pgpool-II 3.6.5 Documentation

The Pgpool Global Development Group

Legal Notice

Table of Contents

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) 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 database servers goes down or becomes unrechable, Pgpool-II will detaches it and continues operations using rest of database servers. There are some sophiscated features which help the automated fail over including timeout and retries.

Replication

> 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

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.

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 Developement 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 descibes 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.

Authentication/Access Controls

In the replication mode or master/slave mode, trust, clear text password, and pam methods are supported. md5 in also supported since Pgpool-II 3.0. md5 is supported by using a authencitaion file pool_passwd. 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.

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

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. For this at least following two directives must be set:

```
master_slave_mode = on
master_slave_sub_mode = 'stream'
```

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_node" 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
---------+----------+-------+--------+-----------+---------+------------+-------------------+-------------------
 0       | /tmp     | 11002 | up     | 0.500000  | primary | 0          | false             | 0
 1       | /tmp     | 11003 | up     | 0.500000  | standby | 0          | true              | 0
(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 Load Balance

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

```
$ pgbench -p 11000 -c 10 -S -T 10 test
starting vacuum...end.
transaction type: <builtin: select only>
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
duration: 10 s
number of transactions actually processed: 148044
latency average = 0.676 ms
tps = 14802.897506 (including connections establishing)
tps = 14810.213749 (excluding connections establishing)

$ psql -p 11000 -c "show pool_nodes" test
 node_id | hostname | port  | status | lb_weight |  role   | select_cnt | load_balance_node | replication_delay
---------+----------+-------+--------+-----------+---------+------------+-------------------+-------------------
 0       | /tmp     | 11002 | up     | 0.500000  | primary | 75152      | true              | 0
 1       | /tmp     | 11003 | up     | 0.500000  | standby | 72893      | false             | 0
(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.

1.4. 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: 2171
latency average = 4.692 ms
tps = 213.147520 (including connections establishing)
tps = 213.258008 (excluding connections establishing)

$ psql -p 11002 -c "SELECT sum(abalance) FROM pgbench_accounts" test
  sum
--------
 192112
(1 row)

$ psql -p 11003 -c "SELECT sum(abalance) FROM pgbench_accounts" test
  sum
--------
 192112
(1 row)
```

Again, the results are identical.

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
---------+----------+-------+--------+-----------+---------+------------+-------------------+-------------------
 0       | /tmp     | 11002 | up     | 0.500000  | primary | 2172       | true              | 0
 1       | /tmp     | 11003 | down   | 0.500000  | standby | 0          | false             | 0
(2 rows)

$ psql -p 11000 -c "SELECT sum(abalance) FROM pgbench_accounts" test
  sum
--------
 192112
(1 row)
```

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
--------
 192112
(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
---------+----------+-------+--------+-----------+---------+------------+-------------------+-------------------
 0       | /tmp     | 11002 | up     | 0.500000  | primary | 2173       | true              | 0
 1       | /tmp     | 11003 | up     | 0.500000  | standby | 0          | false             | 0
(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
---------+----------+-------+--------+-----------+---------+------------+-------------------+-------------------
 0       | /tmp     | 11002 | down   | 0.500000  | standby | 2173       | false             | 0
 1       | /tmp     | 11003 | up     | 0.500000  | primary | 0          | true              | 0
(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 "fail_over_script" 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_clu

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
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 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
---------+----------+-------+--------+-----------+---------+------------+-------------------+-------------------
 0       | /tmp     | 11002 | up     | 0.500000  | standby | 2173       | true              | 0
 1       | /tmp     | 11003 | up     | 0.500000  | primary | 0          | false             | 0
(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 second argument is the target node id. 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

log=$DATADIR_BASE/log/recovery.log

$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
cat > $DEST_CLUSTER/recovery.conf $lt;$lt;REOF
standby_mode          = 'on'
primary_conninfo      = 'port=$PORT user=$PGSUPERUSER'
recovery_target_timeline='latest'
REOF

$psql -p $PORT -c "SELECT pg_stop_backup()" postgres
```

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 2.

Figure 1-1. Process architecure of Pgpool-II

Chapter 2. Watchdog

2.1. 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. However 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.

2.1.1. 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.

2.1.2. 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.

2.1.3. 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.

2.1.4. 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.

2.1.5. 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.

2.1.6. 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.

2.1.7. 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 delegate 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

Lifecheck process is a sub-component of watchdog, its job is to monitor the health of Pgpool-II nodes participating in the watchdog cluster. The Lifecheck 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 lifecheck process only kicks in when all configured watchdog nodes join the cluster and becomes active. If some remote node fails before the Lifecheck become active that failure will not get caught by the lifecheck.

2.2. Integrating external lifecheck 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 lifecheck 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.

2.2.1. Watchdog IPC command packet format

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

Table 2-1. 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

2.2.2. Watchdog IPC result packet format

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

Table 2-2. Watchdog IPC result packet format

Field	Type	Description

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

2.2.3. 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 2-3. 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 watchddog 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

---

2.2.4. 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.

2.2.4.1. 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 returnd 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 lifecheck 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
```

2.3. Restrictions on watchdog

2.3.1. Watchdog restriction with query mode lifecheck

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 neccessary to avoid split-brain, that is, situations where there are multiple active Pgpool-II.

2.3.2. 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.

2.3.3. 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.

2.3.4. 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.

2.4. Architecure 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.

2.4.1. 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.

2.4.1.1. 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.

2.4.1.2. 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 2.2 for details on how to use watchdog IPC interface for integrating external/3rd party systems.

2.4.2. 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 two built-in methods of remote node health checking, "heartbeat" and "query" 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.

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.

II. Server Administration

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

Table of Contents

Chapter 3. Installation of Pgpool-II

3.1. 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.

3.2. 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.

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:

- GNU 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.

3.3. Getting The Source

The Pgpool-II 3.6.5 sources can be obtained from the download section of our website: http://www.pgpool.net. You should get a file named pgpool-II-3.6.5.tar.gz. After you have obtained the file, unpack it:

```
tar xf pgpool-II-3.6.5.tar.gz
```

This will create a directory pgpool-II-3.6.5 under the current directory with the Pgpool-II sources. Change into that directory for the rest of the installation procedure.

3.4. 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.

```
make
make install
```

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

3.5. Installing pgpool_recovery

This is required in all Pgpool-II installation.

```
$ cd pgpool-II-x.x.x/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
```

3.6. 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-x.x.x/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.

3.7. 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-x.x.x/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.

3.8. Setting up pgpool.conf

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

Table 3-1. pgpool.conf samples

Operation mode	Configuration file name
Streaming replication mode	pgpool.conf.sample-stream
Replication mode	pgpool.conf.sample-replication
Master slave mode	pgpool.conf.sample-master-slave
Raw mode	pgpool.conf.sample

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
```

3.9. Installation from RPM

This chapter describes the installation of Pgpool-II from PRM. If you are going to install from the source code, please check Section 3.1.

3.9.1. 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/3.6/redhat/rhel-7-x86_64/pgpool-II-release-3.6-1.noarch.rpm
```

Then:

```
yum install pgpool-II-pg96
```

pg96 means PostgreSQL 9.5. 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-pg96-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-pg96-devel
```

This installs header files which developers are interested in

On all the PostgreSQL servers you need to install:

```
yum install pgpool-II-pg96-extensions
```

3.9.2. Configuration with RPM

All the Pgpool-II configuration files live in /etc/pgpool-II. Please refer to Section 3.8 to see how to set up configuration files.

3.9.3. Starting/stopping Pgpool-II

On RHEL7/CentOS 7, do this once.

```
systemctl enable pgpool.service
```

After this, restart the whole system or:

```
systemctl start pgpool.service
```

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

```
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.

Chapter 4. Server Setup and Operation

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

4.1. 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.

4.2. 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.

4.3. Configuring Pgpool-II

4.3.1. 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.

4.3.2. Running mode of Pgpool-II

There are four different running modes in Pgpool-II: streaming replication mode, master slave mode, native replication mode and raw mode. In any mode, Pgpool-II provides connection pooling, automatic fail over and online recovery. The sample configuration files for each mode are provied. They are located under $prefix/etc. You can copy one of them to $prefix/etc/pgpool.conf.

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 sample configuration file is $prefix/etc/pgpool.conf.sample-stream.

The master slave mode mode can be used with PostgreSQL servers operating Slony. In this mode, Slony/PostgreSQL is responsible for synchronizing databases. Since Slony is being obsoletd 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. The sample configuration file is $prefix/etc/pgpool.conf.sample-master-slave.

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. Note, however, read snapshot control is not done and consistent visibility is not guaranteed in the mode. Load balancing is possible in the mode. The sample configuration file $prefix/etc/pgpool.conf.sample-replication.

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 meaningfull thing. Load balancing is **not** possible in the mode. The sample configuration file $prefix/etc/pgpool.conf.sample.

4.4. 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 paramters are required to be set up to start Pgpool-II server.

4.4.1. Backend Settings

Backend PostgreSQL used by Pgpool-II must be specified in pgpool.conf. See Section 5.4

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. 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**. 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: 1) connecting to Pgpool-II fails 2) 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 max_pool*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 max_connections - superuser_reserved_connections.

In summary, max_pool, num_init_children, max_connections, 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.

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 password file name for md5 authentication. Default value is "pool_passwd". Specifying '' (empty) disables the use of password file. See Section 6.2.2 for more details.

This parameter can only be set at server start.

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

Default is 'slony'.

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
Transaction isolation level is SERIALIZABLE and the transaction has issued a write query	Y	N	ANY	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 desn'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 3.7 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 3.4 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 interger or floating point value greater than or equeal 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 alllow 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	Disllow 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.

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 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. 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 my child process 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 recomended 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 recomended to do a benchmark before deciding wether to use serialize_accept or not, because the corelation 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_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 compatibily with prePgpool-II **V3.4**.

Table 5-5. log_line_prefix escape options

Escape	Effect
%a	Client application name
%p	Process ID (PID)
%P	Process name
%t	Time stamp
%d	Database name
%u	User name
%l	Log line number for each process
%%	'%' 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 primay 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. The only exception is by writing special SQL comments. 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 **/*NO LOAD BALANCE*/** comment 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. SELECTs having write functions as specified in black or white function list is not regarded as a write query. This may be changed in the future.)
        - If black and white function list is empty, SELECTs 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 SELECTs 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 supressing 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. 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
- 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
- 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 the primary 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.

5.7.3. 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 Maste 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_.*'
```

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 Maste 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: 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.

database_redirect_preference_list (string)

Specifies the list of **"database-name:node id"** pairs to send SELECT queries to a particular backend node for a particular database connection. For example, by specifying "test:1", Pgpool-II will redirect all 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.

Example 5-5. 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 all SELECT queries on mydb0 or on mydb1 databases to backend node of ID 1.
- 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,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"** pairs to send SELECT queries to a particular backend node for a particular client application connection.

> **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.

Example 5-6. 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 all SELECT queries from myapp1 client to backend node of ID 1.
- 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,myapp2:standby'
```

> **Note:** app_name_redirect_preference_list takes precedence over the database_redirect_preference_list.

> **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.

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.

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.

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 fail_over_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

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
%D	Database cluster directory of the detached node
%M	Old master node ID
%m	New master node ID

Special character	Description
%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
%%	'%' character

Note: The "master node" referes 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.

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 when a fail-over 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.

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 excuting 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	Old primary node ID
%r	Port number of the new master node
%R	Database cluster directory of the new master node
%%	'%' character

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. This works only in Master Replication mode with streaming replication. Pgpool-II replaces the following special characters with the backend specific information before excuting the command.

Table 5-8. follow master 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
%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
%%	'%' 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 excepted 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.

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

fail_over_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 fail_over_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.

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

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 with-in 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.

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".

For online recovery, the recovery target node must be detached from Pgpool-II. 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.

> **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. Connections from cliens are not allowd only in second phase of online recovery while the data can be updated or retrieved during the first phase. Pgpool-II performs the follows steps in online recovery:

- CHECKPOINT.
- First stage of online recovery.
- Wait until all client connections have disconnected.
- CHECKPOINT.
- Second stage of online recovery.
- Start up postmaster (perform pgpool_remote_start)
- Node attach

> **Note:** There is a restriction in the online recovery. 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.

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 scrit in $PGDATA directory and will try to execute $PGDATA/sync-command.

recovery_1st_stage_command receives following 4 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.

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 prematuraly 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(primary) node at the second stage of online recovery. 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 scrit in $PGDATA directory and will try to execute $PGDATA/sync-command.

recovery_2nd_stage_command receives following 4 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 database to be recovered.

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 prematuraly 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 completes within this time. Since Pgpool-II does not accepts 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).

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 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.

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
'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 SELECTs 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 SELECTs 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 SELECTs 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 be changed by reloading the Pgpool-II configurations.

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 atleast 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 Sockect 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 3.4 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. 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. 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.

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 heatrbeat singals 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.

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. 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.

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" $_IP_$ 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".

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.

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". $_IP_$ will get replaced by the IP address specified in the delegate_IP.

This parameter can only be set at server start.

5.14.5. Behaivor 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 escaltes to active. This prevents the new active Pgpool-II from using old query caches inconsistence 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/floating 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. 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 singals (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

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.

5.14.7. Lifecheck 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_lifecheck_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_lifecheck_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_lifecheck_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 singnals, 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_lifecheck_method is set to 'heartbeat'

This parameter can only be set at server start.

heartbeat_destination_port0 (integer)

Specifies the destination port number for the heartbeat_destinationX of the remote Pgpool-II for sending the heartbeat signals. Multiple destinations can be configured for the heartbeat singnals, 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_lifecheck_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_lifecheck_method is set to 'heartbeat'

This parameter can only be set at server start.

5.14.8. Lifecheck 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_lifecheck_method is set to 'query'

This parameter can only be set at server start.

wd_lifecheck_query (string)

Specifies the query to use for the life check of remote Pgpool-II. Default is "SELECT 1".

wd_lifecheck_query is only applicable if the wd_lifecheck_method is set to 'query'

This parameter can only be set at server start.

wd_lifecheck_dbname (string)

Specifies the database name for the connection used for the life check of remote Pgpool-II. Default is "template1".

wd_lifecheck_dbname is only applicable if the wd_lifecheck_method is set to 'query'

This parameter can only be set at server start.

wd_lifecheck_user (string)

Specifies the user name for the connection used for the life check of remote Pgpool-II. Default is "nobody".

wd_lifecheck_user is only applicable if the wd_lifecheck_method is set to 'query'

This parameter can only be set at server start.

wd_lifecheck_password (string)

Specifies the password for the user used for the life check of remote Pgpool-II. Default is ''(empty).

wd_lifecheck_password is only applicable if the wd_lifecheck_method is set to 'query'

This parameter can only be set at server start.

5.14.9. 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 informations 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. 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.

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.

check_temp_table (boolean)

Setting to on, 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, which increases the load on the primary/master server. If you are absolutely sure that your system never uses temporary tables, then you can safely turn off the check_temp_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.

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 pool_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 inturn connects to the PostgreSQL servers using the same credentials to serve the incomming 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 hostnames. 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 incomming 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. Pgpool-II installation 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.

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  CIDR-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.

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 databaseE 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. 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.

> **Note:** Pv6 IP address/mask is currently not supported

This field only applies to host 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". See Section 6.2.2 for more details.

pam

Authenticate using the Pluggable Authentication Modules (PAM) service provided by the operating system. See Section 6.2.3 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.

6.2.2.1. Authentication file format

This pool_passwd file should contain lines in the following format:

```
"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= " 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. 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.pam" under the install directory.

> **Note:** To enable PAM support the Pgpool-II must be configured with "--with-pam"

III. Examples

Various examples

Table of Contents

Chapter 7. Configuration Examples

7.1. Basic Configuration Example

7.1.1. Let's Begin!

First, we must learn how to install and configure Pgpool-II and database nodes before using replication.

7.1.1.1. 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.

7.1.1.2. 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. If you wish to receive conenctions from other hosts, set listen_addresses to '*'.

```
listen_addresses = 'localhost'
port = 9999
```

We will use the default parameters in thie tutorial.

7.1.1.3. 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/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
```

7.1.1.4. 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.

7.1.1.5. 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
```

7.1.2. 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 7.1.1, 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.

7.1.2.1. 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.

7.1.2.2. Checking Replication

To reflect the changes in pgpool.conf, Pgpool-II must be restarted. Please refer to "Starting/Stopping Pgpool-II" Section 7.1.1.5. 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 7-1. data summary

Table Name	Number of Rows
branches	1
tellers	10
accounts	100000
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 branches, tellers, accounts, and history tables on all the nodes (5432, 5433, 5434).

```
$ for port in 5432 5433 5434; do
>     echo $port
>     for table_name in branches tellers accounts history; do
>         echo $table_name
>         psql -c "SELECT count(*) FROM $table_name" -p $port bench_replication
>     done
> done
```

7.2. Watchdog Configuration Example

This tutrial 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 use "osspc16" as an Active node and "osspc20" as a Standby node. "Someserver" means one of them.

7.2.1. Common configurations

Set the following parameters in both of active and standby nodes.

7.2.1.1. Enabling watchdog

First of all, set use_watchdog to on.

```
use_watchdog = on
                            # Activates watchdog
```

7.2.1.2. 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,...)
```

7.2.1.3. Watchdog Communication

Specify the TCP port number for watchdog communication.

```
wd_port = 9000
                            # port number for watchdog service
```

7.2.1.4. 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.

7.2.2. 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.

7.2.2.1. 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 othet pgpool 0
other_wd_port0 = 9000
                            # Port number for othet watchdog 0
```

7.2.2.2. 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 othet pgpool 0
other_wd_port0 = 9000
                        # Port number for othet watchdog 0
```

7.2.3. Starting Pgpool-II

Start Pgpool-II on each servers from root user with "-n" switch and redirect log messages into pgpool.log file.

7.2.3.1. 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
```

7.2.3.2. 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 joind the watchdog cluster as standby watchdog.

```
LOG:  watchdog cluster configured with 1 remote nodes
LOG:  watchdog remote node:0 on Linux_osspc16_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 outbond connection to Linux_osspc16_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_osspc16_9000"
LOG:  watchdog process is initialized
```

7.2.4. 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@osspc16]# 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@osspc20]# 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
```

7.2.5. 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@osspc16]# {installed_dir}/bin/pgpool stop
```

Then, the Standby server starts to use the virtual IP address. Log shows:

```
LOG:  remote node "Linux_osspc16_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:  skiping 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@osspc16]# ifconfig
eth0      ...

lo        ...
```

Confirm that the Standby server uses the virtual IP address.

```
[root@osspc20]# 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
```

7.2.6. More

7.2.6.1. Lifecheck

There are the parameters about watchdog's monitoring. Specify the interval to check wd_interval, the count to retry wd_life_point, the qyery to check wd_lifecheck_query and finaly the type of lifecheck wd_lifecheck_method.

```
wd_lifecheck_method = 'query'
                            # Method of watchdog lifecheck ('heartbeat' or 'query' or 'external')
                            # (change requires restart)
wd_interval = 10
                            # lifecheck interval (sec) > 0
wd_life_point = 3
                            # lifecheck retry times
wd_lifecheck_query = 'SELECT 1'
                            # lifecheck query to pgpool from watchdog
```

7.2.6.2. 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.

```
ifconfig_path = '/sbin'
                        # ifconfig command path
if_up_cmd = 'ifconfig eth0:0 inet $_IP_$ netmask 255.255.255.0'
                        # startup delegate IP command
if_down_cmd = 'ifconfig eth0:0 down'
                        # shutdown delegate IP command

arping_path = '/usr/sbin'          # arping command path

arping_cmd = 'arping -U $_IP_$ -w 1'
```

You can also use the custom scripts to bring up and bring down the virtual IP using wd_escalation_command and wd_de_escalation_command configurations.

7.3. AWS Configuration Example

This tutrial 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.

7.3.1. 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"

7.3.2. Pgpool-II configurations

Mostly the Pgpool-II configurations for this example will be same as in the Section 7.2, 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 maste/Active Pgpool-II node.

7.3.2.1. 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'
```

7.3.2.2. 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'
```

7.3.3. 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 configure AWS CLI

7.3.3.1. 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
```

7.3.3.2. 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**.

7.3.4. 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
```

7.3.5. 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:  skiping 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
```

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

This part contains reference information for server commands. Currently only pgpool falls into this category.

Table of Contents

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.

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.

-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 realod 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. -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

Name

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 0
host1 5432 1 1073741823.500000
```

The result is in the following order:

```
1. hostname
2. port number
3. status
4. load balance weight
```

Status is represented by a digit from [0 to 3].

- 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 0
Hostname: host1
Port    : 5432
Status  : 1
Weight  : 0.5
```

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 *watcgdog_id*
--node-id=*watcgdog_id*

The index of other Pgpool-II to get information for.

Index 0 gets one's self watchdog information.

If ommitted 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**...] [**processid**]

Description

pcp_proc_info displays the information on the given Pgpool-II child process ID.

Options

-P *PID*
--process-id=*PID*

PID of Pgpool-II child process.

Other options

See pcp_common_options.

Example

Here is an example output:

```
$ pcp_proc_info -h localhost -p 9898 -U postgres 3815
postgres_db postgres 1150769932 1150767351 3 0 1 1467 1
postgres_db postgres 1150769932 1150767351 3 0 1 1468 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 conncted 0 if not

If 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 --verbose -U postgres 3815
Database     : postgres_db
Username     : postgres
Start time   : 1150769932
Creation time: 1150767351
Major        : 3
Minor        : 0
Counter      : 1
PID          : 1467
Connected    : 1
Database     : postgres_db
Username     : postgres
Start time   : 1150769932
Creation time: 1150767351
Major        : 3
Minor        : 0
Counter      : 1
PID          : 1468
Connected    : 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. Exisiting 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. Exisiting 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_node_info attaches the given backend node with recovery.

Options

-n *node_id*
--node-id=*node_id*

The index of backend node.

Other options

See pcp_common_options.

See pcp_common_options.

III. Other commands

This part contains reference information for various Pgpool-II commands.

Table of Contents

pg_md5

Name

pg_md5 -- produces encrypted password in md5

Synopsis

pg_md5 [**option**...]

pg_md5 [**password**]

Description

pg_md5 produces encrypted password in md5.

Options

-p
--prompt

Prompt password using standard input.

-m
--md5auth

Produce md5 authentication password.

-u *your_username*
--username=*your_username*

When producing a md5 authentication password, create the pool_passwd entry for your_username.

Example

Here is an example output:

```
pg_md5 -p
password: [your password]
```

or

```
./pg_md5 foo
acbd18db4cc2f85cedef654fccc4a4d8
```

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 and raw 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), 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.

-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.

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".

Example

```
$ pgpool_setup
Satrting set up in streaming replication mode
creating startall and shutdownall
creating failover script
creating database cluster /home/t-ishii/tmp/test/data0...done.
update postgreql.conf
creating pgpool_remote_start
creating basebackup.sh
creating recovery.conf
creating database cluster /home/t-ishii/tmp/test/data1...done.
update postgreql.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)
```

watchdog_setup

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 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 u 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
Satrting set up
============== setting up pgpool 0 ==============
Satrting 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 postgreql.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 postgreql.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          | 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 ==============
Satrting 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 postgreql.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 postgreql.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 ==============
Satrting 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 postgreql.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 postgreql.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

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.

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

```
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

```
RESET configuration_parameter
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

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 bakance node or not, and the replication delay (only if in streaming replication mode). 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 Pgoool-II. Also the counters are reset to zero upon starting up of Pgpool-II.

Here is an example session:

```
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)
```

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.

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_co
----------+---------------------+---------+------------+----------+----------+---------------------+--------------+--------------+---------
 19696    | 2016-10-17 13:24:17 | 0       | 0          | postgres | t-ishii  | 2016-10-17 13:35:12 | 3            | 0            | 1       | 20079
 19696    | 2016-10-17 13:24:17 | 0       | 1          | postgres | t-ishii  | 2016-10-17 13:35:12 | 3            | 0            | 1       | 20080
 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
 19700    | 2016-10-17 13:24:17 | 0       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19700    | 2016-10-17 13:24:17 | 0       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19700    | 2016-10-17 13:24:17 | 1       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19700    | 2016-10-17 13:24:17 | 1       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19700    | 2016-10-17 13:24:17 | 2       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19700    | 2016-10-17 13:24:17 | 2       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19700    | 2016-10-17 13:24:17 | 3       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19700    | 2016-10-17 13:24:17 | 3       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19701    | 2016-10-17 13:24:17 | 0       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19701    | 2016-10-17 13:24:17 | 0       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19701    | 2016-10-17 13:24:17 | 1       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19701    | 2016-10-17 13:24:17 | 1       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19701    | 2016-10-17 13:24:17 | 2       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19701    | 2016-10-17 13:24:17 | 2       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19701    | 2016-10-17 13:24:17 | 3       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19701    | 2016-10-17 13:24:17 | 3       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19702    | 2016-10-17 13:24:17 | 0       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19702    | 2016-10-17 13:24:17 | 0       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19702    | 2016-10-17 13:24:17 | 1       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19702    | 2016-10-17 13:24:17 | 1       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19702    | 2016-10-17 13:24:17 | 2       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19702    | 2016-10-17 13:24:17 | 2       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19702    | 2016-10-17 13:24:17 | 3       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19702    | 2016-10-17 13:24:17 | 3       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19703    | 2016-10-17 13:24:17 | 0       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19703    | 2016-10-17 13:24:17 | 0       | 1          |          |          |                     | 0            | 0            | 0       | 0
 19703    | 2016-10-17 13:24:17 | 1       | 0          |          |          |                     | 0            | 0            | 0       | 0
 19703    | 2016-10-17 13:24:17 | 1       | 1          |          |          |                     | 0            | 0            | 0       | 0
```

19703	2016-10-17 13:24:17	2	0				0	0	0	0	0
19703	2016-10-17 13:24:17	2	1				0	0	0	0	0
19703	2016-10-17 13:24:17	3	0				0	0	0	0	0
19703	2016-10-17 13:24:17	3	1				0	0	0	0	0
19704	2016-10-17 13:24:17	0	0				0	0	0	0	0
19704	2016-10-17 13:24:17	0	1				0	0	0	0	0
19704	2016-10-17 13:24:17	1	0				0	0	0	0	0
19704	2016-10-17 13:24:17	1	1				0	0	0	0	0
19704	2016-10-17 13:24:17	2	0				0	0	0	0	0
19704	2016-10-17 13:24:17	2	1				0	0	0	0	0
19704	2016-10-17 13:24:17	3	0				0	0	0	0	0
19704	2016-10-17 13:24:17	3	1				0	0	0	0	0
19705	2016-10-17 13:24:17	0	0				0	0	0	0	0
19705	2016-10-17 13:24:17	0	1				0	0	0	0	0
19705	2016-10-17 13:24:17	1	0				0	0	0	0	0
19705	2016-10-17 13:24:17	1	1				0	0	0	0	0
19705	2016-10-17 13:24:17	2	0				0	0	0	0	0
19705	2016-10-17 13:24:17	2	1				0	0	0	0	0
19705	2016-10-17 13:24:17	3	0				0	0	0	0	0
19705	2016-10-17 13:24:17	3	1				0	0	0	0	0
19706	2016-10-17 13:24:17	0	0				0	0	0	0	0
19706	2016-10-17 13:24:17	0	1				0	0	0	0	0
19706	2016-10-17 13:24:17	1	0				0	0	0	0	0
19706	2016-10-17 13:24:17	1	1				0	0	0	0	0
19706	2016-10-17 13:24:17	2	0				0	0	0	0	0
19706	2016-10-17 13:24:17	2	1				0	0	0	0	0
19706	2016-10-17 13:24:17	3	0				0	0	0	0	0
19706	2016-10-17 13:24:17	3	1				0	0	0	0	0
19707	2016-10-17 13:24:17	0	0				0	0	0	0	0
19707	2016-10-17 13:24:17	0	1				0	0	0	0	0
19707	2016-10-17 13:24:17	1	0				0	0	0	0	0
19707	2016-10-17 13:24:17	1	1				0	0	0	0	0
19707	2016-10-17 13:24:17	2	0				0	0	0	0	0
19707	2016-10-17 13:24:17	2	1				0	0	0	0	0
19707	2016-10-17 13:24:17	3	0				0	0	0	0	0
19707	2016-10-17 13:24:17	3	1				0	0	0	0	0
19708	2016-10-17 13:24:17	0	0				0	0	0	0	0
19708	2016-10-17 13:24:17	0	1				0	0	0	0	0
19708	2016-10-17 13:24:17	1	0				0	0	0	0	0
19708	2016-10-17 13:24:17	1	1				0	0	0	0	0
19708	2016-10-17 13:24:17	2	0				0	0	0	0	0
19708	2016-10-17 13:24:17	2	1				0	0	0	0	0
19708	2016-10-17 13:24:17	3	0				0	0	0	0	0
19708	2016-10-17 13:24:17	3	1				0	0	0	0	0
19709	2016-10-17 13:24:17	0	0				0	0	0	0	0
19709	2016-10-17 13:24:17	0	1				0	0	0	0	0
19709	2016-10-17 13:24:17	1	0				0	0	0	0	0
19709	2016-10-17 13:24:17	1	1				0	0	0	0	0
19709	2016-10-17 13:24:17	2	0				0	0	0	0	0
19709	2016-10-17 13:24:17	2	1				0	0	0	0	0
19709	2016-10-17 13:24:17	3	0				0	0	0	0	0
19709	2016-10-17 13:24:17	3	1				0	0	0	0	0
19710	2016-10-17 13:24:17	0	0				0	0	0	0	0
19710	2016-10-17 13:24:17	0	1				0	0	0	0	0
19710	2016-10-17 13:24:17	1	0				0	0	0	0	0
19710	2016-10-17 13:24:17	1	1				0	0	0	0	0
19710	2016-10-17 13:24:17	2	0				0	0	0	0	0
19710	2016-10-17 13:24:17	2	1				0	0	0	0	0
19710	2016-10-17 13:24:17	3	0				0	0	0	0	0
19710	2016-10-17 13:24:17	3	1				0	0	0	0	0
19711	2016-10-17 13:24:17	0	0				0	0	0	0	0
19711	2016-10-17 13:24:17	0	1				0	0	0	0	0
19711	2016-10-17 13:24:17	1	0				0	0	0	0	0
19711	2016-10-17 13:24:17	1	1				0	0	0	0	0
19711	2016-10-17 13:24:17	2	0				0	0	0	0	0
19711	2016-10-17 13:24:17	2	1				0	0	0	0	0
19711	2016-10-17 13:24:17	3	0				0	0	0	0	0
19711	2016-10-17 13:24:17	3	1				0	0	0	0	0
19712	2016-10-17 13:24:17	0	0				0	0	0	0	0
19712	2016-10-17 13:24:17	0	1				0	0	0	0	0
19712	2016-10-17 13:24:17	1	0				0	0	0	0	0
19712	2016-10-17 13:24:17	1	1				0	0	0	0	0
19712	2016-10-17 13:24:17	2	0				0	0	0	0	0
19712	2016-10-17 13:24:17	2	1				0	0	0	0	0
19712	2016-10-17 13:24:17	3	0				0	0	0	0	0
19712	2016-10-17 13:24:17	3	1				0	0	0	0	0
19713	2016-10-17 13:24:17	0	0				0	0	0	0	0

19713	2016-10-17 13:24:17	0	1				0	0	0	0	0
19713	2016-10-17 13:24:17	1	0				0	0	0	0	0
19713	2016-10-17 13:24:17	1	1				0	0	0	0	0
19713	2016-10-17 13:24:17	2	0				0	0	0	0	0
19713	2016-10-17 13:24:17	2	1				0	0	0	0	0
19713	2016-10-17 13:24:17	3	0				0	0	0	0	0
19713	2016-10-17 13:24:17	3	1				0	0	0	0	0
19714	2016-10-17 13:24:17	0	0				0	0	0	0	0
19714	2016-10-17 13:24:17	0	1				0	0	0	0	0
19714	2016-10-17 13:24:17	1	0				0	0	0	0	0
19714	2016-10-17 13:24:17	1	1				0	0	0	0	0
19714	2016-10-17 13:24:17	2	0				0	0	0	0	0
19714	2016-10-17 13:24:17	2	1				0	0	0	0	0
19714	2016-10-17 13:24:17	3	0				0	0	0	0	0
19714	2016-10-17 13:24:17	3	1				0	0	0	0	0
19715	2016-10-17 13:24:17	0	0				0	0	0	0	0
19715	2016-10-17 13:24:17	0	1				0	0	0	0	0
19715	2016-10-17 13:24:17	1	0				0	0	0	0	0
19715	2016-10-17 13:24:17	1	1				0	0	0	0	0
19715	2016-10-17 13:24:17	2	0				0	0	0	0	0
19715	2016-10-17 13:24:17	2	1				0	0	0	0	0
19715	2016-10-17 13:24:17	3	0				0	0	0	0	0
19715	2016-10-17 13:24:17	3	1				0	0	0	0	0
19716	2016-10-17 13:24:17	0	0				0	0	0	0	0
19716	2016-10-17 13:24:17	0	1				0	0	0	0	0
19716	2016-10-17 13:24:17	1	0				0	0	0	0	0
19716	2016-10-17 13:24:17	1	1				0	0	0	0	0
19716	2016-10-17 13:24:17	2	0				0	0	0	0	0
19716	2016-10-17 13:24:17	2	1				0	0	0	0	0
19716	2016-10-17 13:24:17	3	0				0	0	0	0	0
19716	2016-10-17 13:24:17	3	1				0	0	0	0	0
19717	2016-10-17 13:24:17	0	0				0	0	0	0	0
19717	2016-10-17 13:24:17	0	1				0	0	0	0	0
19717	2016-10-17 13:24:17	1	0				0	0	0	0	0
19717	2016-10-17 13:24:17	1	1				0	0	0	0	0
19717	2016-10-17 13:24:17	2	0				0	0	0	0	0
19717	2016-10-17 13:24:17	2	1				0	0	0	0	0
19717	2016-10-17 13:24:17	3	0				0	0	0	0	0
19717	2016-10-17 13:24:17	3	1				0	0	0	0	0
19718	2016-10-17 13:24:17	0	0				0	0	0	0	0
19718	2016-10-17 13:24:17	0	1				0	0	0	0	0
19718	2016-10-17 13:24:17	1	0				0	0	0	0	0
19718	2016-10-17 13:24:17	1	1				0	0	0	0	0
19718	2016-10-17 13:24:17	2	0				0	0	0	0	0
19718	2016-10-17 13:24:17	2	1				0	0	0	0	0
19718	2016-10-17 13:24:17	3	0				0	0	0	0	0
19718	2016-10-17 13:24:17	3	1				0	0	0	0	0
19719	2016-10-17 13:24:17	0	0				0	0	0	0	0
19719	2016-10-17 13:24:17	0	1				0	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	1	22055	
20024	2016-10-17 13:33:46	0	1	test	t-ishii	2016-10-17 14:30:53	3	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_enrties_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 fowards 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 extraced 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

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);

pcp_node_info returns record(integer node_id, text pcp_server, out status text, out weight float4);

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(0,'',11001,'t-ishii','t-ishii');
 host | port  |      status       | weight
------+-------+-------------------+--------
 /tmp | 11002 | Connection in use |      0
(1 row)
```

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

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 3.6.5

Release Date: 2017-07-11

A.1.1. 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 posible hang with streaming replication and extended protocol. (Yugo Nagata)

  This hang occured 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 commiting 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 followeing 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)

  errlang 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.2. Release 3.6.4

Release Date: 2017-05-11

A.2.1. 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.3. Release 3.6.3

Release Date: 2017-04-28

A.3.1. 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_vatiable.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.4. Release 3.6.2

Release Date: 2017-03-17

A.4.1. 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 fixe 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-IIshould 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-IIexits. (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 connecttions. (Yugo Nagata)

A.5. Release 3.6.1

Release Date: 2016-12-26

A.5.1. 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. Pgool-II forward it to backend 1
  4. Frontend sends Parse, Bind, Execute of COMMIT
  5. Pgool-II forward it to backend 0 & 1
  6. Frontend sends sync message
  7. Pgool-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.6. Release 3.6

Release Date: 2016-11-21

A.6.1. Overview

Major enhancements in Pgpool-II 3.6 include:

- Improve the behavior of fail-over. In the steaming replication mode, client sessions will not be disconnected when a fail-over 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 fail-over.

- Change documentation format from raw HTML to SGML.

The above items are explained in more detail in the sections below.

A.6.2. Major Enhancements

- Improve the behavior of fail-over. (Tatsuo Ishii)

  In the steaming replication mode, client sessions will not be disconnected when a fail-over 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 fail-over. If the load balance node is not the failed node, the session will not be affected by fail-over.

  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 fail_over_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 fail-over. 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 inconsitent 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 fail-over. (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.6.3. 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". (Tatstuo 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 Pacejo, Tatsuo Ishii)

  This commit tries to remove the data inconsitency 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 Pacejo.

- 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.6.4. 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. (Tatstuo 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.6.5. 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 becausePgpool-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 inavailability. (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 fail_over_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 posible hang during health checking. (bug 204) (Yugo Nagata)

  Helath 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 checkking 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 documetation bug about raw mode. (Yugo Nagata, Bo Peng)

  Connection pool is avalilable 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 happend if we execute an extended protocol query and it fails.

- Fix query stack problems in extended protocol case. (bug 167, 168) (Tatstuo 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 distingushable for each other.

A.7. Release 3.5.9

Release Date: 2017-07-11

A.7.1. 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 posible hang with streaming replication and extended protocol (Yugo Nagata)

  This hang occured 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 commiting 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 followeing 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)

  errlang 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.8. Release 3.5.8

Release Date: 2017-05-11

A.8.1. Bug fixes

- Add node 0 failover test. (Tatsuo Ishii)
- Fix Pgpool-II child process segfault reported in [pgpool-hackers: 2312]. (Tatsuo Ishii)

A.9. Release 3.5.7

Release Date: 2017-04-28

A.9.1. 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.10. Release 3.5.6

Release Date: 2017-03-17

A.10.1. 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 fixe 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-IIshould 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-IIexits. (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 connecttions. (Yugo Nagata)

A.11. Release 3.5.5

Release Date: 2016-12-26

A.11.1. 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. Pgool-II forward it to backend 1
4. Frontend sends Parse, Bind, Execute of COMMIT
5. Pgool-II forward it to backend 0 & 1
6. Frontend sends sync message
7. Pgool-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.12. Release 3.4.12

Release Date: 2017-07-11

A.12.1. 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.13. Release 3.4.11

Release Date: 2017-04-28

A.13.1. 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.14. Release 3.4.10

Release Date: 2017-03-17

A.14.1. 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 fixe 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-IIshould 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-IIexits. (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 connecttions. (Yugo Nagata)

A.15. Release 3.4.9

Release Date: 2016-12-26

A.15.1. 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.16. Release 3.3.16

Release Date: 2017-07-11

A.16.1. 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.17. Release 3.3.15

Release Date: 2017-04-28

A.17.1. 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.18. Release 3.3.14

Release Date: 2017-03-17

A.18.1. 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 fixe 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-IIexits. (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 connecttions. (Yugo Nagata)

A.19. Release 3.3.13

Release Date: 2016-12-26

A.19.1. 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.20. Release 3.2.21

Release Date: 2017-07-11

A.20.1. 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.21. Release 3.2.20

Release Date: 2017-04-28

A.21.1. Bug fixes

- Fix for 0000299: Errors on the reloading of configuration. (Bug 299) (Muhammad Usama)

A.22. Release 3.2.19

Release Date: 2017-03-17

A.22.1. 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 fixe 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-IIexits. (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 connecttions. (Yugo Nagata)

A.23. Release 3.2.18

Release Date: 2016-12-26

A.23.1. 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.24. Release 3.1.21

Release Date: 2016-12-26

A.24.1. 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

R

S

T

U

W

Y