Superviser PostgreSQL: travaux pratiques

Auteurs : Julien Rouhaud & Thomas Reiss

Énoncés

Initialisation de base

- Configurez les traces de PostgreSQL pour avoir des traces intéressantes
  - connexions et déconnexions
  - autovacuum
  - en-tête de ligne
  - checkpoints
  - fichiers temporaires
  - attente de verrou
  - requêtes de plus d'1 seconde
- Créez une base de données pgbench

Premiers diagnostics

- État des différents processus PostgreSQL
- Lecture rapide des logs
- Utilisation de pgBadger
  - génération d'un rapport pgBadger
  - analyse des informations
  - correction des problèmes de configuration
- Désactivez la synchronisation des écritures de VirtualBox, en exécutant la commande suivante depuis la machine hôte :
  - `vboxmanage setextradata “atelier_pgday” “VBoxInternal/Devices/ahci/0/LUN#0/Config/IgnoreFlush” 1`
- Initialisez la structure spécifique à pgbench (scale-factor de 100)
  - `pgbench -i -s 100 pgbench`
- Ré-activez la synchronisation des écritures de VirtualBox, en exécutant la commande suivante depuis la machine hôte :
  - `vboxmanage setextradata “atelier_pgday” “VBoxInternal/Devices/ahci/0/LUN#0/Config/IgnoreFlush” 0`

Ne pas hésiter à supprimer le fichier de trace (voir dans le répertoire `/var/lib/pgsql/9.3/data/pg_log`), supprimer la base, la recréer et réinitialiser pgBench afin de faciliter la lecture du rapport.

Test de performance

- Lancez le collecteur pgCluu
- Déclenchez un test de performance avec pgbench
  - 30 clients simultanés, 8 threads
  - pendant 2 minutes
  - en lecture seule (uniquement des SELECT)
- Arrêtez le collect pgCluu
- Créez un rapport pgCluu
- Examinez le rapport
- Améliorez la configuration et redémarrez PostgreSQL
- Lancez le collecteur pgCluu
- Déclenchez un test de performance avec pgbench
  - 30 clients simultanés
- Arrêtez le collect pgCluu
- Créez un rapport pgCluu
- analyse du rapport
  - correction de la configuration si d'autres éléments sont mis en lumière

Supervision Nagios

- Étude de la pré-configuration disponible
- mise en place des sondes spécifiques PostgreSQL pour l'instance locale
- lancer différents tests de performances avec pgBench
- vérifier les différentes alertes pouvant se déclencher
- affiner la configuration des différentes sondes en fonction du dimensionnement de la machine virtuelle
- valider cette configuration à l'aide de nouveaux tests de performance pgBench
- reconfigurer l'instance PostgreSQL en cas de besoin

Corrections

Initialisation de base

- Configurez les traces de PostgreSQL pour avoir des traces intéressantes
  - connexions et déconnexions
  - autovacuum
  - en-tête de ligne
  - checkpoints
  - fichiers temporaires
  - attente de verrou
  - requêtes de plus d'1 seconde

Modifier le fichier `/var/lib/pgsql/9.3/data/postgresql.conf` avec les paramètres suivants, dans leur ordre d'apparition :

```
log_min_duration_statement = 1s
log_connections = on
log_disconnections = on
log_checkpoints = on
log_line_prefix = '%t [%p]: [%l-1] '
log_lock_waits = on
log_temp_files = 0
log_autovacuum_min_duration = 0
```

Les modifications peuvent être prises en compte par rechargement de la configuration :

```
/etc/init.d/postgresql-9.3 reload
```

(en tant que root)

- Créez une base de données pgbench

Tout d'abord, il faut se connecter en tant qu'utilisateur `postgres` :

```
# su - postgres
```

Puis exécuter la commande suivante :

```
createdb pgbench
```

- Désactivez la synchronisation des écritures de VirtualBox, en exécutant la commande suivante depuis la machine hôte :
  - `vboxmanage setextradata “atelier_pgday” “VBoxInternal/Devices/ahci/0/LUN#0/Config/IgnoreFlush” 1`

Cette manipulation peut se faire à chaud.

VirtualBox pose un très gros problème lorsqu'il est utilisé avec des bases de données : il ment sur les écritures avec fsync et ne permet pas de créer réellement de contention sur le disque comme ce serait le cas avec une machine physique. Se référer aux tests SQLite et FS-Mark de Phoronix pour s'en rendre compte : http://www.phoronix.com/scan.php?page=article&item=linux_kvm_virtualbox4&num=2.

VirtualBox possède une clé de configuration cachée qui permet de désactiver ce comportement, mais au prix de performances réellement dégradées, à savoir des débits en écritures synchrones proches d'un lecteur de disquettes en cas d'utilisation de disque mécanique standard. Voir : http://www.virtualbox.org/manual/ch12.html#idp59653904.

Pour tester les performances dans les deux configurations, utilisez la commande suivante: `dd if=/dev/zero of=/tmp/testfile count=1000 bs=4096 oflag=sync`. Les résultats parleront d'eux-mêmes.

- Initialisez la structure spécifique à pgbench (scale-factor de 100)

Toujours en tant qu'utilisateur `postgres` :

```
pgbench -i -s 100 pgbench
```

- Ré-activez la synchronisation des écritures de VirtualBox, en exécutant la commande suivante depuis la machine hôte :
  - `vboxmanage setextradata "atelier_pgday" "VBoxInternal/Devices/ahci/0/LUN#0/Config/IgnoreFlush" 0`

N'oubliez pas d'exécuter cette commande sinon il ne sera pas possible d'obtenir des résultats exploitables.

Premiers diagnostics

- État des différents processus PostgreSQL

```
# ps -ef|grep postgres
```

Plusieurs processus sont présents :

- `postmaster`
- `logger`
- `checkpointer`
- `writer`
- `wal writer`
- `archiver`
- `stats collector`

Autovacuum est absent. Le processus `archiver` indique qu'il n'a pas réussi à réaliser l'archivage d'un WAL :

```
postgres   981   974  0 10:39 ?        00:00:00 postgres: archiver process
failed on 000000010000000000000001
```

- Lecture rapide des logs

La lecture des logs montre surtout les problèmes du processus `archiver` :

```
(...)
2014-06-02 10:41:19 CEST [981]: [19-1] LOG:  archive command failed with exit
code 1
2014-06-02 10:41:19 CEST [981]: [20-1] DETAIL:  The failed archive command was:
cp -i pg_xlog/000000010000000000000001
/var/lib/pgsql/9.3/archives/000000010000000000000001 </dev/null
```

```
2014-06-02 10:41:19 CEST [981]: [21-1] WARNING:  archiving transaction log file
"000000010000000000000001" failed too many times, will try again later
cp: cannot create regular file
`/var/lib/pgsql/9.3/archives/000000010000000000000001': No such file or
directory
(...)
```

- Utilisation de pgBadger
  - génération d'un rapport pgBadger

```
pgbadger postgresql-Mon.log

scp root@192.168.1.12://var/lib/pgsql/9.3/data/pg_log/out.html .
```

- analyse des informations

Le rapport HTML généré ne présente pas beaucoup d'informations exploitables. L'onglet *Events* permet d'avoir un rapport sur les erreurs rencontrées. On voit surtout des erreurs d'archivage (niveau `WARNING`) :

```
WARNING:  archiving transaction log file "..." failed too many times, will try
again later
cp: cannot create regular file `/var/lib/pgsql/9.3/archives/...': No such file
or directory
```

- correction des problèmes de configuration

Le paramètre `archive_command` pose problème. La commande d'archivage tombe en erreur car elle ne peut pas créer le fichier à l'endroit souhaité. La raison est simple : le répertoire d'archivage n'existe pas.

```
# mkdir /var/lib/pgsql/9.3/archives
# chown postgres:postgres /var/lib/pgsql/9.3/archives
# ls -ld /var/lib/pgsql/9.3/archives
drwxr-xr-x 2 postgres postgres 4096  2 juin  11:01 /var/lib/pgsql/9.3/archives
```

Le processus d'archivage va refaire une tentative d'archivage un peu plus tard et normalement il ne montrera plus d'erreurs dans les logs. L'état du processus montrera le nom du dernier WAL archivé :

```
postgres   981   974  0 10:39 ?        00:00:00 postgres: archiver process
last was 000000010000000000000003
```

Test de performance

- Lancez le collecteur pgCluu

Dans le *home* de postgres, créez un répertoire pgcluu :

```
mkdir ~postgres/pgcluu
```

Déclenchez le collecteur comme démon, avec un intervalle de collecte de dix secondes :

```
pgcluu_collectd -D -i 10 ~postgres/pgcluu
```

À noter que cet intervalle de collecte est trop élevé pour une situation de production normale.

Notez le PID retourné par la commande, par exemple :

```
LOG: Detach from terminal with pid: 2737
```

- Déclenchez un test de performance avec pgbench
  - 32 clients simultanés sur 8 threads
  - pendant 2 minutes
  - en lecture seule (uniquement des SELECT)

```
$ pgbench -c 32 -j 4 -T 120 -S pgbench
starting vacuum...end.
transaction type: SELECT only
scaling factor: 100
query mode: simple
number of clients: 32
number of threads: 1
duration: 120 s
number of transactions actually processed: 188604
tps = 1570.657672 (including connections establishing)
tps = 1571.966186 (excluding connections establishing)
```

- Arrêtez le collect pgCluu

```
pgcluu_collectd -k
```

pgCluu doit retourner le message suivant :

```
OK: pgcluu_collectd exited with value 0
```

- Créez un rapport pgCluu

Le répertoire ~postgres/pgcluu contient plusieurs fichiers CSV, un fichier texte et un fichier de données sar. Ils sont exploités par la commande `pgcluu` pour créer un rapport HTML :

```
$ cd ~postgres/pgcluu
$ mkdir report1
$ pgcluu -o report1 .
```

Le répertoire `report` comporte maintenant plusieurs fichiers HTML qui forme le rapport.

- Examinez le rapport

Il faut récupérer le rapport sur son poste *physique* et l'ouvrir avec Firefox.

```
scp root@192.168.1.12:/var/lib/pgsql/pgcluu/report.tar.bz2 /tmp
tar xjf /tmp/report.tar.bz2
firefox /tmp/report/index.html
```

Ce test en lecture seule est assez simpliste et ne permet pas de tester l'impact de l'ensemble des paramètres sur la performance de l'instance PostgreSQL. Cependant, la base de données est suffisant grande pour se rendre compte de l'activité du cache de PostgreSQL ainsi que des composants systèmes (CPU, mémoire, disques).

Tout d'abord, l'onglet Cluster/Cache utilization montre le taux d'utilisation du cache de PostgreSQL qui est de 50% environ durant le test. La courbe montre également beaucoup de *cache miss*, c'est à dire des lectures faites en dehors du cache, donc soit dans le cache du système, soit sur le disque.

Explorez les autres onglets. La partie base de données

- Améliorez la configuration et redémarrez PostgreSQL

Le paramètre `shared_buffers` mérite d'être augmenté pour réduire les contentions sur le cache de PostgreSQL et ainsi favoriser les lectures en cache. On l'augmente habituellement à 25% de la RAM, dans la limite de 8Go. Ici, la machine virtuelle dispose de 1Go de RAM, on positionnera donc `shared_buffers` à 256 Mo dans `/var/lib/pgsql/9.3/data/postgresql.conf` :

```
shared_buffers = 256MB
```

PostgreSQL doit être redémarré.

- Lancez le collecteur pgCluu

Supprimez ou archivez d'abord les données de tests précédentes et créez un répertoire pour un second rapport :

```
rm ~postgres/pgcluu/*
mkdir ~postgres/pgcluu/report2
```

Relancez le collecteur :

```
pgcluu_collectd -D -i 10 ~postgres/pgcluu
```

- Déclenchez un test de performance avec pgbench avec les mêmes critères :

```
pgbench -c 32 -j 4 -T 120 -S pgbench
```

- Arrêtez le collect pgCluu

```
pgcluu_collectd -k
```

pgCluu doit retourner le message suivant :

```
OK: pgcluu_collectd exited with value 0
```

- Créez un rapport pgCluu

```
$ cd ~postgres/pgcluu
$ pgcluu -o report2 .
```

- analyse du rapport
  - correction de la configuration si des éléments sont mis en lumière

L'onglet *cache utilization* montre que la situation s'est bien améliorée mais l'ensemble des données lues ne tiennent pas dans le cache de PostgreSQL. La principale différence se verra au niveau des statistiques I/O sur les tables (onglet *Databases/pgbench/Table statistics/Table I/O stats*) : lors du premier test, la courbe des lectures de pages d'index (*idx_blks_read*) suivait la courbe des lectures de pages de tables (*heap_blks_read*). Quasiment aucune lecture de table n'était réalisé en cache (*heap_blks_hit*), mais cependant beaucoup de lectures de pages d'index se fait malgré tout en cache (*idx_blks_hit*). En effet, les feuilles racines de l'index sont les pages le plus fréquemment accédées lors d'un lecture par index, elles ont donc tendance à rester dans le cache au détriment des autres pages de données. Le second rapport montre les améliorations observées : on observe beaucoup moins de lectures d'index, l'essentiel étant maintenant fait en cache. Le nombre de blocs de données lus en cache a aussi augmenté.

Les améliorations ont donc portées leurs fruits pour réduire les contentions sur

le shared_buffers.

Un test en lecture/écriture permettrait de mettre en lumière une contention sur `wal_buffers`. Les performances doublent s'il est positionné à 16MB, soit la taille d'un fichier WAL. La configuration proposée montre bien d'autres défauts,

Supervision Nagios

- Étude de la pré-configuration disponible

Dans le répertoire *home* de root, visualisez le fichier `atelier.cfg`.

Un groupe d'hôte nommé `machines_atelier` est déclaré :

```
define hostgroup {
        hostgroup_name       machines_atelier
        alias                machines_atelier
}
```

Un modèle de configuraion d'hôte `machines_atelier_grp` est déclaré :

```
define host {
        use                 custom-host
        name                machines_atelier_grp
        hostgroups          machines_atelier
        register            0
        check_command       check_ssh
}
```

Le modèle fait appartenir les hôtes qui héritent de cette configuration minimale des propriétés suivantes :

- utilise le modèle (*template*) `custom-host`,
- l'hôte est automatiquement membre du groupe `machines_atelier`,
- la commande de vérification de disponible est celle nommée `check_ssh`,
- l'entrée est spécifiée comme un modèle et non un hôte avec `register 0` (voir http://nagios.sourceforge.net/docs/3_0/objectinheritance.html).

Un hôte nommé `atelier1` est déclaré. Il utilise le template `machines_atelier_grp`. Il a pour adresse `localhost` :

```
define host {
        use                 machines_atelier_grp
        host_name           atelier1
        alias               atelier1
        address             localhost
}
```

Ensuite un certain nombre de sondes doivent être déclarées pour cet hôte. Par exemple la sonde permettant de vérifier le nombre de backends connectés :

```
define service {
        use                 t_check_postgres_backends
        host_name           atelier1
}
```

La liste des sondes check_postgres prédéclarées se trouve dans le fichier **/etc/nagios/conf.d/check_postgres.cfg**.

À noter également la présence d'un fichier `~nagios/.pg_service.conf` qui définie un service nommé `superviseur` pour pouvoir se connecter à la base de

données.

- mise en place des sondes spécifiques PostgreSQL pour l'instance locale

Le fichier `~root/atelier.cfg` peut-être repris comme base de départ pour tester la supervision de l'instance locale. En tant que `root`, exécutez :

```
cp ~root/atelier.cfg /etc/nagios/conf.d
```

Démarrez nagios :

```
/etc/init.d/nagios start
```

Dans votre navigateur Web, tapez l'adresse IP de votre machine virtuelle, suivi de `/nagios`, par exemple :

```
http://192.168.1.15/nagios
```

Vérifiez les sondes disponibles.

- lancer différents tests de performances en lecture/écriture avec pgBench

```
pgbench -c 32 -j 4 -T 120 pgbench
```

- vérifier les différentes alertes pouvant se déclencher

Plusieurs alertes devraient se déclencher : *PGSQL - Wal files*, *PGSQL - Database size*, *PGSQL - Disk space* et *PGSQL - Hit ratio*. Trois sondes parmi ces quatre sont en alerte simplement par le fait d'un problème de configuration.

- affiner la configuration (seuils) des différentes sondes en fonction du dimensionnement de la machine virtuelle

La sonde ***PGSQL - Wal files*** est en erreur simplement car elle est mal configurée. Les seuils warning et critical ne correspondent pas à la configuration courante de PostgreSQL. Cette sonde est chargée de comptabiliser les fichiers WAL présents dans le répertoire `$PGDATA/pg_xlog` et de déclencher une alerte selon les seuils passés en paramètre. Le nombre de WAL habituellement présents dans le répertoire `$PGDATA/pg_xlog` est déterminé par la formule suivante :

```
(2 + checkpoint_completion_target) * checkpoint_segments + 1
```

Elle est souvent simplifiée en `3 * checkpoint_segments` car elle `checkpoint_completion_target` est souvent configuré à 0.9.

Partant de cela, il faut déterminer le paramètre `checkpoint_segments` avec la commande SQL `SHOW checkpoint_segements`. La valeur est celle par défaut, 3. Donc, selon la formule simplifiée, il devrait y avoir habituellement 9 WAL présents dans `$PGDATA/pg_xlog`. C'est le seuil d'alerte de niveau WARNING. Le seuil d'alerte de niveau CRITICAL dépend du contexte applicatif. On choisit de le positionner à 12. La sonde sera donc redéfinie ainsi :

```
define service {
        use                 t_check_postgres_wal_files
        host_name           atelier1
        _OPTION_THRESHOLD   -w 9 -c 12
}
```

La prise en compte de la modification est réalisée par rechargement de la configuration Nagios :

```
/etc/init.d/nagios reload
```

La sonde ***PGSQL - Database size*** émet une alerte lorsqu'une base de données occupe plus d'un certain espace disque. Actuellement, la sonde se déclenche dès qu'une base occupe plus de 10 Mo. Or, la base pgbench occupe plus de 1,4 Go, les seuils doivent être adaptés :

```
define service {
        use                 t_check_postgres_database_size
        host_name           atelier1
        _OPTION_THRESHOLD   -w '1500MB' -c '1600MB'
}
```

Enfin, la sonde ***PGSQL - Disk space*** nécessite plus de travail. Cette sonde ne peut pas être exécutée en tant qu'utilisateur `nagios`. Il est nécessaire de modifier la définition de la commande pour qu'elle s'exécute en tant qu'utilisateur `postgres` :

```
define command {
        command_name            check_postgres_disk_space
        command_line            sudo -u postgres
/usr/local/bin/check_postgres.pl --language=en --showtime=0
--dbservice=$_HOSTOPTION_DBSERVICE$ --action disk_space
$_SERVICEOPTION_THRESHOLD$
}
```

Les deux lignes suivantes doivent être ajoutée dans le fichier `/etc/sudoers` :

```
nagios  ALL=(ALL)       NOPASSWD: ALL
Defaults:nagios !requiretty
```

Un fichier .pg_service.conf doit être créé dans le home de postgres avec le contenu suivant :

```
[superviseur]
user=postgres
dbname=postgres
host=/tmp
port=5432
```

La configuration de Nagios doit bien entendue être rechargée pour prendre tous ces changements en comptes.

- valider cette configuration à l'aide de nouveaux tests de performance pgBench