Tests de performances de systèmes de fichiers pour une application PostgreSQL

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 27 avril 2009

https://www.bortzmeyer.org/tests-disques-postgresql.html

À chaque fois qu'on installe et configure des machines pour une application spécifique, on se pose la question des meilleurs réglages pour tirer les performances maximales du matériel disponible. On peut lire alors plein de pages Web et de blogs remplis d'avis péremptoires mais, au bout du compte, il faut toujours faire des **mesures**, ne serait-ce que parce que les réglages idéaux dépendent de l'application. Ce qui accélère une application peut en ralentir une autre. L'informatique est une science expérimentale. Ici, je devais installer une machine pour une application faisant un usage intensif du SGBD PostgreSQL mais avec peu de concurrence (presque toujours un seule client), sur Linux.

Le matériel utilisé est un serveur Dell PowerEdge un peu vieux, un 2970 équipé de deux processeurs Opteron "*dual-core*" et de huit gigas de RAM. Le contrôleur des disques est un PERC 5/i. Les disques sont des Dell ST9300603SS, configurés en RAID 5. Le noyau du système est un Linux 2.6.26. Le système est Debian 5.0 « lenny ». PostgreSQL est en version 8.3.7.

Je teste avec plusieurs systèmes de fichiers différents et plusieurs algorithmes d'ordonnancement des requêtes d'E/S. Les systèmes de fichiers testés sont ext2, ext3 (ext4 ne semble pas encore présent dans le noyau livré par Debian), JFS et XFS.

Pour les tests génériques des disques, j'utilise l'excellent programme bonnie <http://www.coker.com.au/bonnie++/> qui teste de nombreuses fonctions du système de fichiers, comme la création d'un grand nombre de fichiers (ce qui ne reflète pas vraiment ce que fait un SGBD).

Pour les tests spécifiques de PostgreSQL, j'utilise pgbench <http://developer.postgresql.org/pgdocs/postgres/pgbench.html>, qui permet de mesurer le nombre de transactions faisables par seconde, et dispose de plusieurs options pour essayer de reproduire le mieux possible la charge réelle.

Pour effectuer le test plus facilement et surtout pour le rendre reproductible, j'ai fait un petit programme (en ligne sur https://www.bortzmeyer.org/files/test-disk-linux-postgresql.

sh) qui effectue les tests Bonnie et pgbench pour les différents systèmes de fichiers. Attention, ce programme ne mesure que la performance et il y a bien d'autres critères pour évaluer un système de fichiers comme la sécurité (ne pas perdre de données). L'utilisation prévue ne demandait pas une grande fiabilité, donc je me suis focalisé sur la vitesse.

D'autre part, faire tourner le programme deux fois avec les mêmes paramètres montre des différences allant jusqu'à 5 % (la machine était peu chargée mais néanmoins active pour d'autres tâches, ce qui est en général déconseillé pour des mesures) donc il ne faut pas prendre trop au sérieux les petites variations.

Testons d'abord avec l'algorithme d'ordonnancement des E/S par défaut, CFQ :

```
% cat /sys/block/sdb/queue/scheduler
noop anticipatory deadline [cfq]
```

Les résultats complets sont disponibles dans (en ligne sur https://www.bortzmeyer.org/files/test-disk-linux-postgresql-results-cfq.txt). Que voit-on ? Pour les tests génériques avec bonnie, en quantité de données, ext2 et XFS (quasiment ex-aequo) gagnent de peu devant JFS, mais ext2 consomme bien plus de CPU. En quantité de fichiers, XFS gagne. Pour ext3, pour des raisons que j'ignore, bonnie a sauté les tests sur les fichiers. Pour les tests spécifiques de PostgreSQL, avec pgbench, JFS l'emporte (mais les différences sont faibles).

Les différences sont plus marquées (en ligne sur https://www.bortzmeyer.org/files/test-disk-linux-
txt) si on augmente le nombre de clients PostgreSQL concurrents mais cela reflèterait moins notre usage réel. (Avec 15 « clients », on a typiquement deux fois plus de transactions par seconde, sauf pour ext3 qui est plus lent dans ces conditions.)

Le principe des algorithmes d'ordonnancement comme CFQ est de réarranger les requêtes dans un ordre qui minimise le travail pour le disque. Mais, si une machine virtuelle tourne sur la machine physique, elle aura déjà fait cette optimisation et la machine physique, qui ne sait pas ce qui se passe dans la machine virtuelle, a peu de chances de pouvoir améliorer les choises. C'est pourquoi on dit souvent que le meilleur algorithme dans la machine physique est noop, l'algorithme qui ne fait rien et passe les requêtes inchangées (FIFO). Même chose pour le cas où on utilise un SGBD, celui-ci ayant déjà ordonné ses requêtes d'E/S intelligemment (voir par exemple une intéressante discussion sur ce sujet sur la liste postgresql-performance <http://www.nabble.com/linux-deadline-i-o-elevator-tuning-td22971586
html> et une autre également portée sur l'algorithme deadline <http://archives.postgresql.org/pgsql-performance/2008-04/msg00148.php>). Est-ce vrai qu'on peut améliorer les performances ainsi ? Testons d'abord avec noop :

```
# echo noop > /sys/block/sdb/queue/scheduler
```

Pour ces tests, les résultats (en ligne sur https://www.bortzmeyer.org/files/test-disk-linux-postgr
txt) ne sont pas probants. Seul ext2 voit une nette augmentation des performances.

Et avec l'ordonnanceur deadline ? pgbench va en fait moins vite (en ligne sur https://www.bortzmeyer.org/files/test-disk-linux-postgresql-results-deadline.txt). Il est probable que le changement d'ordonnanceur serait plus efficace dans le cas de nombreux accès concurrents non synchronisés.

Merci à Malek Shabou pour son aide et ses suggestions.

Des articles intéressants sur le sujet de l'ordonnancement des E/S sur Linux :

— *elevator=noop* <http://lonesysadmin.net/2008/02/21/elevatornoop/> et *"VMware Server Tips'n'Tricks"* <http://jaysonrowe.com/2008/01/04/vmware-server-tipsntricks/>, explorent les conséquences de la virtualisation sur l'ordonnancement des E/S,
— *"Choosing an I/O Scheduler for Red Hat[Caractère Unicode non montré [1]] Enterprise Linux[Caractère*

1. Car trop difficile à faire afficher par LaTeX

Unicode non montré] 4 and the 2.6 Kernel" <http://www.redhat.com/magazine/008jun05/features/schedulers/>, article détaillé, et pas spécifique à RedHat,

— "*Linux Change The I/O Scheduler For A Hard Disk*" <http://www.cyberciti.biz/faq/linux-change-io-sch>, court résumé des instructions de base, et "*LinuxIoScheduler*" <http://www.wlug.org.nz/LinuxIoScheduler>, bonne documentation générale.

— Dans les sources du noyau Linux, répertoire `Documentation/`, les documents (très pointus !) `block/as-iosched.txt`, `kernel-parameters.txt` et `block/switching-sched.txt`.

— "*A Tweaker's Guide to Solid State Drives (SSDs) and Linux*" <http://blogs.zdnet.com/perlow/?p=9190> est surtout consacré aux SSD mais plusieurs leçons peuvent être appliquables à tous les disques.

— Spécifiquement sur PostgreSQL, on peut lire "*Performance Optimization*" <http://wiki.postgresql.org/wiki/Performance_Optimization>, liste de liens vers des articles sur l'optimisation de ce SGBD, avec une section "*Disk and hardware setup*" qui nous concerne directement. Dans cette section, j'ai notamment utilisé "*Is that performance I smell ? Ext2 vs Ext3 on 50 spindles, testing for PostgreSQL*" <http://www.commandprompt.com/blogs/joshua_drake/2008/04/is_that_performance_i_smell_ext2_vs_ext3_on_50_spindles_testing_for_postgresql/>.

— Le livre sur Oracle, "*Pro Oracle database 10g RAC on Linux*" <http://books.google.com/books?id=GjYTbJTIr54C&pg=PA143&lpg=PA143&dq=linux+elevator+noop&source=bl&ots=h9ioBYxU5a&sig=Ql-gWfcEFAmubODJmUJTwt6svRE&hl=en&ei=fRf0SeC8L6O7jAfzz6jMDA&sa=X&oi=book_result&ct=result&resnum=9#PPA144,M1>, bien que consacré à un autre SGBD, contient des informations intéressantes.

Pour les tests avec PostgreSQL, Erwan Azur me suggère aussi Sysbench <http://sysbench.sourceforge.net/>. Mais il ne compile pas sur mes machines Debian :

```
% ./configure --with-pgsql --without-mysql
% make
...
/bin/sh ../libtool --tag=CC   --mode=link gcc -pthread -g -O2      -o sysbench sysbench.o sb_timer.o sb_options.
../libtool: line 838: X--tag=CC: command not found
../libtool: line 871: libtool: ignoring unknown tag : command not found
../libtool: line 838: X--mode=link: command not found
../libtool: line 1004: *** Warning: inferring the mode of operation is deprecated.: command not found
../libtool: line 1005: *** Future versions of Libtool will require --mode=MODE be specified.: command not found
../libtool: line 2231: X-g: command not found
...
```