Settings](#)

disable_load_balance_on_write configuration parameter, [Load Balancing Settings](#)

E

enable_consensus_with_half_votes configuration parameter, [Controlling the Failover behavior](#)

enable_pool_hba configuration parameter, [Authentication Settings](#)

enable_shared_relcache configuration parameter, [Misc Configuration Parameters](#)

F

failback_command configuration parameter, [Failover and Failback Settings](#)

failover_command configuration parameter, [Failover and Failback Settings](#)

failover_if_affected_tuples_mismatch configuration parameter, [Replication mode](#)

failover_on_backend_error configuration parameter, [Failover and Failback Settings](#)

failover_require_consensus configuration parameter, [Controlling the Failover behavior](#)

failover_when_quorum_exists configuration parameter, [Controlling the Failover behavior](#)

flex, [Requirements](#)

follow_master_command configuration parameter, [Failover and Failback Settings](#)

H

health_check_database configuration parameter, [Health Check](#)

health_check_max_retries configuration parameter, [Health Check](#)

health_check_password configuration parameter, [Health Check](#)

health_check_period configuration parameter, [Health Check](#)

health_check_retry_delay configuration parameter, [Health Check](#)

health_check_timeout configuration parameter, [Health Check](#)

health_check_user configuration parameter, [Health Check](#)

heartbeat_destination configuration parameter, [Lifecheck Heartbeat mode configuration](#)

heartbeat_destination_port configuration parameter, [Lifecheck Heartbeat mode configuration](#)

heartbeat_device configuration parameter, [Lifecheck Heartbeat mode configuration](#)

history

of Pgpool-II, [A Brief History of Pgpool-II](#)

I

if_cmd_path configuration parameter, [Virtual IP control](#)

if_down_cmd configuration parameter, [Virtual IP control](#)

if_up_cmd configuration parameter, [Virtual IP control](#)

ignore_leading_white_space configuration parameter, [Load Balancing Settings](#)

insert_lock configuration parameter, [Replication mode](#)

installation, [Installation of Pgpool-II](#)

L

lex, [Requirements](#)
listen_addresses configuration parameter, [Connection Settings](#)
listen_backlog_multiplier configuration parameter, [Connection Pooling Settings](#)
load_balance_mode configuration parameter, [Load Balancing Settings](#)
lobj_lock_table configuration parameter, [Replication mode](#)
logdir configuration parameter, [Misc Configuration Parameters](#)
logical replication mode, [Running mode of Pgpool-II](#)
log_client_messages configuration parameter, [What To Log](#)
log_connections configuration parameter, [What To Log](#)
log_destination configuration parameter, [Where To Log](#)
log_error_verbosity configuration parameter, [What To Log](#)
log_hostname configuration parameter, [What To Log](#)
log_line_prefix configuration parameter, [What To Log](#)
log_min_messages configuration parameter, [When To Log](#)
log_per_node_statement configuration parameter, [What To Log](#)
log_standby_delay configuration parameter, [Streaming Replication Check](#)
log_statement configuration parameter, [What To Log](#)

M

make, [Requirements](#)
master slave mode, [Running mode of Pgpool-II](#)
master_slave_mode configuration parameter, [Master slave mode](#)
master_slave_sub_mode configuration parameter, [Master slave mode](#)
max_pool configuration parameter, [Connection Pooling Settings](#)
MD5, [MD5 Password Authentication](#), [Setting md5 Authentication](#)
memory_cache_enabled configuration parameter, [Enabling in memory query cache](#)
memqcache_auto_cache_invalidation configuration parameter, [Common configurations](#)
memqcache_cache_block_size configuration parameter, [Configurations to use shared memory](#)
memqcache_expire configuration parameter, [Common configurations](#)
memqcache_maxcache configuration parameter, [Common configurations](#)
memqcache_max_num_cache configuration parameter, [Configurations to use shared memory](#)
memqcache_memcached_host configuration parameter, [Configurations to use memcached](#)
memqcache_memcached_port configuration parameter, [Configurations to use memcached](#)
memqcache_method configuration parameter, [Choosing cache storage](#)
memqcache_oiddir configuration parameter, [Common configurations](#)
memqcache_total_size configuration parameter, [Configurations to use shared memory](#)

N

native replication mode, [Running mode of Pgpool-II](#)
num_init_children configuration parameter, [Connection Settings](#)

O

other_pgpool_hostname configuration parameter, [Watchdog servers configurations](#)
other_pgpool_port configuration parameter, [Watchdog servers configurations](#)
other_wd_port0 configuration parameter, [Watchdog servers configurations](#)

P

PAM, [PAM Authentication](#)
pcp configuration, [Configuring pcp.conf](#)
pcp_attach_node, [pcp_attach_node](#)
pcp_common_options, [pcp_common_options](#)
pcp_detach_node, [pcp_detach_node](#)
pcp_listen_addresses configuration parameter, [Connection Settings](#)
pcp_node_count, [pcp_node_count](#)
pcp_node_info, [pcp_node_info](#)
pcp_pool_status, [pcp_pool_status](#)
pcp_port configuration parameter, [Connection Settings](#)

pcp_proc_count, [pcp_proc_count](#)
pcp_proc_info, [pcp_proc_info](#)
pcp_promote_node, [pcp_promote_node](#)
pcp_recovery_node, [pcp_recovery_node](#)
pcp_socket_dir configuration parameter, [Connection Settings](#)
pcp_stop_pgpool, [pcp_stop_pgpool](#)
pcp_watchdog_info, [pcp_watchdog_info](#)
performance

of the server, [Performance Considerations](#)

pgpool, [pgpool](#)
Pgpool-II configuration, [Configuring Pgpool-II](#)
Pgpool-II user, [The Pgpool-II User Account](#)
pgpool.conf, [Parameter Interaction via the Configuration File](#)
pgpool_adm_pcp_attach_node, [pgpool_adm_pcp_attach_node](#)
pgpool_adm_pcp_detach_node, [pgpool_adm_pcp_detach_node](#)
pgpool_adm_pcp_node_count, [pgpool_adm_pcp_node_count](#)
pgpool_adm_pcp_node_info, [pgpool_adm_pcp_node_info](#)
pgpool_adm_pcp_pool_status, [pgpool_adm_pcp_pool_status](#)
pgpool_setup, [pgpool_setup](#)
pgproto, [pgproto](#)
pg_enc, [pg_enc](#)
pg_md5, [pg_md5](#)
pid_file_name configuration parameter, [Misc Configuration Parameters](#)
ping_path configuration parameter, [Upstream server connection](#)
pool_hba.conf, [The pool_hba.conf File](#)
pool_passwd configuration parameter, [Authentication Settings](#)
port configuration parameter, [Connection Settings](#)

Q

quarantine, [Controlling the Failover behavior](#)
quorum, [Behaviour on escalation and de-escalation](#), [Controlling the Failover behavior](#)

R

recovery_1st_stage_command configuration parameter, [Online Recovery](#)
recovery_2nd_stage_command configuration parameter, [Online Recovery](#)
recovery_password configuration parameter, [Online Recovery](#)
recovery_timeout configuration parameter, [Online Recovery](#)
recovery_user configuration parameter, [Online Recovery](#)
relcache_expire configuration parameter, [Misc Configuration Parameters](#)
relcache_query_target configuration parameter, [Misc Configuration Parameters](#)
relcache_size configuration parameter, [Misc Configuration Parameters](#)
replicate_select configuration parameter, [Replication mode](#)
replication_mode configuration parameter, [Replication mode](#)
replication_stop_on_mismatch configuration parameter, [Replication mode](#)
reserverd_connections configuration parameter, [Connection Settings](#)
RESET, [PGPOOL RESET](#)
reset_query_list configuration parameter, [Connection Pooling Settings](#)

S

SCRAM, [scram-sha-256 Authentication](#), [Setting scram-sha-256 Authentication](#)
search_primary_node_timeout configuration parameter, [Failover and Failback Settings](#)
serialize_accept configuration parameter, [Connection Pooling Settings](#)
SET, [PGPOOL SET](#)
SHOW, [PGPOOL SHOW](#), [SHOW POOL_CACHE](#)
SHOW POOL_NODES, [SHOW POOL_NODES](#)
SHOW POOL_POOLS, [SHOW POOL_POOLS](#)
SHOW POOL_PROCESSES, [SHOW POOL_PROCESSES](#)
SHOW POOL_STATUS, [SHOW POOL_STATUS](#)

SHOW POOL_VERSION, [SHOW POOL_VERSION](#)
SIGHUP, [Parameter Interaction via the Configuration File](#)
socket_dir configuration parameter, [Connection Settings](#)
sr_check_database configuration parameter, [Streaming Replication Check](#)
sr_check_password configuration parameter, [Streaming Replication Check](#)
sr_check_period configuration parameter, [Streaming Replication Check](#)
sr_check_user configuration parameter, [Streaming Replication Check](#)
ssl configuration parameter, [SSL Settings](#)
ssl_ca_cert configuration parameter, [SSL Settings](#)
ssl_ca_cert_dir configuration parameter, [SSL Settings](#)
ssl_cert configuration parameter, [SSL Settings](#)
ssl_ciphers configuration parameter, [SSL Settings](#)
ssl_dh_params_file configuration parameter, [SSL Settings](#)
ssl_ecdh_curve configuration parameter, [SSL Settings](#)
ssl_key configuration parameter, [SSL Settings](#)
ssl_prefer_server_ciphers configuration parameter, [SSL Settings](#)
statement_level_load_balance configuration parameter, [Load Balancing Settings](#)
streaming replication mode, [Running mode of Pgpool-II](#)
syslog_facility configuration parameter, [Where To Log](#)
syslog_ident configuration parameter, [Where To Log](#)

T

trusted_servers configuration parameter, [Upstream server connection](#)

U

use_watchdog configuration parameter, [Enable watchdog](#)

W

WATCHDOG, [Coordinating multiple Pgpool-II nodes](#), [Life checking of other Pgpool-II nodes](#), [Consistency of configuration parameters on all Pgpool-II nodes](#), [Changing active/standby state when certain fault is detected](#), [Automatic virtual IP switching](#), [Automatic registration of a server as a standby in recovery](#), [Starting/stopping watchdog](#), [Watchdog IPC command packet format](#), [Watchdog IPC result packet format](#), [Watchdog IPC command packet types](#), [External lifeccheck IPC packets and data](#), [Getting list of configured watchdog nodes](#), [Restrictions on watchdog](#), [Watchdog restriction with query mode lifeccheck](#), [Connecting to Pgpool-II whose watchdog status is down](#), [Pgpool-II whose watchdog status is down requires restart](#), [Watchdog promotion to active takes few seconds](#)
watchdog_setup, [watchdog_setup](#)
wd_authkey configuration parameter, [Watchdog communication](#)
wd_de_escalation_command configuration parameter, [Behaviour on escalation and de-escalation](#)
wd_escalation_command configuration parameter, [Behaviour on escalation and de-escalation](#)
wd_heartbeat_deadtime configuration parameter, [Lifeccheck Heartbeat mode configuration](#)
wd_heartbeat_keepalive configuration parameter, [Lifeccheck Heartbeat mode configuration](#)
wd_heartbeat_port configuration parameter, [Lifeccheck Heartbeat mode configuration](#)
wd_hostname configuration parameter, [Watchdog communication](#)
wd_interval configuration parameter, [Life checking Pgpool-II](#)
wd_ipc_socket_dir configuration parameter, [Life checking Pgpool-II](#)
wd_lifeccheck_dbname configuration parameter, [Lifeccheck Query mode configuration](#)
wd_lifeccheck_method configuration parameter, [Life checking Pgpool-II](#)
wd_lifeccheck_password configuration parameter, [Lifeccheck Query mode configuration](#)
wd_lifeccheck_query configuration parameter, [Lifeccheck Query mode configuration](#)
wd_lifeccheck_user configuration parameter, [Lifeccheck Query mode configuration](#)
wd_life_point configuration parameter, [Lifeccheck Query mode configuration](#)
wd_monitoring_interfaces_list configuration parameter, [Life checking Pgpool-II](#)
wd_port configuration parameter, [Watchdog communication](#)
wd_priority configuration parameter, [Life checking Pgpool-II](#)
white_function_list configuration parameter, [Load Balancing Settings](#)
white_memqcache_table_list configuration parameter, [Common configurations](#)

Y

yacc, [Requirements](#)