Pgpool-II for beginners

Table of Contents

Original text written by Gerd Koenig

please send comments to: gk_ulm at web.de

---

prerequisite

•postgresql headerfiles (postgresql-devel-8.3.5-2.1.x86_64.rpm)
•libpq.so (check /usr/lib64/libpq*)
•make,gcc

introduction

•There are 2 servers, identical hardware, shared nothing with installed OpenSuse10.3 (64bit).
•existing ssh-key-exchange from node1 → node2
•PostgreSQL8.3.5, PGDATA = /opt/postgres/data
•In this guide pgpool will be installed under directory /opt/pgpoolII, adjust the parameter „--prefix“ if you want to install to a different location.
•This tutorial covers the usage of replication mode only !
•I strongly recommend to read the documentation at http://pgpool.projects.postgresql.org/ to get as much information as possible about pgpool
•Please keep in mind that the pgpool instance is a single-point-of-failur (SPOF) and should be made high available with an appropriate tool like heartbeat.

installation

Check if you have proper rights to call „make install“, or just perform it as root.
cd /incoming/
tar -xzf pgpool-II-2.1.tar.gz
cd pgpool-II-2.1/
*./configure *
*--prefix=/opt/pgpoolII *
*--with-pgsql-libdir=/usr/lib64/ *
--with-pgsql-includedir=/usr/include/pgsql/
make && make install
[sudo chown -R postgres /opt/pgpoolII]

configuration

cd /opt/pgpoolII/
cp etc/pcp.conf.sample etc/pcp.conf
cp etc/pgpool.conf.sample etc/pgpool.conf
vi etc/pcp.conf
set user:pw, password in MD5 format
create password in MD5 format:
/opt/pgpoolII/bin/pg_md5 -p
password:<type password>
e8a48653851e28c69d0506508fb27fc5

vi etc/pgpool.conf
Here you can define ports for pgpool,pgpool communication manager, listen addresses and a lot of other things. As mentioned earlier I'll focus on enabling replication mode and define both database backends.

```
...
listen_addresses ='*'
...
replication_mode = true
...
backend_hostname0 = node01'
backend_port0 = 5432
backend_weight0 = 1
backend_data_directory0 = '/opt/postgres/data'
backend_hostname1 = 'node02'
backend_port1 = 5432
backend_weight1 = 1
backend_data_directory1 = '/opt/postgres/data'
```

start/stop pgpool

Start pgpool by calling the binary and, dependent on your needs, add some parameters:
/opt/pgpoolII/bin/pgpool -d -n > /opt/pgpoolII/log/pgpool.log 2>&1 &
Parameters:
[*-d* => verbose debug]
[*-n* => detach terminal]
[*-f <configfile>* => if the config file isn't inside the default path $PREFIX/etc]

Stop pgpool by calling the binary with subsequent command „stop“
/opt/pgpoolII/bin/pgpool stop

initial replication setup

copy database

The following steps requires an established session to node1 as user postgres. PostgreSQL is running on node1 and configured with enabled wal archiving. If you don't have an existing database you want to replicate, just skip this section.

```
psql -U postgres -d testdb
  -> select pg_start_backup('initial_backup');
  -> \q
cd /opt/postgres
rsync -avz ./data/* postgres@node2:/opt/postgres/data/
psql -U postgres -d testdb
  -> select pg_stop_backup();
  -> \q
```

starting up

Finally we can start PostgreSQL on each node and afterwards pgpool-II.
Therefore login to every backend and call:
as user root: */etc/init.d/postgres start*
or as user postgres: *pg_ctl -D /opt/postgres/data start*

on node1 as user postgres:
/opt/pgpoolII/bin/pgpool -d -n > /opt/pgpoolII/log/pgpool.log 2>&1 &

=> check log file if pgpool started correctly and all the defined backends are enabled. I strongly recommend to enable debug output (-d switch) at this stage.

check if replication is working

Initial situation

•postgres is running on all nodes
•pgpool is running on port 9999 on node 1
•shell session on node1 established

create a test database and insert some data

Perform the following steps to create database „bench_replication“, insert some base data and insert a bunch of rows to table „history“.

createdb -p 9999 bench_replication
pgbench -i -p 9999 bench_replication
psql -p 9999 bench_replication
bench_replication=# insert into history (tid, bid,aid,mtime,filler) (select 1,1,1,now(),i::text from (select generate_series(1,1000000) as i) as q);

check data in each node

You can check if the databases are in sync with a simple shell script, which connects to each node and fires a select query there.

e.g.:
```
#!/bin/bash
for host in node1 node2; do
  for table_name in accounts history; do
     echo $host: $table_name
     psql -c "SELECT count(*) FROM $table_name" -h $host -p 5432 bench_replication
  done
done
```

failure example

To simulate a failure scenario, I just killed postgres processes on node2 while an update statement is running (I've chosen an update on table history because we inserted a lot of data in the previous step there).

You need at least two shell sessions, one at node1 to fire the update/insert statement, and one session on node2 to kill the processes.

Node1:
psql -p 9999 bench_replication
update history set tid=2;

Node2:
pkill postgres

log entries in pgpool.log
LOG: pid 2088: statement: update history set tid=2;
DEBUG: pid 2088: do_command: Query: BEGIN
DEBUG: pid 2088: command tag: BEGIN
DEBUG: pid 2088: ReadyForQuery: transaction state: T
DEBUG: pid 2088: do_command: Query: BEGIN
DEBUG: pid 2088: command tag: BEGIN
DEBUG: pid 2088: ReadyForQuery: transaction state: T
DEBUG: pid 2088: waiting for backend 0 completing the query
DEBUG: pid 2088: waiting for backend 1 completing the query
DEBUG: pid 2088: read_kind_from_backend: read kind from 0 th backend C NUM_BACKENDS: 2
DEBUG: pid 2088: read_kind_from_backend: read kind from 1 th backend E NUM_BACKENDS: 2
ERROR: pid 2088: pool_process_query: 1 th kind E does not match with master connection kind C
LOG: pid 2088: do_child: exits with status 1 due to error
DEBUG: pid 2089: I am 2089 accept fd 0
LOG: pid 2089: connection received: host=[local]
DEBUG: pid 2089: Protocol Major: 3 Minor: 0 database: bench_replication user: postgres
LOG: pid 2089: connection closed. retry to create new connection pool.
DEBUG: pid 2089: new_connection: connecting 0 backend
DEBUG: pid 2089: new_connection: connecting 1 backend
DEBUG: pid 2079: reap_handler called
DEBUG: pid 2079: reap_handler: call wait3
DEBUG: pid 2079: child 2088 exits with status 256 by signal 0
ERROR: pid 2089: connect_inet_domain_socket: connect() failed: Connection refused
ERROR: pid 2089: connection to node2(5432) failed
ERROR: pid 2089: new_connection: create_cp() failed
LOG: pid 2089: notice_backend_error: 1 fail over request from pid 2089
DEBUG: pid 2340: I am 2340
DEBUG: pid 2079: fork a new child pid 2340
DEBUG: pid 2079: child 2089 exits with status 256 by signal 0
DEBUG: pid 2341: I am 2341

DEBUG: pid 2079: fork a new child pid 2341
DEBUG: pid 2079: reap_handler: normally exited
DEBUG: pid 2079: failover_handler called
DEBUG: pid 2079: failover_handler: starting to select new master node
LOG: pid 2079: starting degeneration. shutdown host node2(5432)
LOG: pid 2079: failover_handler: do not restart pgpool. same master node 0 was selected
LOG: pid 2079: failover done. shutdown host node2(5432)
DEBUG: pid 2079: reap_handler called
DEBUG: pid 2079: reap_handler: call wait3
DEBUG: pid 2079: reap_handler: normally exited

Now we can have a closer look how to handle a failover and how to get back to replication mode with databases in sync. To reduce the manual steps pgpool offers „ONLINE RECOVERY“ mechanism, see chapter Online recovery

Online recovery

This mechanism can be used for both attaching a node after failover and attaching a new node.
A node has been detached automatically after failover, and a complete new node is in detached state also after defining the backend_XYZ parameters in pgpool.conf and reloading pgpool.
Database copy and recovery will be covered by PostgreSQL's PITR functionality. Various scripts are needed to complete the recovery successfully and should be located under directory $PGDATA. The sample scripts provided in the source tarball are a good starting point and I'll use them later on and adapt them to this tutorial environment.
An overview over the different steps included in online recovery is shown in drawing online_recovery_theory.

enable online recovery

To be able to use online recovery an additional function for template1 is needed, therefore the following steps should be performed on every database node:

```
cd /incoming/pgpool-II-2.1/sql/pgpool-recovery/
make install
psql -f pgpool-recovery.sql template1
```

In the next step we have to set the parameters in pgpool.conf to enable the recovery commands and the health check. In this tutorial I use the failover-/failback commands just for creating a text file and log a message there. You can define whatever command/script you want instead. The values of the recovery_XYZ_command parameters are names of scripts, we'll have a look at them in the next step.
My settings are as follows:

```
...
health_check_period = 30     # check every 30s
...
failover_command = 'echo host:%h, new master id:%m, old master id:%M >
/opt/pgpoolII/log/failover.log'
failback_command = 'echo host:%h, new master id:%m, old master id:%M >
/opt/pgpoolII/log/failback.log'
...
recovery_user = 'postgres'
recovery_1st_stage_command = 'copy_base_backup'
recovery_2nd_stage_command = 'pgpool_recovery_pitr'
```

You'll have to reload pgpool to reflect the changes.

creating scripts used by online recovery

In this step we have to create the scripts which will be called at the two recovery stages, and we've defined in the previous step. Copy the sample scripts from */incoming/pgpool-II-2.1/sample* to $PGDATA (*/opt/postgres/data* in this case)

copy_base_backup

This script creates a checkpoint on node1 and copies the directory $PGDATA from node1 to node2. Additionally the file recovery.conf will be created (and copied) to enable PostgreSQL recovering to the latest level at startup on node2.

Enabled wal archiving on node1:

archive_command = 'rsync %p postgres@node2:/exchange/wal/%f </dev/null'

source of script copy_base_backup:

```
#! /bin/sh
psql -c "select pg_start_backup('pgpool-recovery')" postgres
echo "restore_command = 'cp /exchange/wal/%f %p'" > /opt/postgres/data/recovery.conf
tar -C /opt/postgres/data -zcf pgsql.tar.gz base global pg_clog pg_multixact pg_subtrans pg_tblspc
pg_twophase pg_xlog recovery.conf
psql -c 'select pg_stop_backup()' postgres
scp pgsql.tar.gz node2:/opt/postgres/data
```

pgpool_recovery_pitr

This file performs a switch_xlog command on node1 to flush the latest transactions from buffer to disk. It's simple source is:

```
#! /bin/sh
psql -c 'select pg_switch_xlog()' postgres
```

pgpool_remote_start

After copying the database files to node2 the database on this node has to be started. Since there's the file recovery.conf in the $PGDATA directory postmaster will recover the database to the latest known state and start it afterwards.

Here we go:

```
#! /bin/sh
if [ $# -ne 2 ]
then
   echo "pgpool_remote_start remote_host remote_datadir"
   exit 1
fi
DEST=$1
DESTDIR=$2
```

```
PGCTL=/usr/bin/pg_ctl
# Expand a base backup
ssh -T $DEST 'cd /opt/postgres/data; tar zxf pgsql.tar.gz' 2>/dev/null 1>/dev/null < /dev/null
# Startup PostgreSQL server
ssh -T $DEST $PGCTL -w -D $DESTDIR start 2>/dev/null 1>/dev/null < /dev/null &
```

initiate online recovery

Let's see how we can start the online recovery automatism and how the scripts from the previous step will be used.
To re-attach a failed node you simply have to call *pcp_recovery_node* command. It is located in the bin directory of the pgpool installation.

```
/opt/pgpoolII/bin/pcp_recovery_node 20 node1 9898 postgres postgres 1
```

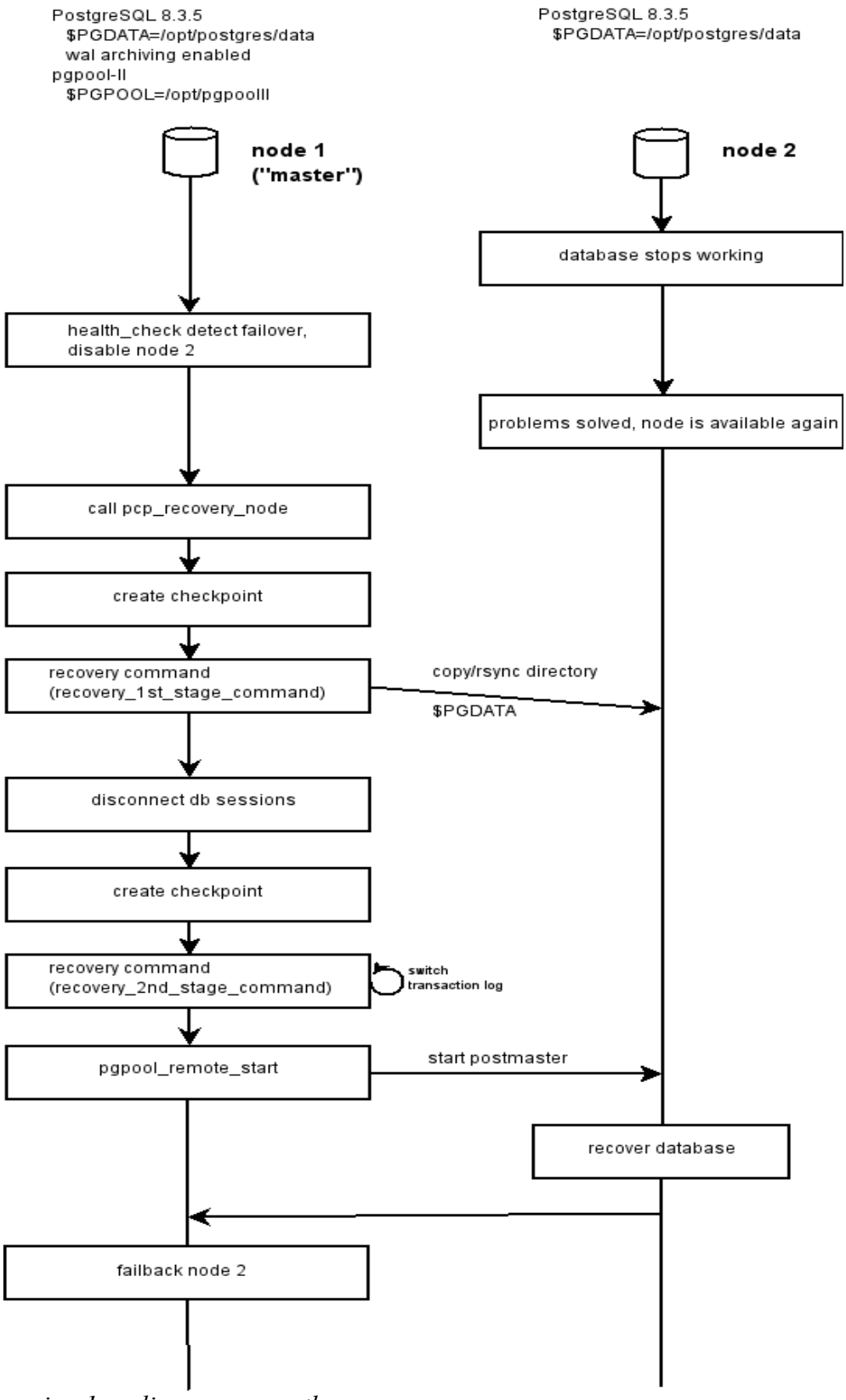
Parameters are:
„20“ → timeout in seconds
„node1“ → hostname where pgpool is running
„9898“ → port at which pgpool communication manager listens
„postgres“ „postgres“ → username password
„1“ → ID of the node we want to attach (refers to the backend number in pgpool.conf)

The statements produces a lot of log entries, like:

```
DEBUG: pid 4411: pcp_child: start online recovery
LOG:   pid 4411: starting recovering node 1
DEBUG: pid 4411: exec_checkpoint: start checkpoint
DEBUG: pid 4411: exec_checkpoint: finish checkpoint
LOG:   pid 4411: CHECKPOINT in the 1st stage done
LOG:   pid 4411: starting recovery command: "SELECT pgpool_recovery('copy_base_backup', 'infra02',
'/opt/postgres/data')"
DEBUG: pid 4411: exec_recovery: start recovery
DEBUG: pid 29658: starting health checking
DEBUG: pid 29658: health_check: 0 the DB node status: 2
DEBUG: pid 29658: health_check: 1 the DB node status: 3
DEBUG: pid 4411: exec_recovery: finish recovery
LOG:   pid 4411: 1st stage is done
LOG:   pid 4411: starting 2nd stage
LOG:   pid 4411: all connections from clients have been closed
DEBUG: pid 4411: exec_checkpoint: start checkpoint
DEBUG: pid 4411: exec_checkpoint: finish checkpoint
LOG:   pid 4411: CHECKPOINT in the 2nd stage done
LOG:   pid 4411: starting recovery command: "SELECT pgpool_recovery('pgpool_recovery_pitr', 'infra02',
'/opt/postgres/data')"
DEBUG: pid 4411: exec_recovery: start recovery
DEBUG: pid 4411: exec_recovery: finish recovery
DEBUG: pid 4411: exec_remote_start: start pgpool_remote_start
```

DEBUG: pid 29658: starting health checking
DEBUG: pid 4411: exec_remote_start: finish pgpool_remote_start
DEBUG: pid 29658: starting health checking
LOG: pid 4411: 1 node restarted
LOG: pid 4411: send_failback_request: fail back 1 th node request from pid 4411
LOG: pid 4411: recovery done
DEBUG: pid 29658: failover_handler called
DEBUG: pid 29658: failover_handler: starting to select new master node
LOG: pid 29658: starting fail back. reconnect host infra02(5432)
LOG: pid 29658: execute command: echo nodeid:1, host:infra02, port:5432, db-cluster path:/opt/postgres/data, new master id:0, old master id:0 > /opt/pgpoolII/log/failback1.log
DEBUG: pid 4411: pcp_child: received PCP packet type of service 'X'
DEBUG: pid 4411: pcp_child: client disconnecting. close connection
LOG: pid 29658: failover_handler: do not restart pgpool. same master node 0 was selected
LOG: pid 29658: failback done. reconnect host infra02(5432)

The drawing - online_recovery_theory- on the next page will show you the „workflow“ of online recovery mechanism:



Drawing 1: online_recovery_theory