

MySQL: performances E/S

- Intervenant: <http://makarevitch.org/>
- Objectif: améliorer les performances d'une instance de MySQL gérant de grand volumes de données
- Périmètre: MySQL (surtout 5 et INNODB) sous Unix (Linux) et autres, sur *fs* plutôt que *raw*
- Approche: large spectre ; mi-théorique, mi-appliqué ; générique (peu spécifique à MySQL)

OLTP

- **On-Line Transaction Processing**: genre d'activité d'un moteur de base de données: accès aléatoires denses (nombreuses requêtes par unité de temps) portant chacune sur un faible volume de données stockées dans une base dont la taille est un multiple, souvent supérieur à 1000, de celle de la mémoire vive du serveur. Cf. <http://jargonf.org/wiki/OLTP>
- Souci d'augmenter les IOPS!

OLAP (décisionnel)

- Un serveur destiné au *décisionnel* (**O**n-**L**ine **A**nalytical **P**rocessing) nécessite plutôt de hauts débits tendus car de nombreuses requêtes y impliquent chacune l'analyse, donc la lecture, d'une importante fraction de l'ensemble des données.
- Besoins similaires à ceux du *streaming*: du débit plutôt que des IOPS

Goulet d'étranglement

- Aucun problème lorsque tout tient dans le cache (*buffer*)
- Sinon tout dépend du disque: temps d'accès et débit

Comparer des performances

- Unités (requêtes/seconde) trompeuses car elles dépendent du contexte
- TPC: génériques (OK pour vendeurs)
- Comparer n'est possible qu'au niveau applicatif

Définition d'objectif

- L'exprimer en:
 - Nombre moyen de requêtes/seconde
 - Latence (délai entre soumission d'une requête et expédition du résultat):
 - Maximale,
 - Moyenne (par ex arithmétique),
 - Écart-type.

Comment tester

- Définition d'un test réaliste:
 - Produisant un résultat correspondant à l'objectif,
 - Par rejeu de véritables requêtes avec des données représentatives,
 - Sur machine par ailleurs inutilisée,
 - Script de lancement de test (restauration d'état par snapshot...),
 - Vider caches (Linux: `echo 3 > /proc/sys/vm/drop_caches`)
 - Outil de collecte des mesures

Préalables (système)

- Optimisations classiques effectuées:
 - BBU (*battery backup unit*)
 - Options de montage (*noatime, commit=...*)
 - *swappiness*
 - Applicatives...

Préalables (MySQL)

- Optimisations classiques effectuées:
 - *innodb_buffer_pool_size*,
innodb_flush_log_at_trx_commit
(controversé), *innodb_log_buffer_size*,
innodb_log_file_size,
innodb_thread_concurrency, *transaction-isolation* ...
Utiliser Mysqltuner, mysqlreport...
 - Applicatives (index, SQL_NO_CACHE,
LOW_PRIORITY, INSERT_DELAYED...)

Approche

- Mettre ainsi au point grâce au test **puis** définir l'objectif
- Ne modifier qu'un seul paramètre, puis tester et associer (commentaire) les résultats à la modification

Bien tester 1/2

- Test en *runlevel* "single-user"
- Tout logiciel (démon compris) non nécessaire est stoppé
- Volume de données **réaliste** (saturation des disques)
 - Linux: utiliser le paramètre noyau *mem* avant les tests intensifs de disque (peu/pas de cache, cela rend possible de réduire le temps de test)... mais pas trop (metadonnées du *fs* doivent tenir en RAM)

Bien tester 2/2

- Disques: *acoustic parameters* OFF, vérifier mode d'exploitation (SATA-2...)
- Éviter de partager des lignes d'IRQ
- Linux: *swapoff -a*, de sorte que Linux ne puisse ménager le *buffercache*
- Linux: */bin/raw* , afin de négliger le *buffercache* (danger!)
- Laisser le cache de l'interface disque, il optimise en réordonnant

Outillage autonome (pour mémoire)

- lozone
- seeker
- Tiobench
- Fsbench
- Postmark
- OSDLBT
- SuperSmack

Outillage (plusieurs niveaux)

- Mysqlreport
- *top, iostat, dstat, vmstat...*
- *Blktrace*

Paramètres MySQL

Sûreté

- *innodb_checksums*
- *innodb_doublewrite*

Paramètres MySQL

Gestion de charge

- *innodb_file_per_table* (dépend du système de fichiers)
- *innodb_open_files* (nombre de fichiers simultanément ouverts, attention à l'équivalent côté système)
- *innodb_table_locks*

Paramètres MySQL

Parallélisation

- *innodb_commit_concurrency*
- *innodb_file_io_threads* (*innodb_thread_concurrency*)

Paramètres MySQL

Flush

- *innodb_flush_method* (pas sous Windows)
 - *O_DIRECT* ou *O_DSYNC* (ce dernier n'est pas partout stable)

Lecture / écriture

- Les données à écrire sur disque sont accumulées dans le cache... jusqu'à un certain point
- BBU: cache non-volatile (cohérence des données)
- Cache de la logique du disque
- Read-ahead:
 - Hard (disque)
 - Soft (OS voire applicatif)

Organisation physique: RAID

- RAID5 ou 6 peuvent suffire si les écritures sont rares, RAID10 s'impose sinon
- Linux: tester *md* en "*RAID10 far*"

Organisation physique

- Définir finement la taille de bloc/*slice*/*chunk* afin (trop gros) de ne pas lire/écrire inutilement ni (trop petit) de mobiliser plusieurs *spindles* lorsqu'un seul suffirait. Vaut pour tout: LVM, logique RAID, le *fs*...
- Attention à LVM (réduit les performances si décalage, indirection supplémentaire), toutefois utile (snapshots)

Ordonnanceur d'E/S (I/O scheduler)

- Rôle et effets
- Attendre afin de réordonner les requêtes...
mais pas trop!
- Linux: *deadline* ou *CFQ* (*ionice*)

Ordonnanceur d'ES

Paramètres

- Paramètres
 - Linux (dans `/sys/block/périph/device/`):
 - *queue_depth* (nombre de requêtes en queue dans le noyau)
 - *nr_request* (nombre de requêtes simultanément expédiées à l'interface)
 - *read_expire*, *write_expire*, *write_starved*

TCQ et NCQ

- Réordonnancement par le disque
- Balise de priorité honorée par le disque
(rarement employée par OS et applicatifs,
faute de ressources dans les API)
- Gain potentiel: seule la logique du disque
sait à tout moment où se trouve une tête

Pistes

- *Raw*
- Sans journal de *fs*
- Journal de *fs* sur spindle(s) spécifiques
- *Logs (binlog et traces) MySQL* sur spindles spécifique(s)
- *aio*
- Partitionnement de bases