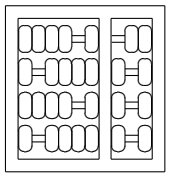

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Análise de um cluster de dados de alta disponibilidade usando Postgresql e pgpool2

Leonnardo Rabello *Luiz Eduardo Buzato*

Technical Report - IC-16-10 - Relatório Técnico

December - 2016 - Dezembro

The contents of this report are the sole responsibility of the authors.
O conteúdo do presente relatório é de única responsabilidade dos autores.

Análise de um cluster de dados de alta disponibilidade usando Postgresql e pgpool2

Leonnardo Rabello Luiz Eduardo Buzato

Resumo

O volume de dados usado na internet hoje é extremamente alto, e muitos desses dados são sensíveis. Para permitir a alta disponibilidade desses dados, soluções de load balancing e high avaliability estão disponíveis no mercado. Utilizando o PostgreSQL[1] com um método de replicação chamado de Streaming Replication[2], que cria uma arquitetura de master-slave, e a ferramenta pgpool[3], cujo um dos papéis é distribuir as cargas de operações de leitura entre os bancos replicados, foi possível medir o tempo de queda entre uma instância master e a recuperação da instância slave promovendo-se a master.

1 Introdução

1.1 Visão geral

Atualmente, a alta disponibilidade de dados é praticamente um pré-requisito de confiabilidade. Empresas como o Google, Amazon e Facebook possuem um volume de dados bastante elevado. E os usuários confiam na alta disponibilidade desses dados. Por exemplo, um usuário que se dispõe a salvar os seus documentos no Google Drive deseja que seja sempre possível acessar os documentos com a maior disponibilidade e a menor latência. Ou no caso da Amazon, um comprador deseja que sua compra seja efetivada sempre que for efetuada, sem a necessidade de refazer o processo caso haja algum dado seja perdido por condições adversas durante o processo.

1.2 Motivação

A proposta desse trabalho é analisar uma solução em particular para o caso de e-commerce. Em e-commerce, é necessário alta disponibilidade dos recursos e principalmente dos dados, já que produtos tem estoque e pode haver disputa de produtos entre usuários concorrentes. Também é necessário que em caso de falha de um banco de dados, seja possível haver um outro banco de backup para que o lojista perca o menor número possível de transações. Usando o TPC-W[4], um benchmark de e-commerce, e a arquitetura que será apresentada, foi possível avaliar a disponibilidade do serviço.

2 Arquitetura

Para montar uma arquitetura de alta disponibilidade, onde os dados possam ser redundantes e de alta disponibilidade, foi utilizada uma arquitetura que não possui único ponto de falha. Foi utilizado dois bancos de dados PostgreSQL configurados para trabalhar em Streaming Replication e duas instâncias da ferramenta pgpool2, que atua como load balancer[5] e failure detector[6]

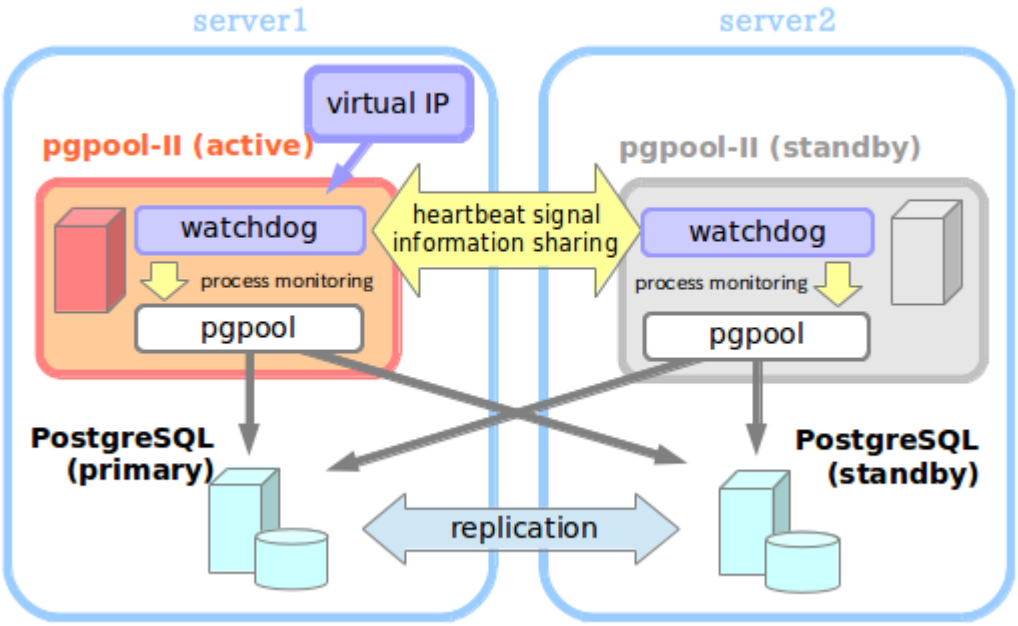

Figura 1: Arquitetura da solução com PostgreSQL e pgpool2

2.1 PostgreSQL

O PostgreSQL é um banco de dados objeto-relacional, open source, estando a mais de 15 anos sendo desenvolvido ativamente, além de uma arquitetura que possui uma forte reputação em confiabilidade, integridade e corretude dos dados. Possui também uma funcionalidade chamada de Streaming Replication, que consiste em aplicar registros online no banco de dados que está em standby. O banco em standby recebe do banco primário (via TCP) as informações de novos registros e aplica na sua estrutura, mantendo os dois bancos consistentes. Para que não haja perda de consistência, o banco em standby não aceita operações de escrita, apenas de leitura.

Foi utilizada a versão 9.5 do Postgres nesse trabalho.

2.2 pgpool2

O pgpool2 é um middleware[7] que atua entre o servidor que possui uma instância do PostgreSQL e o cliente que se conecta ao servidor. Das funcionalidades que esse possui, utilizamos as seguintes:

1. **Pool de conexões**: salsa as conexões com o servidor do PostgreSQL e reusa sempre que uma nova conexão com as mesmas propriedades é solicitada. Isso reduz o overhead de conexões e melhora em geral a taxa de transferência do sistema

2. **Load Balancing**: Como no nosso caso o banco de dados é replicado, executar uma query de leitura (SELECT, por exemplo) em qualquer um dos bancos, master ou standby, retorna o mesmo resultado. O pgpool tira vantagem da replicação para reduzir a carga em cada um dos bancos, distribuindo as queries de leitura entre os vários bancos configurados. Também melhora a taxa de transferência em geral.

3. **Failure detection**: Detecta se um dos servidores de banco falhou e permite criar configurações para tomar alguma decisão baseada na falha (failover).

O pgpool2 além de load balancing e failure detection, possui uma funcionalidade chamada de *watchdog*[8], que provê duas características:

1. **Virtual IP**: Transparência para a aplicação de qual instância do pgpool ela está se conectando.

2. **Heartbeat**: Detecção da presença de outras instâncias do pgpool, sendo possível uma instância se eleger como líder se o líder anterior estiver desconectado.

Neste trabalho foi usada a versão 3.5.4 do pgpool.

3 Preparo do ambiente

Para este trabalho, foi utilizado um ambiente no cluster do Laboratório de Sistemas Distribuidos. Duas máquinas, lsd07 e lsd08, com Debian GNU/Linux 8.6 jessie foram utilizadas e configuradas.

Em uma dessas máquinas se fez necessária uma instância do Apache Tomcat versão 8, que é um Java Servlet Container que implementa as especificações do Java EE. Essa instância já estava disponível na máquina lsd07.

3.1 Instalação e configuração do PostgreSQL nas duas máquinas

Antes de tudo, foi necessário adicionar o repositório do PostgresSQL na lista de fontes do Debian:

```
deb http://apt.postgresql.org/pub/repos/apt/ jessie-pgdg main
wget --quiet -O - http://apt.postgresql.org/pub/repos/apt/ACCC4CF8.asc |
sudo apt-get update
```

Após isso, podemos instalar o PostgresSQL junto com a biblioteca que faz a comunicação com o pgpool2

```
$ sudo apt-get install postgresql postgresql-9.5-pgpool2
```

O pacote instalado irá fornecer bibliotecas compartilhadas com funções do Postgres que serão acessadas pelo pgpool2:

1. /usr/lib/postgresql/9.5/lib/pgpool_adm.so
2. /usr/lib/postgresql/9.5/lib/pgpool_recovery.so
3. /usr/lib/postgresql/9.5/lib/pgpool_regclass.so

Os arquivos de configuração das duas máquinas (pg_hba.conf e postgresql.conf) deverão ser idênticos e conter as seguintes configs:

Listing 1: pg_hba.conf

```
host all         postgres   127.0.0.1/32    trust
host all         postgres   10.1.0.0/24     trust
host all         postgres   143.106.24.0/24 trust
host tpcw        tpcw_user  127.0.0.1/32    trust
host tpcw        tpcw_user  10.1.0.0/24     trust
host tpcw        tpcw_user  143.106.24.0/24 trust
host replication postgres   10.1.0.0/24     trust
host replication postgres   143.106.24.0/24 trust
```

Essa configuração permite que o postgres aceite conexões de confiança (sem senha) dos ips acima, para os usuários acima. No caso do usuário postgres, como ele é dono do banco, ele pode acessar todos os bancos do cluster sem senha. No caso do usuário tpcw, apenas o usuário tpcw_user. Também se faz necessário uma configuração a parte da replicação para o usuário postgres. O não uso de senha é uma falha de segurança grave em ambientes de produção, mas não foi usado aqui apenas por questão de facilidade.

Listing 2: postgresql.conf

```
wal_level = hot_standby
max_wal_senders = 3
max_replication_slots = 3
hot_standby = on
hot_standby_feedback = on
pgpool.pg_ctl = '/usr/lib/postgresql/9.5/bin/pg_ctl'
```

Com essas configurações, é possível ligar o modo de streaming replication para os dois bancos. A flag *hot_standby* só é lida no caso de o banco estar como standby, e isso é feito

com a presença de um arquivo chamado recovery.conf que deverá estar presente dentro do diretório de dados (citado mais a frente).

Após essas configurações, podemos criar um novo cluster, onde o dono é o usuário postgres, porta 5433 e tem o nome de tpcw:

```
sudo pg_createcluster -u postgres -p 5433 9.5 tpcw
```

Com esse comando, será criado o diretório /etc/postgresql/9.5/tpcw com os arquivos de configuração e o diretório /var/lib/postgresql/9.5/tpcw como diretório de dados. Essas configurações podem ser modificadas ao criar um novo cluster. Foi usada a configuração padrão apenas por conveniência. Para mais informações de como modificar esses diretórios, consultar o manual:

```
man pg_createcluster
```

3.2 Instalação e configuração do pgpool2 nas duas máquinas

Podemos instalar o pgpool pelo apt:

```
$ sudo apt-get install pgpool2
```

Também se faz necessário o download do source, pois o mesmo contém scripts sql que criam funções no banco. Esses scripts devem ser executados com os 2 bancos ativos e com a replicação configurada. Também deve-se criar uma pasta dos scripts dentro do diretório de configuração do Postgres para melhor manutenção (/etc/postgresql/9.5/tpcw/sql)

```
mkdir /etc/postgresql/9.5/tpcw/sql
wget http://www.pgpool.net/download.php?f=pgpool-II-3.5.4.tar.gz
tar xf pgpool-II-3.5.4.tar.gz
cd pgpool-II-3.5.4/src/sql
cp pgpool_adm/pgpool_adm.sql.in /etc/postgresql/9.5/tpcw/sql/pgpool_adm.
cp pgpool-recovery/pgpool-recovery.sql.in /etc/postgresql/9.5/tpcw/sql/p
cp pgpool-regclass/pgpool-regclass.sql.in /etc/postgresql/9.5/tpcw/sql/p
```

Depois disso, em todos os scripts existe uma string 'MODULE_PATHNAME' que deverá ser substituida por '/usr/lib/postgresql/9.5/lib/{pgpool_adm, pgpool-recovery, pgpool-regclass}', dependendo do arquivo editado.

Agora, podemos executar os scripts (com o banco master e standby online):

```
cd /etc/postgresql/9.5/tpcw/sql
psql -h lsd07 -U postgres -f pgpool_adm.sql
psql -h lsd07 -U postgres -f pgpool-recovery.sql
psql -h lsd07 -U postgres -f pgpool-regclass.sql
```

Para que seja possível levantar a interface de Virtual IP, é necessário um pacote extra, chamado *iputils-arping*:

```
sudo apt-get install iputils-arping
```

3.2.1 pgpool.conf

A configuração do pgpool fica no diretório */etc/pgpool2*. Existem 3 principais arquivos: *pgpool.conf*, *pcp.conf* e *pool_hba.conf*. O pool_hba.conf tem um papel semelhante ao pg_hba.conf, mas pode (e foi) ser desativado no pgpool.conf. Já o pcp.conf deve ser usado em conjunto com o pool_hba.conf, pois nele deve ser salvo a informação de nome de usuário e senha hasheada do banco de dados, para que não seja necessário digitar a senha durante os processos automáticos que o pgpool dispara.

O pgpool.conf vem com uma configuração padrão, mas para trabalhar com a arquitetura descrita, se faz necessário fazer algumas alterações. Além disso, como vamos rodar duas instâncias do pgpool em duas máquinas distintas, existem configurações específicas para cada máquina. Primeiramente, será mostrado a configuração que será comum as duas máquinas e, em seguida, a configuração particular de cada instância.

```
listen_address = '*'
port = 9999
socket_dir = '/tmp'
pcp_listen_address = '*'
pcp_port = 9898
pcp_socket_dir = '/tmp'

backend_hostname0 = 'lsd07'
backend_port0 = 5433
backend_weight0 = 1
backend_data_directory0 = '/var/lib/postgresql/9.5/tpcw/'
backend_flag0 = 'ALLOW_TO_FAILOVER'
backend_hostname1 = 'lsd08'
backend_port1 = 5433
backend_weight1 = 1
backend_data_directory1 = '/var/lib/postgresql/9.5/tpcw/'
backend_flag1 = 'ALLOW_TO_FAILOVER'

enable_pool_hba = off

# release client and server after 2 seconds of inactivity
connection_life_time = 2
client_idle_limit = 2

# activate load balancing mode
load_balance_mode = on

# activate streaming replication
master_slave_mode = on
```

```
master_slave_sub_mode = 'stream'

# health check
health_check_period = 5
health_check_timeout = 20
health_check_user = 'postgres'

# failover command
failover_command = '/home/postgres/failover.sh %d %h %p %m "%H" %r %M %P'

# online recovery
recovery_1st_stage_command = 'recovery_1st_stage.sh'

# watchdog config
use_watchdog = on
delegate_IP = '10.1.0.99'
wd_port = 9000
wd_lifecheck_method = 'heartbeat'
wd_interval = 3
if_cmd_path = '/bin'
if_up_cmd = 'ip addr add $_IP_$/24 dev eth0 label eth0:0'
if_down_cmd = 'ip addr del $_IP_$/24 dev eth0'
arping_path = '/usr/bin'
arping_cmd = 'arping -U $_IP_$ -w 1 -I eth0'
```

E as configurações específicas de cada máquina:

Listing 3: lsd07

```
wd_hostname = 'lsd07'
wd_heartbeat_port = 9694
heartbeat_destination0 = 'lsd08'
heartbeat_destination_port0 = 9694
other_pgpool_hostname0 = 'lsd08'
other_pgpool_port0 = 9999
other_wd_port0 = 9000
```

Listing 4: lsd08

```
wd_hostname = 'lsd08'
wd_heartbeat_port = 9694
heartbeat_destination0 = 'lsd07'
heartbeat_destination_port0 = 9694
other_pgpool_hostname0 = 'lsd07'
other_pgpool_port0 = 9999
other_wd_port0 = 9000
```

No arquivo de configuração, existem dois itens especiais: *failover_command*[9] e *recovery_1st_stage_command*[10]. Estes parâmetros apontam para scripts que ocorreram durante o failover e o online recovery, respectivamente.

O script *failover.sh* basicamente recebe vários dados como parâmetros, entre eles o endereço do host cujo banco deve ser promovido a master. Com esses dados é trivial executar um comando para promover o cluster:

```
ssh -T postgres@$new_master_host_name pg_ctlcluster $cluster promote
```

Já o *recovery_1st_stage_command.sh* é um script mais complexo, que executa ao usarmos a ferramenta pcp do pgpool, com o comando *pcp_recovery_node*[11]. Na verdade, como a quantidade de parâmetros é limitado (ele recebe apenas o hostname do nó que necessita ser recuperado, e é executado no nó master), então existe um caminho mais longo a ser executado. Existe um script chamado *replication.sh* contido no home de cada uma das máquinas, que estes são executados no servidor que deverá ser recuperado e se comunica com o servidor que está atuando como master no momento da recuperação.

O que esse script faz é: um *pg_basebackup* do banco master, cria um arquivo recovery.conf no servidor standby com as configurações apontando para o master atual, e cria o script *pgpool_remote_start*, que serve como comando pra iniciar o servidor postgresql na máquina recuperada.

Vale ressaltar que o basebackup não é a melhor maneira de se alinhar um servidor após uma falha, pois a partir do postgresql 9.5 existe uma funcionalidade chamada *pg_rewind* [12], que, como o nome diz, rebobina o servidor master e compara com o estado do servidor que falhou, até encontrar um ponto em comum. Achado esse ponto em comum, tudo aquilo que é novo no servidor master é aplicado no standby. Esse processo é mais eficiente do que copiar todos os dados de um servidor para outro. Porém, o pgpool 3.5.4 não suporta ainda o pg_rewind[13], por isso foi usado o basebackup.

Por questões de segurança, o pgpool exige que tanto o *recovery_1st_stage_command.sh* quanto o *pgpool_remote_start* estejam dentro do diretório de dados (no nosso caso /var/lib/-postgresql/9.5/tpcw em ambas as máquinas).

Após as configurações, podemos iniciar as instâncias do pgpool com o comando:

```
sudo pgpool -n -D > /var/log/postgresql/pgpool.log 2>&1 &
```

Onde:

- **-n**: faz com o que o pgpool não rode como daemon
- **-D**: descarta o arquivo de status /tmp/pgpool_status, ou seja, ignora os estados anteriores salvos

4 Experimento

O experimento consiste em duas etapas:

1. Simular uma falha no banco master e verificar o tempo de recuperação do banco standby

2. Simular uma falha na instância master do pgpool e verificar o tempo de recuperação do nó secundário

Para simular um e-commerce real, o TPC-W possui uma ferramenta chamada *Remote Browser Emulator* (RBE). O RBE possui 3 modos de operação:

1. **Browsing**: simula apenas clientes olhando os produtos. Operações majoritariamente de leitura.

2. **Shopping**: simula clientes comprando produtos. Operações majoritariamente de escrita.

3. **Ordering**: simula clientes mistos. Operações de leitura e escrita.

Ele possui diversos parâmetros, além do modo de operação, que utilizamos no nosso experimento:

- **Ramp-up time**: Tempo usado para aquecer o simulador. No nosso caso, 5 segundos.
- **Ramp-down time**: Tempo de espera para o fim do experimento após o intervalo de medida. Também de 5 segundos.
- **Measurement interval**: Intervalo de medida do experimento. Nesse tempo é que serão medidos os WIPS (web interactions per sec). 70 segundos no nosso caso.
- **EB number**: O número de browsers emulados. No nosso caso, 70 segundos.
- **Think Time**: O tempo de emulação de um humano navegando de uma página para outra. 0.5 segundos.

Todos os outros parâmetros existentes foram deixados com valores padrão.

O experimento constituiu em um script que executa os seguintes passos:

1. Reseta o banco primário, levantando um dump do banco de dados do TPC-W.

2. Reseta o banco secundário, clonando o banco primário.

3. Reseta o servidor Tomcat.

4. Inicia as instâncias do pgpool.

5. Inicia uma execução do RBE.

6. Após metade do tempo total do experimento (Ramp-up+Measurement interval+Ramp-down) executa:

    - Mata o banco primário ou
    - Mata instância primária do pgpool

Após isso, o experimento gera um arquivo .m de MatLab que, usando uma ferramenta fornecida pelo professor, é possível gerar um gráfico de WIPS $\times$ T.

5 Resultados

Temos os seguintes gráficos de resultado após os experimentos:

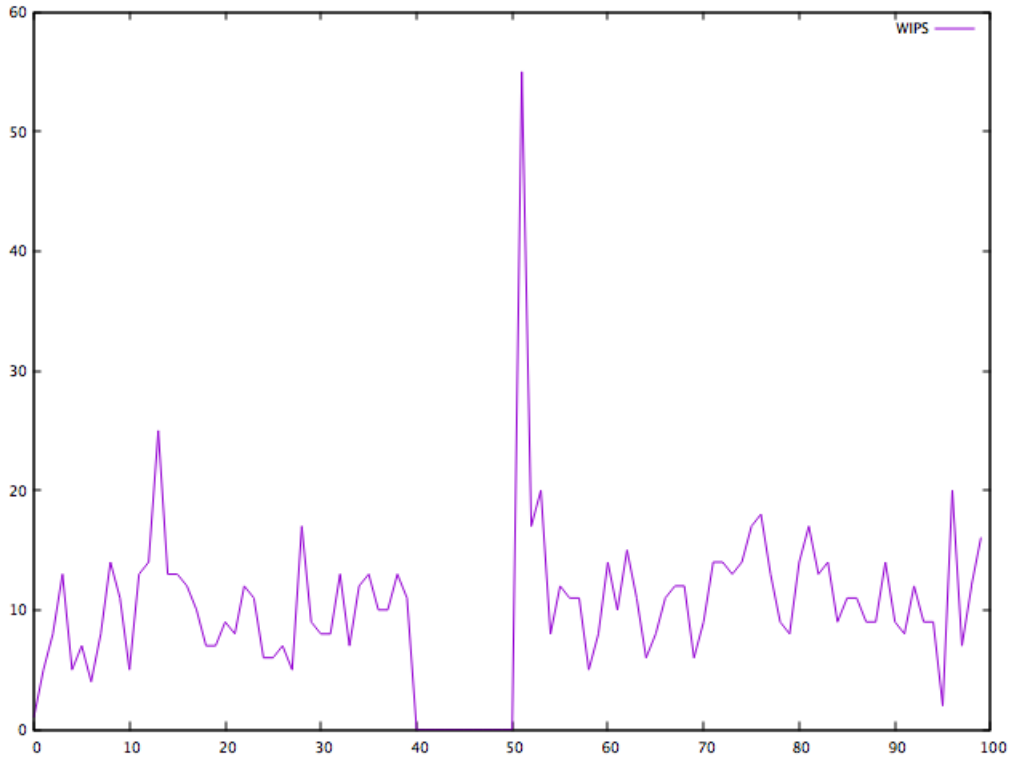

Figura 2: Gráfico do experimento que mata o servidor postgresql

Podemos concluir que a recuperação de um banco falho é mais rápida do que a do servidor pgpool falho. Pode ser devido ao heartbeat do processo watchdog ser de 3 segundos. Talvez se for diminuido, o tempo de recuperação seja diminuido também, mas isso aumenta o overhead da rede, pois a cada intervalo de tempo é enviado um pacote para checar se o servidor está vivo.

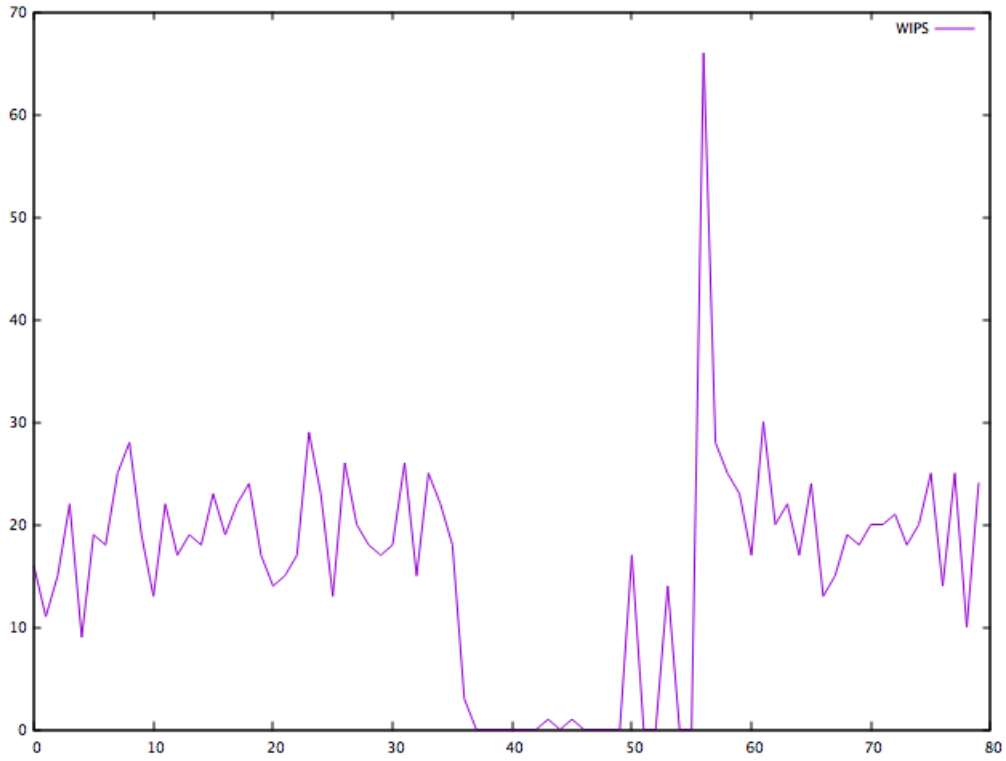

Figura 3: Gráfico do experimento que mata o servidor pgpool

O tempo de recuperação do experimento que mata o servidor postgres se mostrou constante dentro de inúmeros experimentos feitos com tempos totais de experimentos variados: em torno de 10 segundos. Apesar de também ser um tempo alto, os experimentos mostram que o pgpool é consistente e sempre consegue subir o banco standby para master em um intervalo de tempo constante. Diminuir o tempo do healthcheck de 5 para 2 segundos não se mostrou eficiente em diminuir o tempo de indisponibilidade.

6 Conclusão e trabalhos futuros

O pgpool2 é uma ferramenta que é interessante de ser usada para clusters de dados de alta disponibilidade. É madura o suficiente para usar em sistemas com mais de 2 nós sevidores de banco de dados, mas com ressalvas. Sua documentação ainda é muito precária, mas os desenvolvedores se mostram rápidos em responder dúvidas nas listas de e-mail. Ainda falta amadurecer algumas funcionalidades, como por exemplo o fato de o pgpool não dar suporte ao *pg_rewind*.

Durante a execução deste trabalho, foi lançada a versão 3.6[14] do pgpool, com uma docu-

mentação levemente melhor, com um formato semelhante à documentação do PostgreSQL, usando SGML[15]. Uma sugestão para um trabalho futuro é usar essa versão e modificar algumas configurações, como por exemplo, colocar mais nós como servidores de banco de dados.

Referências

[1] Postgresql. http://www.postgresql.org.

[2] Streaming replication. http://wiki.postgresql.org/wiki/Streaming_Replication.

[3] pgpool. http://www.pgpool.net.

[4] Tpc-w. http://www.tpc.org/tpcw.

[5] Load balancing. https://en.wikipedia.org/wiki/Load_balancing_(computing).

[6] Failure detection. https://en.wikipedia.org/wiki/Failure_detector.

[7] Middleware. https://en.wikipedia.org/wiki/Middleware.

[8] Watchdog. http://www.pgpool.net/docs/latest/en/html/tutorial-watchdog-intro.html.

[9] Failover command. http://www.pgpool.net/docs/latest/en/html/runtime-config-failover.html.

[10] Online recovery. http://www.pgpool.net/docs/latest/en/html/runtime-online-recovery.html.

[11] Pcp recovery node. http://www.pgpool.net/docs/latest/en/html/pcp-recovery-node.html.

[12] pg_rewind. https://www.postgresql.org/docs/9.5/static/app-pgrewind.html.

[13] pgpool todo. http://pgpool.net/mediawiki/index.php/TODO#Allow_to_use_pg_rewind_in_online_recovery.

[14] pgpool 3.6 release notes. http://www.pgpool.net/docs/latest/en/html/release-3-6.html.

[15] Standard generalized markup language. https://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language.