Annexe C – Mise en œuvre Haproxy et pgbouncer.

le système de pooling sera mise en place sur la machine debian
nagios.exponentielhippy.fr à l'adresse 10.11.12.249

Mise en place de haproxy :
root@nagios:~#apt-get install haproxy

puis se diriger vers le répertoire /etc/haproxy/haproxy.cfg que l'on peut copier avant de le modifier

Voici ce que donne mon fichier de configuration compte tenu de mes configs.

```
global
        #log 127.0.0.1   local0
    #log 127.0.0.1   local1 notice
        maxconn 400
    user haproxy
    group haproxy
    daemon

listen DB_frontend
    bind              10.11.12.249:5432
    mode              tcp
    #log              global
    #option                    tcplog
        #option            dontlognull
    #maxconn            200
    clitimeout          9000 # 9000 sur le port 5432
    balance                    roundrobin # le poids varie automatiquement en fonction de la charge
        #balance               static-rr # le poids ne varie plus, seul le rapport compte
        #balance                        leastconn          # celui des serveurs qui a le moins de connexions en cours
    contimeout          9000 # 9000 sur le port 5432
    srvtimeout          9000 # 9000 sur le port 5432

        #retries               2 # default value => 3
        #option                redispatch # autres tentatives sur un autre serveuro

        # on passe par posgresql directement sur le port 5432
        server            DB_master 10.11.12.241:5432 check inter 1000 weight 10
        server            DB_master-2 10.11.12.243:5432 check inter 1000 weight 10
        server             DB_Slave 10.11.12.242:5432 check inter 1000 weight 10 backup

    # on passe par pgbouncer installé sur  chaque host sur le port 6432
        #server            DB_master 10.11.12.241:6432 check inter 1000 weight 1
        #server            DB_master-2 10.11.12.243:6432 check inter 1000 weight 1
        #server            DB_Slave 10.11.12.242:6432 check inter 1000 weight 1 backup
```

De façon simple, on peut tester l'équilibrage de charge avec pgbench de la façon suivante :

```
root@debian:~# ./pgbench -h 10.11.12.249 -p 5432 -U postgres -c 160 -t 1 test1
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 1
query mode: simple
number of clients: 160
number of threads: 1
number of transactions per client: 1
number of transactions actually processed: 160/160
tps = 95.444893 (including connections establishing)
tps = 185.323320 (excluding connections establishing)
```

en considérant 80 connexions sur chaque serveur, à raison d'une transaction
pour 160 clients, on constate la pgbench_history de chacune des tables des 3 bases
contient un total de 79 à 82 lignes, ce qui signifie que les transactions émises par
pgbench ont été correctement réparties dans un rapport 50/50.
modifions le paramètre de poids et réalisons les mêmes séries.

En passant le poids de master-2 à 256, ce qui constitue le poids maximal, on indique
donc que le serveur est surchargé, on ne peut donc plus le servir et ainsi, nous manquons de
connexions vu que :

root@debian:~# ./pgbench -h 10.11.12.249 -p 5432 -U postgres -c 160 -t 1 test1
starting vacuum...end.
Connection to database "test1" failed:
FATAL: d?sol?, trop de clients sont d?j? connect?s
transaction type: TPC-B (sort of)
scaling factor: 1
query mode: simple
number of clients: 160
number of threads: 1
number of transactions per client: 1
number of transactions actually processed: 0/160
tps = 0.000000 (including connections establishing)
tps = 0.000000 (excluding connections establishing)

Voici le mieux que l'on puisse espérer avec cette configuration

root@debian:~# ./pgbench -h 10.11.12.249 -p 5432 -U postgres -c 80 -t 1 test1
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 1
query mode: simple
number of clients: 80
number of threads: 1
number of transactions per client: 1
number of transactions actually processed: 80/80
tps = 50.051021 (including connections establishing)
tps = 113.174662 (excluding connections establishing)

---

le nombre de lignes de pgbench_history varie donc de 58 à 80 lignes, dans ce cas,
ce qui signifie que la table master est bien plus sollicitée, voire entièrement sollicitée.
reprenons un poids d'équilibrage égal pour chaque serveur mais faisons le varier de façon
identique.
Ce qui importe, c'est le ratio des poids plutôt que le poids lui-même, à ceci près qu'en mode
roundrobin, ce poids peut ajusté automatiquement par haproxy. Ce qui explique sans doute
les variations du nombre de lignes dans la table pgbench_history.

essayons à présent le mécanisme static-rr pour static roundrobin, qui lui, ne modifie pas
le poids au cours de l'utilisation de la base, et maintenons un rapport de 1.

Quelques mises au point sur la confi haproxy.cfg :

```
global
        maxconn 200
        user haproxy
        group haproxy
        daemon


listen DB_frontend
```

```
bind            10.11.12.249:5432
mode             tcp
log             global
option           dontlognull
#maxconn            200
clitimeout        5000
balance          roundrobin
#balance           static-rr # le poids ne varie plus, seul le rapport compte
#balance           leastconn
contimeout         5000
srvtimeout         5000
server          DB_master 10.11.12.241:5432 check inter 1000 weight 10
server          DB_master-2 10.11.12.243:5432 check inter 1000 weight 10
server          DB_Slave 10.11.12.242:5432 check inter 1000 weight 10 backup
```

Notons que la ligne suffixé par backup permet au loadbalancing de mettre en oeuvre la bascule sur le DB_Slave en cas d'interruption des deux master et master-2, et qu'en cas de panne d'un des master, le survivant prend toute la charge. Evidemment, si plus de 100 clients sollicitent ce cluster, certaines requêtes seront rejetées.

Retour sur haproxy et le spof (Simple Point Of Failure)

Une façon simple d'opérer consiste à s'appuyer sur le dns interne
et d'un deuxième serveur haproxy, sinon, il est aussi possible
de déployer la solution heartbeat sur une configuration redondante
de machine haproxy, rien d'exceptionnel.

Le pooling de connexion selon Pgbouncer.

Mise en place de PGBouncer pour la mise en place d'un pool de connexions,
de la même façon que le fait pgpool II.

```
[root@postgresql-centos-master ~]# yum install libevent
[root@postgresql-centos-master ~]# yum install libevent-devel
```

bibliothèque de notifications d' évènements nécessaire pour pouvoir
compiler pgbouncer.

```
[root@postgresql-centos-master ~]# wget http://pgfoundry.org/frs/download.php/3393/pgbouncer-1.5.4.tar.gz
--2013-03-15 13:52:06--  http://pgfoundry.org/frs/download.php/3393/pgbouncer-1.5.4.tar.gz
Résolution de pgfoundry.org... 200.46.204.130
Connexion vers pgfoundry.org|200.46.204.130|:80...connecté.
requête HTTP transmise, en attente de la réponse...200 OK
Longueur: 339610 (332K) [application/binary]
Sauvegarde en : «pgbouncer-1.5.4.tar.gz»

100%[==========================================================================>] 339 610
341K/s   ds 1,0s

2013-03-15 13:52:07 (341 KB/s) - «pgbouncer-1.5.4.tar.gz» sauvegardé [339610/339610]

[root@postgresql-centos-master ~]# tar xzf pgbouncer-1.5.4.tar.gz
[root@postgresql-centos-master ~]# cd pgbouncer-1.5.4


[root@postgresql-centos-master pgbouncer-1.5.4]# yum install gcc
au besoin
[root@postgresql-centos-master pgbouncer-1.5.4]# ./configure --prefix=/usr/local
```

```
[root@postgresql-centos-master pgbouncer-1.5.4]# make

[root@postgresql-centos-master pgbouncer-1.5.4]# make install
    -->     doc
    INSTALL  pgbouncer.1 /usr/local/share/man/man1
    INSTALL  pgbouncer.5 /usr/local/share/man/man5
    <--     doc
    INSTALL  pgbouncer /usr/local/bin
    INSTALL  README /usr/local/share/doc/pgbouncer
    INSTALL  NEWS /usr/local/share/doc/pgbouncer
    INSTALL  etc/pgbouncer.ini /usr/local/share/doc/pgbouncer
    INSTALL  etc/userlist.txt /usr/local/share/doc/pgbouncer


[root@postgresql-centos-master pgbouncer-1.5.4]# cp /usr/local/share/doc/pgbouncer/pgbouncer.ini /etc/pgbouncer.ini

###############  paramètres pgbouncer.ini

################################################

[root@postgresql-centos-master pgbouncer-1.5.4]# touch /var/log/pgbouncer.log
[root@postgresql-centos-master pgbouncer-1.5.4]# chmod a+w /var/log/pgbouncer.log
[root@postgresql-centos-master pgbouncer-1.5.4]# su postgres

bash-4.1$ cd
bash-4.1$ pgbouncer -d /etc/pgbouncer.ini
2013-03-15 14:01:25.223 10846 LOG File descriptor limit: 1024 (H:4096), max_client_conn: 1000, max fds possible:
1010
```

Notre configuration pgbouncer est maintenant opérationnel sur notre master, et l'on procède à l'identique sur les autres hosts, à savoir master-2 et slave, et d'ailleurs, vérifions le :

```
[root@postgresql-centos-master pgbouncer-1.5.4]# netstat -taupen | grep -i 6432
tcp        0      0 0.0.0.0:6432                0.0.0.0:*                   LISTEN      26         363288     13753/pgbouncer

[root@postgresql-centos-master pgbouncer-1.5.4]# psql -U postgres -p 6432 pgbouncer
psql (8.4.13, serveur 1.5.4/bouncer)
Saisissez « help » pour l'aide.

pgbouncer=# \q
```

Pour relancer pgbouncer sans avoir à redémarrer le processus,

```
[root@postgresql-centos-slave pg_log]# psql -p 6432 -U postgres pgbouncer
psql (8.4.13, serveur 1.5.4/bouncer)
Saisissez « help » pour l'aide.

pgbouncer=# RELOAD;
RELOAD
pgbouncer=# \q
```

Les premiers tests avec et sans pgbouncer, nous avons 10 clients (-c 10) qui pour chaque transaction, ouvre une nouvelle connexion (-C), et cela pendant 60 secondes (-T 60).

d'abord en direct sans pgbouncer :

```
root@debian:~# ./pgbench -h 10.11.12.241 -p 5432 -U postgres -c 10 -C -T 60 test1
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 1
query mode: simple
number of clients: 10
```

number of threads: 1
duration: 60 s
number of transactions actually processed: 4550
tps = 75.746152 (including connections establishing)
tps = 222.125474 (excluding connections establishing)

puis en passant par pgbouncer :

root@debian:~# ./pgbench -h 10.11.12.241 -p 6432 -U postgres -c 10 -C -T 60 test1
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
number of transactions actually processed: 5634
tps = 93.776053 (including connections establishing)
tps = 130.555312 (excluding connections establishing)

On a effectivement une amélioration sensible du nombre de
transactions par secondes d'un peu plus de 15% lorsqu'il s'agit
de comptabiliser les temps de connexions.

tests sur le deuxième master-2 :

root@debian:~# ./pgbench -h 10.11.12.243 -p 5432 -U postgres -c 10 -C -T 60 test1
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
number of transactions actually processed: 4259
tps = 70.894078 (including connections establishing)
tps = 199.361062 (excluding connections establishing)

root@debian:~# ./pgbench -h 10.11.12.243 -p 6432 -U postgres -c 10 -C -T 60 test1
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
number of transactions actually processed: 5814
tps = 96.797506 (including connections establishing)
tps = 134.158780 (excluding connections establishing)

Et enfin sur le poste slave :

root@debian:~# ./pgbench -h 10.11.12.242 -p 5432 -U postgres -c 10 -C -T 60 test1
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
number of transactions actually processed: 11585
tps = 193.010514 (including connections establishing)
tps = 517.990663 (excluding connections establishing)

```
root@debian:~# ./pgbench -h 10.11.12.242 -p 6432 -U postgres -c 10 -C -T 60 test1
starting vacuum...end.
transaction type: TPC-B (sort of)
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
number of transactions actually processed: 16622
tps = 276.949505 (including connections establishing)
tps = 456.659066 (excluding connections establishing)
```

Remarquons que sur le poste slave, nous avons aussi une amélioration des performances en terme de transaction mais aussi et surtout, ce qui est remarquable, c'est le nombre de transactions qui est bien plus important que sur les deux autres configurations, et cela provient du fait que Bucardo monopolise également son hôte en terme de ressources machines, et de connexions au serveur posgresql source ou cible. C'est peut être d'ailleurs un de ces points faibles.