TheCodingMachine
TCM://

Améliorer les performances de vos projets web

—

Mettre en place une démarche d'amélioration des performances sur un environnement LAMP.

David NEGRIER, Directeur Technique
Kévin NGUYEN, Associé
Arthmaël POLY, Chef de projer

Juin 2018

Introduction

Une histoire qui se répète souvent ...

Vous commencez à récolter le fruit de vos efforts, votre site web est un succès - félicitations ! Cependant, vous sentez que votre site devient plus lent et un beau jour, avec encore un peu plus de succès et donc plus de trafic, votre site web ne répond plus... sur une page, sur une fonctionnalité particulière, ou encore de manière globale, en cas de trop forte affluence.

Lorsque l'on rencontre des problèmes de performance, il est souvent trop tard. Le site est déjà en production et l'activité de la société peut être fortement pénalisée. Lorsqu'un site devient lent, plus de 50% des visiteurs arrêtent leur navigation. Pour l'équipe du projet, trouver une solution rapidement devient la première préoccupation.

Or, en matière d'optimisation, les solutions sont très nombreuses. Dans la plupart des cas, des solutions sont appliquées les unes après les autres dans la précipitation avec une dépense d'énergie incroyable : les équipes travaillent la nuit, le week-end ... "mais cette fois, c'est sûr, on a la bonne solution ! "

Dans la plupart des cas, il faut comprendre que ces solutions uniques et appliquées sans discernement vont vous faire perdre un temps précieux. Même si souvent, on finit quand même par trouver, avec un peu de chance et au prix d'efforts considérables.

Notre conviction est que l'application d'une démarche structurée et rationnelle est la clé pour améliorer les performances de votre site et passer ce cap difficile. Même (voire surtout) s'il s'agit d'une situation de crise.

L'importance de mettre en place un plan d'amélioration continu des performances

Au-delà de l'aspect douloureux des mauvaises performances, il est essentiel de mettre en place un plan d'amélioration des performances pour au moins deux raisons :

1. De nombreuses statistiques sur les sites e-Commerce montrent que réduire les temps d'affichage des pages réduit le taux d'abandon de panier ;

2. Les moteurs de recherche prennent de plus en plus en compte le confort des utilisateurs. Une des seules mesures à leur disposition est le temps de chargement des pages. Améliorer les performances de son site permet donc d'améliorer son référencement naturel.

A qui s'adresse ce livre blanc et comment l'utiliser ?

Ce livre blanc s'adresse à l'ensemble des acteurs faisant face à des problèmes de performances sur des applications web en production. NOTE : les conseils portent sur les architectures LAMP (PHP MySQL) mais la plupart s'appliquent aussi aux applications web de type Java/J2EE ou Ruby.

Nous nous sommes fixés l'objectif de faire un guide qui soit le plus pratique et efficace possible. Nous n'avons donc pas cherché à citer l'ensemble des outils de diagnostic ni à répertorier l'ensemble des solutions. Avec cet ouvrage vous devriez facilement comprendre dans quel domaine rechercher une solution et trouver cette solution dans la plupart des cas.

Il est rare que les performances d'un site (même si c'est dommage) soient réellement étudiées avec attention au démarrage du projet, souvent pour des questions de budget. Mais vous pouvez malgré tout utiliser cet ouvrage avant de mettre votre site en production. La démarche est identique.

Pourquoi ce livre blanc ?

De nombreux ouvrages ou articles expliquent des points précis pour améliorer les performances. Ils détaillent par exemple comment améliorer les performances de PHP, comment paramétrer MySQL ou bien encore quelles sont les meilleures architectures techniques. Pourtant, rares sont ceux qui traitent de la démarche d'ensemble qu'il faut suivre pour améliorer les performances une fois l'application déployée.

Les performances de votre site dépendent étroitement de votre activité et de l'architecture technique que vous avez mise en place. Si vous avez un site qui ne présente que des contenus, le poids des pages et la gestion du cache seront essentiels et ce qui concerne les optimisations des bases de données ou du code n'auront pas ou peu d'impacts pour vous. Cependant, toute la démarche, l'outillage et de nombreuses solutions pourront vous être utiles.

Heureusement, les architectures LAMP sont suffisamment diffusées et classiques pour être, dans la plupart des cas, correctement développées. Il est rare qu'une application soit structurellement tellement mal conçue qu'il faille reprendre le projet depuis le début.

En revanche, peu de société disposent à la fois de l'expérience et des connaissances permettant de solutionner ces problèmes de performance. Ils sont trop peu fréquents pour en faire une spécialité et les problèmes rencontrés ont des solutions dans des domaines de compétence très différents (matériel, base de données, développements etc.).

Mettre en place un plan d'amélioration des performances exige de la sérénité. Ce qui est rarement le cas lorsque des problèmes de performance se posent. Cet ouvrage vise donc à partager l'expérience de The Coding Machine. Il indique les différentes étapes permettant de conduire les différentes investigations nécessaires et présente les solutions envisageables.

1 Introduction, démarche & diagnostic

Quelques réflexions pour bien commencer

1. *Ne recevoir aucune chose pour vraie tant que son esprit ne l'aura clairement et distinctement assimilé préalablement.*
2. *Trier ses difficultés afin de mieux les examiner et les résoudre.*
3. *Établir un ordre de pensées, en commençant par les plus simples jusqu'aux plus complexes et diverses, et ainsi de les retenir toutes et en ordre.*
4. *Passer toutes les choses en revue afin de ne rien omettre.*

Descartes, discours de la méthode.

Toutes ces étapes sont rigoureusement les mêmes lors de la mise en place d'un plan d'amélioration des performances :

1. **Ne pas appliquer de solutions préconçues**

   Vous risquez de perdre du temps à mettre en place une solution qui ne résout pas votre problème. La première étape est de reproduire les conditions dans lesquelles les performances se dégradent et d'identifier de manière certaine le problème.

2. **Commencer dans l'ordre**

   Généralement il y a un problème dominant. Il ne sert à rien de résoudre les points secondaires avant le premier. Les performances sont souvent améliorées par les actions les plus simples. Par exemple, s'il y a un problème sur l'accès des données, on peut obtenir un facteur 100 en travaillant sur les index de la base de données tandis que changer les serveurs permettrait d'obtenir un facteur 4.

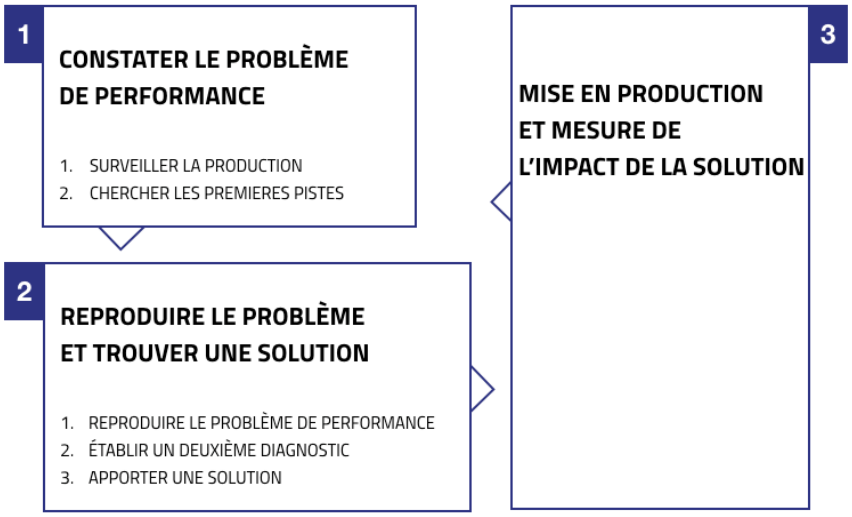
3. **Mettre en place les solutions jusqu'au bout**

   Dans la plupart des cas, il ne se pose pas un mais plusieurs problèmes de performance. Vous devez les identifier et les résoudre un par un. Vous vous rendrez certainement compte que trouver une première solution n'est que le début de la démarche. L'écueil dans ce cas-là est de croire que ce n'était pas la bonne solution et de l'abandonner pour en chercher une autre plus "universelle". C'est une erreur. Les performances s'améliorent en accumulant les solutions.

4. **Procéder de manière itérative**

   Les performances vont s'améliorer après ces premières actions, mais cela ne suffira peut-être pas. Il est important de mesurer le progrès et d'itérer.

Vue d'ensemble de la démarche



1. **Constater le problème de performance**

Cette étape consiste à chercher les premières pistes d'optimisation. Elle est conduite à l'aide des outils de surveillance de la production. L'objectif est de trouver une orientation à la recherche de solutions dans un des quatre domaines suivants :

- Applicatif (Apache – PHP)
- Base de données
- Réseau
- Infrastructure matérielle

Dans les premières itérations de la démarche, certaines solutions sont évidentes. Dans le cas d'un problème sur une page ou une fonctionnalité précise, il est souvent simple d'identifier rapidement d'où vient le problème. Aussi, si la solution est évidente, il suffit de proposer une correction et de passer directement à l'étape 3.

2. **Reproduire le problème et trouver une solution**

La deuxième étape consiste à reproduire les conditions dans lesquelles les performances se dégradent. Cela permettra de travailler beaucoup plus rapidement sur le problème, et parfois même sans solliciter l'environnement de production.

Une fois que les conditions dans lesquelles les performances ne sont pas bonnes sont identifiées, il convient de mettre en œuvre et de tester les différentes solutions.

3. **Mise en production et mesure de l'impact**

Deux éléments perturbent la mesure exacte de l'optimisation apportée :

- les différences entre les environnements : pour des raisons économiques, il est rare d'avoir un environnement de pré- production parfaitement identique à celui de production ;
- les conditions dans lesquelles les performances se dégradent : il est parfois impossible ou trop coûteux de reproduire ces conditions. Par exemple, si les performances se dégradent lors d'un pic d'audience, reproduire l'accès au service de nombreux utilisateurs simultanément peut être complexe.

Aussi, dans bien des cas, la seule mesure valable est faite sur l'environnement de production. C'est pour cela que mettre en place une démarche préalable a beaucoup d'importance. Elle permet de réduire le risque et de réduire le temps nécessaire à la mise en place des dispositifs d'optimisation.

Établir les premières piste d'optimisation

Un projet est un assemblage de plusieurs ressources matérielles et logicielles. Aussi, soumise à une forte charge, une de ces ressources est susceptible de limiter la performance de l'ensemble. Les outils de surveillance des infrastructures de production permettent de déterminer quelle ressource est liée à la dégradation des performances.

Parmi les ressources matérielles :

- CPU
- RAM
- Bande Passante
- Disques Durs

Parmi les ressources logicielles, on peut citer :

- Nombre de fichiers ouverts
- Nombre de cœurs utilisés
- Nombre de connexions à la base de données

Afin d'établir des pistes de solutions à étudier, il convient de mesurer l'utilisation des différentes ressources citées ci-dessus afin de déterminer quelles sont celles qui sont exploitées de manière trop intense.

Ces mesures servent aussi à comprendre sous quelles conditions et à quels moments les performances sont moins bonnes. De nombreux outils permettent d'effectuer ces mesures. Ils sont décrits dans les paragraphes suivants.

Il ne faut pas croire que la libération de la ressource la plus sollicitée permet nécessairement d'améliorer de manière spectaculaire la performance de l'ensemble. Bien souvent, produire une première optimisation va conduire à trouver qu'une autre ressource est saturée à son tour. C'est pourquoi la démarche est itérative.

Dans cette démarche il faut respecter un certain ordre. Les ressources matérielles doivent systématiquement être analysées avant les ressources logicielles (mais cela ne signifie pas que les solutions sont nécessairement matérielles). Par exemple, il faut s'assurer que l'utilisation du CPU par la base de données n'est pas critique avant de se pencher sur la configuration du nombre de connexions à la base de données.

Bon à savoir : Le disque dur doit être analysé après la RAM en raison du phénomène de SWAP qui fait porter l'excédent de mémoire utilisée par le disque dur lorsque la RAM disponible est insuffisante.

Les outils serveurs

Si vous savez à quel moment précis se produisent les problèmes de performance, vous pourrez utiliser des outils de mesure « temps-réel ». Ces outils sont pour la plupart utilisés en ligne de commande sous Linux.

1. TOP

La commande TOP est l'outil de monitoring des systèmes linux par excellence.

TOP produit une **liste des processus actifs du serveur, et détaille leur consommation de temps processeur (CPU) et de mémoire (RAM).** Il indique également la charge moyenne du système sur trois périodes : 1 minute, 5 minutes et 15 minutes.

Si cet outil ne permet pas de connaître directement l'origine du problème, ce chiffre aide à confirmer s'il s'agit d'un pic d'activité ou bien si la machine est saturée depuis un certain temps.

Il est à noter que TOP a été pensé de manière à ne pas faire partie lui-même des processus le plus consommateurs de ressources qu'il mesure.

Outil extrêmement utile, il permet d'identifier les problèmes les plus évidents sur la consommation de RAM et de CPU. En revanche, il est possible que TOP ne détecte aucun problème apparent, alors que votre machine est belle et bien ralentie. Cela se produit notamment si le problème se situe au niveau des accès disque, ou encore au niveau des ressources logicielles.

2. IO STATS

La commande IOSTATS est utilisée pour **contrôler la charge des périphériques entrée/sortie** en observant leur temps d'activité par rapport à leur taux de transfert. Cette commande est souvent utile pour harmoniser la charge lecture/écriture entre les différents disques durs.

3. IO TOP

Si l'accès en lecture/écriture de votre disque est saturé (par exemple trop de requêtes INSERT, UPDATE ou DELETE peuvent surcharger votre disque en écriture), la commande TOP peut s'avérer insuffisante comme exposé ci-dessus.

IOTOP présente, de la même manière que TOP, **les processus qui consomment le plus de lecture et d'écriture sur votre système.**

4. DSTAT

DSTAT est un outil de mesure transversal. Alors que TOP se spécialise dans l'activité CPU et IOTOP dans les accès disques, DSTAT permet de surveiller l'activité sur le serveur de manière transverse. L'utilisateur peut décider d'afficher des **indicateurs tels que l'activité CPU, l'activité des disques, l'activité réseau** … côte à côte.

Ce n'est pas l'outil le plus précis, mais il est très pratique pour comparer rapidement différentes mesures du système. De plus, il permet aisément de réaliser des logs analysables sur un tableur comme Excel.

Les outils clients

1. WEBPAGETEST.ORG

Webpagetest.org est un site Internet permettant de faire un audit du chargement d'une page web. On retrouve le diagramme chronologique des éléments téléchargés, la check-list des points courants à optimiser, un rendu de la page avec une image au bout de quelques secondes et des graphiques résumant les éléments récupérés.

2. FIREBUG

Firebug est un plugin de Firefox apprécié des développeurs web. Il permet d'inspecter le code HTML en ciblant directement l'élément dans la page, d'éditer le style CSS pour tester le résultat directement, debugger le javascript avec des points d'arrêts et un mode pas à pas et espionner le trafic XMLHttpRequest (Ajax).

Tout comme webpagetest.org, Firebug permet de disposer d'un diagramme chronologie du chargement de la page. Firebug étant un outil local, vous pouvez même l'utiliser pour tester votre application en phase de développement.

3. YSLOW ET PAGE SPEED

Yslow et Page Speed sont des extensions de Firebug, respectivement développé par Yahoo et Google, permettant d'analyser les performances du trafic réseau de la page. Ces extensions analysent le code de votre page web et proposent de nombreux conseils d'optimisation de votre code HTML et de votre serveur web (utilisation du cache HTTP, …).

Webpagetest.org et Yslow fournissent à peu près les mêmes éléments à une nuance près. Webpagetest.org permet de localiser (physiquement) d'où émane la demande. En fonction de ce lieu, les diagrammes chronologiques peuvent varier.

Les outils de monitoring

Si vous ne pouvez pas prévoir l'occurrence des problèmes de performance, il sera plus simple de se servir d'outils de monitoring afin de pouvoir accéder à l'historique des mesures de vos ressources.

1. MUNIN

Munin permet de surveiller les différentes ressources de vos serveurs. Un client est installé sur chacune des machines, et les données mesurées sont agrégées à intervalle régulier vers une machine centrale (où le serveur Munin se trouve), et présentées sous forme graphique.

Il permet, dans son installation la plus simple, de surveiller l'utilisation du disque, de la RAM, du CPU et du réseau. Cet outil est Open Source et il existe de nombreux plugins permettant de mesurer d'autres ressources, telles que :

- Les temps de réponse (http response times) ;
- La consommation CPU pour une sélection de processus (multi memory plugin)
- Le taux d'utilisation des connections MYSQL (mysql_connections)

La granularité minimale de Munin (intervalle de temps entre deux mesures) est de 5 minutes. Si vous faites face à des problèmes de performance lors de pics de charge très courts (de 5 à 15 minutes), Munin peut s'avérer insuffisant pour vous apporter des informations suffisamment détaillées.

2. NAGIOS

Nagios est un outil de surveillance de serveurs. Contrairement à Munin, Nagios n'est pas un outil de mesure de l'activité. Il est utilisé uniquement pour remonter des alertes.

Extensible avec des plugins, il peut remonter des alertes comme :

- Activité CPU trop importante
- Indisponibilité du serveur web ;
- RAM insuffisante ;
- Temps de réponses dégradés ...

Lorsqu'une alerte est remontée, l'outil peut être paramétré pour prendre des actions, comme envoyer un mail d'alerte aux administrateurs, ou encore envoyer des SMS.

Si les problèmes de performance se produisent de manière aléatoire et non reproductible, Nagios peut être utilisé pour vous alerter des problèmes rencontrés. Vous pourrez alors vous connecter au serveur pour effectuer des mesures plus approfondies en temps réel.

3. AUTRES OUTILS

D'autres outils vous permettent de connaître l'activité de votre infrastructure. Par exemple, un outil de web analytics permet de connaître approximativement le nombre de visiteurs sur votre plateforme.

D'autres outils (émergents) vous permettent d'avoir une vision encore plus précise de votre activité. Par exemple Collectd (http://collectd.org) permet de descendre sous la barre des 5 minutes imposée par Munin.

Établir les premières pistes en fonction des ressources

1. CPU/RAM

Si la piste s'oriente vers une activité critique de la RAM ou du CPU, c'est probablement parce que la commande « TOP » aura indiqué que les processus apache (PHP) ou MySQLD (BDD MySQL) consomment une quantité trop importante de ces ressources.

Selon le processus concerné, il faudra alors identifier plus précisément la ou les causes de cette activité trop importante :

- **Apache/PHP**

  Si votre problème se pose sur une page en particulier (sauf la page d'accueil où les problèmes sont souvent liés au nombre de connexions simultanées), il est utile d'analyser le code exécuté par celle-ci pour déceler un potentiel défaut.

  Si, au contraire il s'agit de l'ensemble de l'application, c'est certainement dû à un trop grand nombre d'utilisateurs. Il est alors intéressant de mettre en place des mises en cache (côté serveur et côté client) ou des traitements asynchrones afin de limiter l'engorgement lors d'un afflux trop important.

- **MySQL**

  Un très grande partie des problèmes de performance liés à la base de données peut être résolue en utilisant les index adéquats ou en mutualisant les requêtes.

  Avant cela, il faut systématiquement établir une liste détaillant à la fois la fréquence de la requête (ou du type de requête) et le coût de cette requête.

- **AUTRES PROCESSUS**

  Vous pourriez constater via « top » qu'un autre processus de votre serveur occupe la totalité de votre temps processeur. Il est alors important d'identifier le coupable. Si vous avez installé d'autres services sur votre serveur, l'un d'entre eux pourrait être le coupable. Par exemple, un problème sur un serveur de mail ou une base de données full-text, etc... pourrait déclencher cette consommation CPU qui ralentirait votre serveur.

  Si vous ne connaissez pas le nom du processus, n'hésitez pas à vous aider d'une recherche sur Google pour en apprendre plus. Et si vous êtes persuadés de n'avoir jamais installé le processus incriminé, investiguez ! Votre serveur a peut-être été piraté. C'est certainement le cas si vous laissez un accès SSH disponible depuis Internet avec des mots de passes trop simples. N'oubliez pas qu'un hacker n'a pas besoin des accès « root » pour installer et exécuter ses programmes. Un simple accès utilisateur suffit. Il peut alors transformer votre machine en serveur de fichiers ou bot, et les processus que vous voyez dans « top » sont à coup sûr néfastes. Tirez à vue !

2. BANDE PASSANTE

Pour obtenir une première piste d'optimisation, vous devez rechercher le ou les éléments qui consomment de la bande passante. Par exemple, une page qui renvoie une vidéo de 100 mo peut dégrader les performances de votre site web si elle est appelée de nombreuses fois.

La première piste à examiner sont les logs Apache afin d'observer combien consomme chaque requête et rechercher quels sont les pages ou les fichiers volumineux. En effet, les logs Apache sont assez flexibles et il est possible de les configurer pour afficher la quantité de données envoyées à chaque requête.

Lorsqu'on mesure la bande passante du serveur, il est important de garder à l'esprit que le serveur n'est pas le seul à saturer. Votre connexion saturera certainement avant. Si IOStat ne montre pas de surconsommation réseau, la raison d'un affichage lent pourrait être votre connexion internet, ou n'importe quel élément réseau entre votre client et votre serveur.

3. DISQUE DUR

Vos consommations de RAM et de CPU semblent normales et votre bande passante n'est pas saturée. Il y a de grandes chances que votre disque dur soit le facteur limitant. A moins que votre application n'effectue un très grand nombre de lecture/écriture sur le disque (ouverture et modification de fichiers, création de documents, ...), il est probable que MYSQL soit responsable.

En effet, les requêtes MySQL SELECT peuvent provoquer des accès disque en lecture (si le résultat de cette requête n'est pas en cache), et les autres types de requêtes (INSERT, UPDATE, DELETE) font systématiquement des accès en écriture.

4. AUCUN DE CES POINTS NE SEMBLE CRITIQUE ?

Le système peut être « bridé » par ses ressources logicielles ou par des latences réseau. Quelques pistes : nombre de threads MySQL, Nombre de fichiers ouverts simultanément.

Reproduire le problème et établir un diagnostic

1. REPRODUIRE LES PROBLEMES DE PERFORMANCE

Reproduire les conditions dans lesquelles les performances se dégradent est de plus en plus dur au fur et à mesure que l'on effectue des optimisations. Or, savoir si l'on fait le bon scénario de test est presque aussi important que de faire la bonne optimisation (celle qui nous permettra vraiment d'améliorer les performances sur l'environnement de production).

L'objectif est donc de simuler le fonctionnement de votre site web sur un seul des aspect (base de données, accès disque etc.).

- **AB**

Ab est un outil d'évaluation des performances de votre serveur web. Il indique le nombre de requêtes par seconde que votre application est capable de prendre en charge. Les deux paramètres principaux sont le nombre de requêtes simultanées ainsi que le nombre total de requêtes à effectuer.

Bien qu'il soit possible d'utiliser d'autres paramètres pour passer des valeurs POST, des cookies, etc.... cet outil n'est pas fait pour simuler efficacement l'activité d'un ou plusieurs utilisateurs qui naviguent sur votre site, mais plutôt pour simuler l'arrivée massive d'utilisateurs sur une page. En effet, ab n'intègre pas de notion de scénario, car il ne peut tester qu'une seule URL à la fois.

- **JMETER**

JMeter est une application que l'on peut déployer sur un poste de travail (il vaut mieux qu'il soit dédié afin que le développeur qui s'en sert puisse travailler pendant les runs de tests). C'est un outil qui permet de simuler un grand nombre de requêtes concurrentes HTTP et donc de simuler le comportement du site avec un grand nombre de visiteurs. L'outil produit une synthèse graphique des résultats du test.

Dans le cas d'une architecture LAMP, l'outil ne présente d'intérêt que pour les requêtes HTTP. En revanche, il permet de tester beaucoup d'autres éléments annexes (Serveurs de mails, connexion JDBC, etc.).

Notez que JMeter n'émule pas complètement un navigateur, contrairement à un outil comme sélénium. Il est donc plus difficile de créer un jeu de tests qui soit représentatif du comportement exact de l'application.

- **SELENIUM GRID – REPRODUCTION DE SCENARIO**

Selenium est une boite à outils permettant de réaliser des tests en simulant l'activité d'un ou de plusieurs utilisateurs sur un site web. Grâce à un plugin Firefox, vous avez la possibilité ‹ d'enregistrer › un scénario en naviguant sur le site, puis de le rejouer à volonté, en vérifiant que l'affichage des pages est conforme aux attentes.

Selenium Grid vous permet de simuler une montée en charge de votre application en jouant plusieurs fois le même scénario en parallèle. Si vous disposez de suffisamment de ressources, il vous sera possible de reproduire les problèmes de performance.

- **BATCHS DE REQUÊTES SQL**

Obtenir la liste des toutes les requêtes faites à la base de données pendant l'exécution d'un scénario représentatif, puis le rejouer vous permettra de simuler l'impact de l'application sur la charge de votre serveur de données.

Pour faire cela, vous pouvez soit écrire dans un journal les requêtes au niveau de l'applicatif, soit configurer MySQL pour tracer toutes les requêtes (paramètre ‹ log › de votre fichier my.ini ou my.cnf). Vous avez la possibilité de rejouer ce batch de façon unitaire pour mesurer le gain de performance grâce à vos optimisations, mais aussi de lancer plusieurs batchs en parallèle pour simuler un pic activité.

2. ETABLIR LE DIAGNOSTIC

Comme nous l'avons vu précédemment, la solution est parfois évidente, il n'est parfois pas nécessaire de pousser plus loin l'analyse de votre problème. En revanche, dans certains cas il faut préciser le diagnostic afin d'appliquer la correction appropriée.

Dans ce cas, il existe des outils et des démarches simples permettant d'investiguer plus en profondeur l'utilisation des ressources :

- **Logs Apache**

SYMPTOMES : Votre site répond correctement en temps normal, mais les temps de réponse se dégradent sous la charge.

Les logs Apache tracent toutes les requêtes faites au serveur apache. En les configurant correctement, vous aurez la possibilité d'obtenir le temps de réponse de votre site. En utilisant des outils de parsing appropriés (AWstats, Visitors, ou WebLogExpert), vous pourrez facilement visualiser le comportement de votre site en fonction du nombre d'utilisateurs connectés.

- **Logs MySQL**

SYMPTOMES : MySQL consomme énormément de CPU ou le disque dur sature.

Les logs MySQL retracent l'activité de votre serveur de données. Le journal de log simple (celui configuré par défaut) pourra vous renseigner sur la stabilité du serveur MySQL. Le fichier de slow queries (il est parfois nécessaire de configurer ce log) liste l'ensemble des requêtes dont la durée d'exécution est supérieure à un seuil donné (lui aussi configurable).

Les requêtes contenues dans ce fichier seront donc celles qui mobilisent le plus le serveur, et donc souvent celles qui devront être optimisées.

Exemple de configuration du log slow queries dans le fichier de configuration MySQL :

```
[mysqld]
port=3306
log_slow_queries = 1
long_query_time  = [seuil]
log-slow-queries = [chemin vers le fichier de log]
```

- **Analyse/Parsing de code – XDebug**

SYMPTOMES : PHP consomme énormément de CPU.

La fonction Profiler de XDebug est un outil qui permet d'analyser les temps d'exécution de votre code PHP. Il permet de savoir combien de temps est passé dans chacune des fonctions du code, et donc de déterminer quelle partie de code est moins performante.

- **MySQL Tuner**

MySQL Tuner est un script écrit en Perl qui vous permet d'éprouver une installation MySQL rapidement et de faire des recommandations pour améliorer les performances et la stabilité de la base de données.

Il produit un rapport détaillant :

- Les possibilités offertes par la configuration en cours
- des mesures telles que le nombre de jointures effectuées sans index, hit ratio du cache, ...
- des recommandations à la fois sur la manière d'exécuter les requêtes, et sur les paramètres de configuration à modifier.

- **Audit de la structure de la base de données**

La structure de la base de données influence fortement ses performances. Ainsi, il faut s'assurer que vos tables contiennent les index appropriés, que les clés étrangères sont en place. Vérifiez aussi que les triggers éventuels ne nuisent pas aux performances : si chaque insert dans une table implique l'exécution d'un script complexe, il vaudra mieux mettre en place des batchs SQL asynchrones (quand c'est possible bien sûr).

Si les bonnes pratiques en matière de structure relationnelle des bases de données recommandent le respect des normes (on parle de base de données normée), la complexité de relation entre les différentes tables peut nuire aux performances. Il faut toujours partir d'une base normalisée, celle-ci étant bien plus facile à maintenir. En cas de problème de performance, on effectuera une dénormalisation au cas par cas.

- **Réseau Interne / Réseau externe**

Si vous avez séparé votre serveur applicatif et votre base de données, assurez-vous au préalable que votre bande passante n'est pas consommée principalement en interne. En effet, si vous effectuez une grande quantité de requêtes ou des requêtes avec des résultats de taille importante, il est possible que soit la raison de la baisse des performances en cas de forte affluence.

De manière générale, vous devez veillez à envoyer le moins de données possible du serveur de base de données vers l'application. Écrivez les requêtes les plus sélectives possibles et faites le maximum d'agrégations dans vos requêtes SQL.

2 Améliorer la gestion de votre bande passante

Si vous disposez d'un site internet à fort trafic ou d'un serveur avec une faible bande passante, vous pouvez apporter des optimisations consistant à réduire le flux qui transite entre le site et le navigateur du visiteur.

Pour cela, il est possible :

- **D'utiliser des systèmes de cache**
- **D'éviter d'envoyer ou de déporter l'envoi de fichiers**
- **De réduire la taille et le nombre de fichiers échangés**

Avant de commencer, il faut différencier 2 types d'attente différents :

- le temps de connexion
- le temps de latence.

Un navigateur Internet doit recevoir l'ensemble du code HTML (DOM) du serveur avant de pouvoir afficher le contenu. Pour chaque élément, contenu, image, vidéo etc., le navigateur se connecte au serveur grâce au protocole TCP/IP. Ce processus prend en moyenne 70ms pour un serveur proche et peut aller à plus de 300ms pour le réseau mobile. C'est le temps de connexion.

A cela, il faut ajouter le temps de latence du serveur qui ne répondra pas toujours instantanément car il traite les demandes dans l'ordre d'arrivée. Ces temps sont irréductibles. Aussi, les navigateurs récents établissent simultanément plusieurs connexions avec le serveur. Ceci permet de paralléliser les temps d'attente.

Sur une page HTML classique, 70% du temps d'affichage n'est pas le temps de téléchargement, mais le temps nécessaire à la connexion auquel s'ajoute le temps de latence, la meilleure des optimisations est donc de réduire le nombre de requêtes au serveur.

Il est possible, si vous avez une architecture comprenant plusieurs serveurs, que les dialogues entre ces derniers impactent vos performances. Même si ce cas est très peu probable, il est bon de s'en souvenir si vous ne trouvez pas d'où viennent vos problèmes de performances, notamment si les serveurs sont distants physiquement.

Système de cache http

Les systèmes de cache permettent d'éviter de recharger des éléments déjà téléchargés. Ceci permet d'avoir des allers-retours serveur beaucoup plus rapides ou même de les éviter. Les navigateurs des visiteurs gèrent leur propre cache.

A part quelques options configurables par l'utilisateur, le fonctionnement de ce cache est défini par les paramètres du serveur. L'intérêt est de profiter du cache du navigateur pour éviter de récupérer les fichiers et donc limiter la bande passante utilisée par les navigateurs.

1. **Gestion des Etag**

SYMPTOMES : La bande passante de votre site est saturée, ou le premier chargement des pages est lent pour les visiteurs.

Par défaut, la balise HTTP Etag (entity tag) est activée, elle permet de ne pas télécharger 2 fois les mêmes fichiers. Chaque ressource statique (image, css, js...) possède un identifiant unique pour une date de modification donnée.

Ainsi lors d'un second chargement de la page, le navigateur enverra l'Etag au serveur qui répondra soit par un code 302 Found (trouvé, donc non modifié) soit par un 200 OK (envoi du fichier). Ce système n'évite pas les allers-retours client- serveur, évite de renvoyer le contenu des fichiers, donc évite la consommation de bande passante ainsi que le traitement que doit effectuer le serveur.

Si vous avez une architecture avec plusieurs serveurs vous devez utiliser cette option avec précaution. Dans ce cas, l'Etag est différent pour chaque serveur. Il est possible que cette balise augmente la bande passante plutôt que de la réduire. C'est pourquoi certain outil de mesure vous conseille de la désactiver.

2. **Date d'expiration d'une ressource**

SYMPTOMES : Malgré les Etags en place, le chargement d'une page reste lent. Vous voyez dans Firebug qu'un grand nombre de fichiers sont chargés par page et que le temps d'affichage de votre page est principalement du temps de latence.

Pour éviter le temps de latence, il existe, pour le serveur Apache, le mod_expire. Il ajoute une en-tête HTTP ‹Expires› qui indique la date d'expiration d'une ressource au navigateur. Tant que cette date n'est pas atteinte, le navigateur utilisera directement les données qu'il a en cache. Il n'y a donc plus aucun temps d'attente.

La stratégie la plus efficace est donc de mettre une durée de vie illimitée et de renommer la ressource lorsqu'elle est modifiée.

Exemple d'utilisation : A chaque modification d'un fichier, on en change le nom : style.1.0.css devient style.1.1.css. Il est aussi possible de modifier l'URL d'appel en passant un paramètre fictif : style.css?c=1 devient style.css?c=2.

Ce procédé est très performant après le chargement de la première page qui ne peut bénéficier du cache. De plus, il faut privilégier la réutilisation des ressources dans les différentes pages. Par exemple : Utiliser dans toutes les pages un même et unique fichier CSS.

Réduire le nombre de fichiers

SYMPTOMES : Vous voyez dans Firebug qu'un grand nombre de fichiers sont chargés par page et que le temps d'affichage de votre page est principalement du temps de latence.

1. Limiter le nombre de fichier CSS et JS

Il est possible de regrouper les fichiers CSS ou JS en un seul. Plusieurs outils en ligne offrent ce service.

Cependant, si vous utilisez un framework ou un CMS, vérifiez que celui-ci ne propose pas un mode d'agrégation.

2. Regrouper les images – Sprite

Il s'agit ici de regrouper les images du thème qui sont couramment utilisées en une seule. Cette technique s'appelle "Sprite". L'avantage est de n'utiliser qu'un nombre réduit d'images. En revanche, il faudra, via une feuille de style, afficher l'image en background et la positionner.

Idéalement une page de votre site sera composée de :

- Une feuille de style css
- Une feuille de JavaScript js
- 3 à 6 images pour le thème
- éléments « dynamiques » et spécifiques de la page

Réduire la taille des fichiers

SYMPTOMES : La bande passante de votre site est saturée, ou le premier chargement des pages est lent pour les visiteurs.

1. **Réduire la taille des images**

Il est possible de regrouper les fichiers CSS ou JS en un seul. Plusieurs outils en ligne

Si vous avez des images qui ne font pas partie intégrante du thème, vous devez les réduire au maximum, éviter de charger des images trop grandes (donc d'un poids important). Il est préférable de les réduire ou d'utiliser des mécanismes automatiques de mise à l'échelle. Une image ne devrait jamais dépasser 500ko (sauf exception).

Un outil en ligne permet de compresser un peu plus vos images sans détériorer la qualité. N'hésitez pas à l'utiliser sur vos images avant de les mettre en ligne : http://www.smushit.com/ysmush.it/

Vous trouverez facilement sur le web de nombreux articles traitant ce thème (Gif vs. PNG, PNG8 etc.).

2. **Limiter le poids des fichiers – Compression Gzip**

Les fichiers Html, Css et JavaScript sont des fichiers texte et donc particulièrement adaptés à une compression ZIP importante. Le temps nécessaire à cette décompression est négligeable pour le visiteur. En réduisant les flux envoyés aux utilisateurs, vous réaliserez une économie de bande passante.

Ce gain de bande passante s'effectue au détriment d'une consommation accrue de CPU (la compression utilise de la puissance serveur).

3. **Limiter le poids des fichiers – Minimify**

Une autre possibilité pour limiter le poids de vos fichiers CSS et JavaScript est de les minimiser. La minimisation supprime les retours à la ligne, les tabulations ainsi que les commentaires. La lecture de ces fichiers devient donc quasiment impossible, mais ce n'est pas un problème en production.

La plupart des bibliothèques JavaScript sont disponibles en version "minimisé". Elles sont donc plus rapides à charger. Par exemple, la version 1.4.4 min de jQuery pèse 76.6 Ko tandis que la version normale (de développement) pèse 178 Ko.

Déporter des ressources

SYMPTOMES : Vous avez effectué toutes les modifications précédentes, mais que le temps d'affichage de votre page est toujours long et que l'augmentation de la bande passante de votre serveur est impossible, vous pouvez utiliser une solution multiserveurs.

1. Multi-sites

Déployer son site sur plusieurs noms de domaine (ou sous domaine) permet à l'utilisateur de charger plusieurs fichiers simultanément sur plusieurs serveurs. En effet, le navigateur ouvre un nombre de connexion défini par nom de domaine. Ainsi avec 2 serveurs on ouvre 2 fois plus de connexion.

Le nombre de connexions simultanées ouvertes dépend de votre navigateur. Internet Explorer 6 n'ouvre que 2 connexions simultanées, là où Firefox 3 en ouvre 8. Si votre page contient une vingtaine d'images / fichiers CSS / fichiers Javascript, sous Internet Explorer 6, elles seront réparties sur 2 connexions seulement soit 10 ressources chargées en série et non en parallèle.

En répartissant les ressources sur 2 noms de domaine différents, il est possible que le navigateur ouvre jusqu'à 4 connexions (ou 16 pour FireFox). Donc deux fois plus rapides pour charger la page.

Pour autant, il ne faut pas non plus tomber dans l'excès inverse en possédant un grand nombre de domaine. Tout accès à un nouveau domaine entraine des temps de recherche au DNS (Domain Name Serveur, Serveur de nom de domaine, temps de ‹ recherche › du serveur). Que le serveur soit le même ou non, ce temps sera incompressible.

De plus, le fait de multiplier le nom de domaine peut entrainer une complexité de maintenance. Il est donc important de calibrer le rapport nombre de ressource/nombre de domaine.

2. Externaliser les ressources – Content Delivery Network (CDN)

Si la bande passante est saturée, le CDN permet de stocker une page complète pour la restituer à l'utilisateur le plus rapidement possible sans passer par le serveur web. C'est une sorte de cache déporté donc qui ne charge pas le serveur principal. Deux outils open source se démarquent : Varnish & Squid

Un service plus évolué permet de géolocaliser les destinataires ce système repose sur un mécanisme de routage afin d'utiliser les serveurs ‹ les plus proches › pour lui fournir les données de la page dont il a besoin. Akamai est le leader du marché.

3. **Edge Site Includes – Esi**

Norme maintenue par le consortium W3C, elle permet de « reconstituer » une page web grâce à des balises dans la page qui offrent la possibilité de charger la partie « statique » directement du serveur CDN et la partie « dynamique » du serveur web.

Exemple : Si votre site possède une page où l'utilisateur sera identifié avec son nom, il vous est impossible de la stocker dans le CDN car elle sera différente pour tous les utilisateurs. L'ESI permet d'identifier cette zone générée dynamiquement pour stocker l'ensemble du contenu de la page sauf le nom utilisateur. Ce dernier sera récupéré directement du serveur web par le CDN qui l'agrégera dans la page envoyée au client.

Dernières techniques

SYMPTOMES : Votre serveur répond bien mais vous cherchez à optimiser le temps d'affichage de la page pour augmenter la réactivité perçue du site par vos visiteurs.

1. **Chronologie de chargement**

C'est une chose généralement connue, mais dont les détails et les implications ne sont pas toujours bien mesurés : une page web est constituée de nombreux composants, qui sont chargés les uns après les autres par le navigateur.

Il faut s'assurer que la page peut être consultée de manière confortable, même si le temps de chargement est long. L'idée est d'ordonner le téléchargement des éléments en fonction de leur importance pour l'utilisateur. Par exemple, les bibliothèques JavaScript peuvent être mises en bas de page.

Cette optimisation ne concerne pas directement les performances mais a un vrai impact sur l'impression de vitesse pour l'utilisateur.

2. **Page speed, module Apache**

Google fournit un module Apache nommé Page speed permettant d'accélérer la transaction serveur-client. Cet outil permet d'appliquer un certain nombre de filtres sur les pages HTML, les fichiers CSS et JavaScript, les images PNG et JPEG. Ces filtres permettent d'automatiser de nombreuses optimisations sans avoir à parcourir l'ensemble du code du projet. Les gains en performance peuvent être très importants sur certains sites.

L'outil peut être très consommateur des ressources serveurs (processeur et mémoire). Il est donc préférable de faire très attention en utilisant cette solution.

3. **Précision sur les solutions présentées**

Il existe encore d'autres méthodes pour optimiser la gestion de votre bande passante comme ouvrir des connexions persistantes, utiliser des balises méta HTML pour inclure des ressources directement dans le code HTML...

Mais celles-ci peuvent devenir rapidement compliquées à mettre en place. C'est pourquoi il faudra d'abord privilégier les techniques proposées dans les autres parties avant de les prendre en considération.

3 Rendre plus performant votre Code PHP JavaScript

Stocker les calculs en cache

SYMPTOMES : Votre serveur PHP ou MySQL est saturé. Le processeur est utilisé à 100%, ou dans le cas de la base MySQL, le disque sature.

1. **Envoi sans calcul – cache spécifique**

Si les temps de calcul d'un élément de la page sont importants et que l'élément change peu, il est intéressant de le sauvegarder tel quel pour le restituer rapidement et soulager le processeur.

Par exemple, dans le cas de statistiques simples (nombre d'utilisateurs inscrits, etc.), il peut être intéressant de stocker les statistiques en cache. Dans le cas de graphiques complexes, il peut être intéressant de stocker l'image, générée par le code PHP, pour une durée déterminée plutôt que de ré-effectuer les calculs pour chaque utilisateur.

2. Memcache

Memcache est un système de cache pour stocker les éléments qui concernent la session des utilisateurs, des résultats de requêtes etc. Il permet d'éviter d'utiliser des ressources du serveur afin qu'il réponde plus rapidement. Performant, il peut stocker n'importe quel objet PHP sérialisable, et peut être partagé entre plusieurs serveurs dans le cas d'une architecture avec load-balancing.

Vous pouvez utiliser Memcache pour stocker des résultats de requêtes SQL coûteuses, des statistiques, etc...

Configuration PHP – Apache

SYMPTOMES : Les processus Apache/PHP utilisent la totalité des ressources processeur.

1. APC – Cache PHP

Tout comme Memcache, APC (pour Alternative PHP Cache) est un système de cache, mais à la différence de Memcache, APC est capable de mettre en cache les ‹ opcodes ›. Quand un script PHP est exécuté, il est d'abord compilé en interne par PHP. PHP ne stocke pas le résultat compilé du script et donc, à la prochaine exécution du script, PHP le recompilera.

APC est conçu pour stocker le résultat compilé dans son cache. Ainsi, lorsque PHP exécute le script, APC lui servira directement le script compilé. PHP n'aura pas besoin de recompiler le script et économisera donc beaucoup de temps.

La simple installation d'APC peut diviser la charge des processus Apache/PHP par 2 ou 3. N'hésitez donc pas à l'utiliser ! Vous pouvez également utiliser APC comme un cache classique, mais à la différence de Memcache, APC ne fonctionne que sur un serveur. Dans des environnements complexes, les 2 existeront côte à côte : APC pour mettre en cache les opcodes, et Memcache pour mettre en cache les calculs / requêtes.

2. Autres paramétrages Apache – PHP

SYMPTOMES : Le nombre de connexions simultanés en pic est très important. Les processus Apache saturent, certains clients ne reçoivent pas les pages web quand le trafic est élevé.

Apache permet de configurer le nombre de connexions simultanées qu'il peut ouvrir (c'est-à-dire le nombre de requêtes client auxquelles il peut répondre simultanément). Dans la plupart des installations, vous utiliserez les modules Worker ou Prefork.

Les installations LAMP classiques utilisent par défaut le module Prefork mais il faut savoir que d'autres solutions existent. De réputation, prefork serait plus stable et workers plus scalable.

Quoi qu'il en soit, chaque module MPM (que ce soit Prefork ou Worker) utilisent des paramètres qu'il convient d'examiner car ils ne conviennent pas forcément à des sites ayant une forte fréquentation. Ces paramètres peuvent brider votre application.

Dans le journal Apache, vous verrez facilement si le MaxClients est atteint.

Il existe des alternatives au serveur web Apache. La plus connue est LightHTTPD. Ce serveur web est réputé plus performant. Mais avant de considérer cette option, sachez que la communauté travaillant sous Apache est bien plus vaste, et que les gains en performance ne sont pas toujours sensibles.

Limiter la consommation de Ram/CPU

1. FileHandlers et ResultSets

SYMPTOMES : Vos ressources RAM et CPU peuvent être consommées par l'applicatif (PHP) en temps de calcul, ou par la base de données. Dans le cas ou vous avez diagnostiqué qu'il s'agit d'une surconsommation de ces ressources par le processus Apache (https), il sera nécessaire de passer en revue le code de l'application et sa configuration.

PHP, comme JAVA ou la plupart des langages de programmation récents gèrent automatiquement la libération des ressources qui ne sont plus utilisées (gérée par le garbage collector). Néanmoins, les FileHandlers et les ResultSets doivent impérativement être fermés de façon explicite, car ils ne sont pas pris en compte par le garbage collector.

Ainsi, si votre application effectue des lectures/écriture sur des fichiers, n'oubliez pas que chaque pointeur de fichier créé doit être fermé :

```
$fh = fopen($filename, 'r');
while(!feof($fh)){
    //...
}
fclose($fh);
```

D'autre part, si vous n'utilisez pas de framework ORM qui gère lui- même la récupération des données, veillez à fermer les ResultSets qui ont été ouverts :

```php
$rSet = mysql_query("SELECT * FROM users");
while ($user = mysql_fetch_object($rSet)){
    //...
}
mysql_freeresult($rSet);
```

2. Autoload

SYMPTOMES : La consommation processeur de vos processus PHP est trop élevée.

Cette fonctionnalité permet de trouver une classe si elle n'a pas été identifiée dans le contexte courant. Ainsi, votre code PHP peut charger les classes PHP dont il a besoin « à la volée ».

Par exemple, l'objet $user, de la classe User ne peut être instancié si le fichier User.class.php n'a pas été inclus (« require_once »).

Exemple :

```php
/* Definir une manière de trouver la classe User */
function custom_autoload($className){
    require_once
dirname(__FILE__) . "/classes/" .$className .
".class.php";
}
```

```php
/*
 * Ajouter la fonction custom_autoload à la liste
 * des fonctions permettant de trouver une classe.
 */
spl_autoload_register('custom_autoload');

/* Sans autoload, cette instanciation aurait généré
une Erreur:
 * Fatal error: Class 'User' not found in ...
 */
$user = new User();
```

Cet exemple semble trivial, mais imaginez que vous disposez d'une librairie de classes que vous utilisez partout dans votre application.

Par défaut, 2 options sont envisagées :

- Inclure « à la volée » les classes que vous utilisez à chaque fois, ce qui devient vite fastidieux
- Faire l'inclusion de tous les fichiers au début du script, mais vous incluez alors des fichiers qui ne sont pas utilisés dans tous les scripts, d'où une consommation inutile de RAM et de temps processeur

Utiliser l'autoload est donc fortement recommandé, et cette solution est même plus efficace que celle qui consiste à inclure les fichiers à la volée, car un « require_once » doit vérifier si ce fichier a été déjà inclus ou non

Déporter l'exécution du code

SYMPTOMES : La consommation processeur de vos processus PHP est trop élevée ou le temps de réponse des pages est trop long, même sous une charge normale.

1. **Mettre en place des traitements asynchrones**

La conception la plus intuitive et la plus simple d'un projet est souvent pensée en synchrone. Néanmoins, la mise en place de processus asynchrones permet à la fois de diminuer le temps d'attente de l'utilisateur, et de lisser les pics de charge.

Pensez notamment à externaliser les tâches suivantes :

- Production de rapport, mise à jour de statistiques
- Envois de mails
- Appels de services externes

2. Ajax

L'Ajax (Asynchronous JavaScript And XML) est une manière de dialoguer avec le serveur après le chargement de la page, grâce au JavaScript. L'intérêt est de charger les éléments qui prennent du temps après que le navigateur de l'utilisateur a affiché le contenu de la page Internet.

Si vous avez un calcul long que vous devez faire à chaque fois et que malgré toutes les recommandations que vous avez suivies, vous ne pouvez pas réduire le temps. Il est préférable d'afficher la page sans cet élément puis de récupérer le résultat sur le serveur ensuite.

Toutefois, il faut faire attention car certains moteurs de recherche ne pourront pas indexer le contenu de ce bloc.

Attention, certains frameworks et certains CMS doivent charger un ensemble de modules et d'outils avant de passer à l'exécution du code « effectif ». Dans la mesure du possible, lorsque vous faites de l'Ajax essayez de charger le moins d'éléments possible pour limiter la durée d'exécution et ainsi accélérer le retour d'information.

Si une page effectue plusieurs appels AJAX, sachez que PHP ne pourra traiter la requête suivante qu'une fois que la session aura été libérée (fonction session_write_close). En effet, la session étant par défaut libérée en fin d'exécution du script, les requêtes AJAX pourront alors être traitées simultanément.

Éviter les « fausses optimisations »

LE CONTEXTE : Votre serveur PHP sature. Vous vous tournez vers votre développeur, qui vous explique qu'il a LA solution, il faut absolument changer tous les guillemets doubles par des guillemets simples parce que ceux-ci sont plus rapide.

Méfiez-vous absolument de ces « fausses » optimisations ! Le web regorge de mythes. Comment les reconnaître ? Généralement, il s'agit d'un conseil sur une manière de coder « bas niveau ».

Quelques exemples :

- Les boucles « for » qui décrémentent la variable au lien de l'incrémenter sont plus rapides
- La boucle « for » est plus rapide que la boucle « while »
- Il faut utiliser des guillemets simples au lieu de guillemets doubles
- Le développement objet est plus lent que le développement procédural

Tous ces conseils résultent de débats sans fin entre développeurs de langages de programmation.

Vous trouverez sur Internet des benchmarks à foison comparant les performances entre guillemets simples et guillemets doubles. Et il est vrai que lors d'une boucle qui concatène 100000 fois une chaîne de caractères, les guillemets simples sont 3% plus rapides que les guillemets doubles.

Mais restons focalisé sur l'important ! Si vous lisez ce livre blanc, vous n'êtes pas à la recherche d'une amélioration de 3% de vos performances, mais bien de 100 ou 200%.

Oubliez donc ces fausses optimisations, et concentrez-vous sur de vraies solutions comme l'étude des index ou l'optimisation des flux !

4 Optimiser votre base de données MySQL

Les principales solutions d'optimisation consistent à réduire les coûts de lecture des tables MySQL.

Trois types de solutions sont possibles :

- **Optimiser la gestion des index**
- **Agréger des données**
- **Améliorer les requêtes**

Gestion des index

1. **Indexation de la base de données**

SYMPTOMES : votre application se comporte bien lorsque la base de données est vide. Mais lorsque la base de données se remplit avec des volumétries de production, l'application ralentit.

La base de données consomme la plupart des ressources du système : la consommation CPU ou les accès disques atteignent 100%. Les logs MySQL « slow queries » montrent que certaines requêtes SELECT sont lentes à exécuter.

LA SOLUTION : En termes de base de données, la première solution à examiner est de vérifier que les index sont bien définis.

Mais qu'est-ce qu'un index ? Imaginons un bibliothécaire devant rechercher un livre parmi une collection. Si ces livres sont empilés dans le désordre, il va devoir parcourir sa collection livre par livre pour trouver le livre qui l'intéresse.

De la même manière, dans une base de données, si une table « livre » contient un million d'enregistrement et que l'on cherche un livre par son « nom », le moteur de la base de données devoir parcourir un à un tous les enregistrements. Le bibliothécaire va bien sûr classer les ouvrages.

Une fois classés, il sera capable de retrouver un livre très rapidement, par simple dichotomie. En termes de bases de données, ce classement s'appelle un « index ». Si la colonne « nom » est indexée, la base de données pourra retrouver un ouvrage par son nom sans parcourir l'intégralité de la table.

De la même manière que le bibliothécaire va passer d'une journée de recherche à quelques minutes, la base de données passera de quelques secondes de recherches à quelques millisecondes. Et l'efficacité des index augmente avec la taille de la base !

L'avantage des bases de données par rapport aux classements physiques est de permettre plusieurs tris à la fois. On peut créer un index sur le nom, sur l'auteur, et sur le numéro ISBN à la fois.

COUT D'UN INDEX : Bien sûr, lorsqu'un nouveau livre arrive, il faut l'indexer. Ainsi, si les index permettent de gagner en temps de recherche, ils ralentissent légèrement l'insertion des données. Un index occupe aussi de la place sur le disque dur. Cependant, dans une application web typique, les données sont lues bien plus souvent qu'elles ne sont écrites. De manière quasi systématique, le gain de performance apporté par un index dépasse largement son coût.

ANALYSER LES REQUETES : Lorsque l'on créer un index sur une base de données, il est important de tracer les améliorations que l'index apporte sur le temps d'exécution de la requête. Le plus souvent, l'amélioration est flagrante (on passe de plusieurs secondes à quelques millisecondes).

Si l'amélioration n'est pas flagrante, il faut vérifier que l'index est bien utilisé. Pour cela, on utilisera la commande ANALYZE TABLE de MySQL :
http://dev.mysql.com/doc/refman/5.0/fr/analyze- table.html

La lecture des analyses n'est pas aisée, mais c'est un des meilleurs moyens de comprendre le fonctionnement de la base de données. Dans ce rapport, vous devez vous assurer qu'il n'y a plus de « full-scan » sur les tables. Les full-scans correspondent à un parcours de tous les enregistrements sans utiliser d'index.

Les logs de « slow queries » MySQL devraient mettre en évidence les requêtes les plus lentes. Analysez ces requêtes et vérifiez notamment la partie « WHERE » et « ORDER BY » de la requête. Les colonnes utilisées devraient être indexées. Vérifiez également si votre requête effectue une jointure entre plusieurs tables,

Les clefs étrangères devraient être indexées. L'option log_queries_not_using_indexes permet d'ajouter au log des de « slow queries » toutes les requêtes qui effectuent un « full- scan ». Elles pourront ainsi être rapidement repérées.

Si vous utilisez des tables INNODB, nous vous recommandons vivement de déclarer toutes les clefs étrangères. Une clef étrangère déclarée dans INNODB doit forcément être indexée.

Ainsi, vous ne pourrez pas oublier l'index et vous éviterez un futur problème de performances. Vous pouvez consulter à ce sujet l'article sur les clefs étrangères de la documentation MySQL : http://dev.mysql.com/doc/refman/5.0/fr/example-foreign-keys.html

2. Les index multi-colonnes

Si votre requête SQL à optimiser contient plusieurs colonnes dans la clause WHERE, créer un index sur chacune des colonnes ne sera pas optimal. La base de données utilisera en effet un des index, mais ne peut pas utiliser plusieurs index sur la même table simultanément. La base de données va alors chercher à utiliser l'index le plus efficace (le plus discriminant). Pour ces cas, il est possible d'utiliser des index multi-colonne.

En reprenant l'exemple de la bibliothèque, on pourrait créer un index sur les auteurs, puis les noms d'ouvrage. Les livres seraient d'abord classés par auteurs, puis par nom d'ouvrage. Ainsi, il serait rapide de retrouver un livre dont on connait l'auteur et le nom.

Sachez que dans les index multi-colonnes, l'ordre des colonnes est primordial !

3. Cardinalité des index

Il peut se produire des cas où la base de données dispose de la possibilité d'utiliser plusieurs index pour une même requête et doivent faire le choix entre 2 index à utiliser. Elle prendra alors généralement l'index le plus discriminant, c'est-à-dire l'index qui dispose du plus grand nombre de valeurs différentes pour une même colonne. On parle de « cardinalité » de l'index.

Par exemple, si notre table d'ouvrage possède une colonne « auteur » qui contient le nom de l'auteur, et une colonne « type » qui contient soit « livre », soit « magazine », la cardinalité de l'index « type » sera de 2, alors que la cardinalité de l'index « auteur » sera égale au nombre d'auteurs.

Si une base de données a le choix entre les 2 index, elle prendra l'index « auteur » à chaque fois. La cardinalité est estimée par la base de données au fur et à mesure des opérations. Pour les tables MyISAM, si on effectue un très grand nombre d'opérations de suppression/insertion, la cardinalité d'un index peut être faussée.

La base MySQL prendra alors le mauvais index ce qui peut nuire aux performances. La commande « ANALYZE TABLE » ré effectue le calcul de cardinalité pour corriger ce problème. Attention cependant, cette commande bloque la table le temps de son exécution. Il ne faut donc surtout pas l'exécuter pendant un pic de charge !

4. Recherche de contenus text : index FULLTEXT

Dans le reste de ce paragraphe dédié aux index, nous allons cependant aborder un type particulier d'index : les index FULL-TEXT. Reprenons l'exemple de notre bibliothécaire. Il vient de classer tous les ouvrages par ordre alphabétique. Quelqu'un arrive, et lui demande de retrouver un ouvrage dont il ne se souvient pas complétement du titre. Il se rappelle cependant que le titre finit par « de la méthode ».

Notre bibliothécaire est bien embêté. Son classement alphabétique fonctionne, à condition que l'on connaisse le début de nom du livre, pas la fin. Il ne va pas avoir d'autre choix que de parcourir sa collection de livre entière.

De la même manière, un index « classique » sur la colonne nom ne pourra pas être utilisé sur une requête du type :

```
SELECT * FROM books WHERE name LIKE '%de la méthode'
```

Si ce type de requête vous pose des problèmes de performances, il faudra créer un index FULL-TEXT. Les index FULL-TEXT permettent de rechercher rapidement du texte placé au milieu ou à la fin d'une colonne.

Sous MySQL, il y a cependant des restrictions. Les bases de données InnoDB ne supportent pas les index FULL-TEXT. Il faudra utiliser le moteur MyISAM, qui possède lui-même d'autres limitations.

5. Moteur FULLTEXT spécialisé – LUCENE, Apache, SOLR…

Vous avez appliqué toutes les optimisations ci-dessus, vous avez créé des index de manière optimale, et malgré cela, vos requêtes sont toujours lentes. Dans le cas de notre table « livres », il s'agirait d'un client souhaitant retrouver un livre dont le contenu du livre contienne le mot « collier », mais aussi le mot « girafe ». Le mécanisme devrait être suffisamment souple pour permettre les fautes d'orthographe (par exemple « giraffe » au lieu de « girafe »), et ne retourner que les livres publiés après 2009. Dans ce type d'exemple, il faut envisager de revoir l'architecture de l'application. De toute évidence, MySQL n'est pas bien adapté aux recherches FULL-TEXT complexes, et est incapable de réaliser des requêtes approximatives (fuzzy-searches).

Si vous souhaitez réaliser des recherches de ce type, et si vous avez des problèmes de performance à cause de ces requêtes, il existe des bases de données spécialisées pour la recherche dans une grande quantité de documents « full-text ».

- **Lucene**

Lucene est un moteur de recherche Full-text. A l'origine implémenté en Java, il en existe une implémentation PHP dans le Zend Framework : http://framework.zend.com/manual/fr/zend.search.lucene.html Lucene stocke des « documents » et permet de rechercher en full-text dans le document ou ses métadonnées. Il permet d'effectuer des « fuzzy-searches » et des requêtes complexes.

L'implémentation PHP de Lucene est relativement lente par rapport à son homologue Java. Aussi, Lucene ne devrait pas être utilisé si la volumétrie est trop grande. On se tournera alors vers SOLR.

- **Apache-SOLR**

Serveur basé sur la version Java de Lucene. Il s'interroge via web-services (sur un modèle REST), et permet facilement de stocker les documents puis de les chercher. Si vos problèmes de performances sont dus à une fonctionnalité de recherche FULL-TEXT et que l'implémentation de Lucene dans le Zend Framework ne suffit plus, Apache-SOLR est la solution dont vous avez besoin : http://lucene.apache.org/solr/

Nous avons présenté Lucene et Apache SOLR, qui sont les solutions les plus connues, mais il en existe d'autres.

Dans ce paragraphe, nous n'avons que présenté les possibilités d'optimisation les plus élémentaires. Bien maitriser la gestion des index de la base de données est primordial pour obtenir des performances satisfaisantes mais le sujet est vaste et dépasse le cadre de ce livre blanc. Il faudrait par exemple traiter des différents types d'index (HASH, TREE, etc..). N'hésitez pas à approfondir le sujet avec la documentation MySQL : http://dev.mysql.com/doc/refman/5.0/fr/mysql-indexes.html

Agrégation

La bonne gestion des index à une importance directe sur les performances de votre projet web. Mais il est possible de rencontrer malgré tout des problèmes liés au temps d'exécution de certaines requêtes. C'est notamment le cas des applications qui présentent des statistiques.

Agréger des données permet de faire des requêtes sur les tables moins volumineuses. Le principe est d'appliquer un ensemble de pré-calcul sur la table puis de stocker le résultat dans une nouvelle table. Les données seront prises dans la table agrégée. Le but est de n'avoir que très peu de calcul à faire au moment de l'exécution de la requête afin d'améliorer le temps de réponse.

1. Agrégation en batch

L'une des techniques parmi les plus simples à mettre en œuvre pour agréger des données consiste à créer un script (PHP ou autre) qui s'exécute régulièrement pour effectuer des pré-calculs (batch). Ce script devra être lancé via une tache automatique. Le temps entre 2 exécutions est critique.

En effet, il y a plusieurs réponses pour un même problème :

- Exécuter très souvent le script : Ceci permet d'avoir des données proche du temps réel, et peut être exécuté rapidement, à condition que la base de données s'y prête. Il est possible que la requête la plus simple mette plusieurs minutes, auquel cas il sera impossible de lancer souvent le script sous peine de saturer le serveur.

- Exécuter une fois par jour : Même si le script est long, il ne bloquera pas le serveur. Il est conseillé de l'exécuter au moment où la charge serveur est la plus faible (souvent en pleine nuit).

Avantage : La solution est souvent facile à développer et à déployer, grande liberté dans le code.

Inconvénient : Les données présentées ne sont pas le reflet exact de la production. Selon les volumes, les tests sont parfois long et complexes à réaliser.

2. Agrégation direct (triggers)

Le trigger permet de créer l'agrégation directement en base de données. A chaque nouvel enregistrement dans une table, il est possible de créer un trigger qui effectuera le pré-calcul et l'enregistrera directement dans la table agrégée. Il faut cependant faire attention à la limite des bases de données.

Avantage : Temps réel

Inconvénient : Maintenance, migration de base de données et limites en écriture.

Exemple : A chacun insert dans la table principale, vous ajoutez 3 lignes dans la table d'agrégation. Ainsi au lieu de faire 1 insert, il y en aura 4. La base saturera donc beaucoup plus rapidement s'il y a de nombreuses écritures.

Optimisation des requêtes à la base de données

1. Réduire le nombre de requêtes

SYMPTOMES : Votre application effectue pour chaque page un grand nombre de requête que vous voyez dans les logs MySQL (plusieurs centaines). Les pages sont longues à s'afficher bien que ni le serveur PHP, ni le serveur MySQL ne soient saturés.

Si votre application effectue un trop grand nombre de requêtes, la base de données peut-être la raison des mauvaises performances. En effet, en dehors du temps d'exécution de la requête, il faut compter le temps de latence à la base de données. Or ce temps de latence n'est pas négligeable, car il est parfois plus important que le temps d'exécution de la requête elle-même ! De manière générale, il faut donc préférer une requête importante plutôt que plusieurs petites requêtes afin de factoriser le temps de latence.

Par exemple :

```
$result = mysql_query("SELECT * FROM users");
while ($user = db_fetch_object($result)){
    $countResult = mysql_query("SELECT count(1)
FROM cars
WHERE user_id = $user->id");
    // ...
}
```

Ce script exécutera une première requête pour récupérer tous les utilisateurs, puis, pour chaque utilisateur retourné, une nouvelle requête sera effectuée pour connaître le nombre de voitures qu'il possède.

Une meilleure pratique consiste à lancer une seule requête pour tout récupérer en une seule fois :

```
$result = mysql_query("
SELECT u.*, count(c.id)
FROM users u JOIN cars c ON u.id=c.user_id
GROUP BY u.id");
while ($user = db_fetch_object($result)){
   // ...
}
```

2. Optimiser les requêtes les plus critiques

SYMPTOMES : Votre serveur MySQL est saturé.

Pour optimiser les appels à la base de données, on est souvent tenté de traiter en priorité les requêtes les plus longues à être exécutées. En réalité, si une requête met 10 secondes à s'exécuter, mais qu'elle est peu appelée, l'optimisation n'améliorera pas les performances de l'application. Il faut plutôt s'intéresser aux requêtes qui consomment le plus de temps sur un intervalle de temps représentatif d'une utilisation normale de l'application.

Une solution pour savoir quelles sont les requêtes qui doivent être optimisées est d'identifier l'ensemble des requêtes exécutées sur une période représentative (la mise en place du log des slow queries avec un seuil à 0 permet de les voir toutes), puis d'analyser le résultat en comptant le nombre d'occurrences des différentes requêtes (au paramètre près). En pondérant ensuite le temps d'exécution par le nombre d'appels de chaque requête, vous obtiendrez un classement des requêtes les plus coûteuses.

3. Éviter les requêtes imbriquées

SYMPTOMES : Votre serveur MySQL est saturé.

L'exécution de requêtes imbriquées implique la création d'une table temporaire. Si elle peut être évitée (ce n'est pas toujours le cas), il vaut mieux effectuer des jointures.

Exemple :

```
SELECT *
FROM users
WHERE user.role_id IN (
   SELECT uid FROM users_roles
   WHERE rid IN (
         SELECT rid
FROM role
WHERE name IN ('administrateur', 'client')
   )
)
```

Cette requête utilise deux requêtes imbriquées, alors que l'usage de jointures serait plus performant :

```
SELECT *
FROM users u
   JOIN users_roles ur on ur.user_id = u.id
   JOIN role r on r.id=ur.role_id
WHERE r.name IN ('administrateur', 'client')
```

Dénormalisation du modèle de données

Un bon modèle de données doit être facile à maintenir.

Pour cela, une donnée ne doit être présente qu'à un endroit dans la base. On dit alors que la base est sous sa « forme normale ».

Imaginons une base de données contenant des produits achetés et des utilisateurs. La table liant les produits aux utilisateurs sera sous sa forme normale si elle contient une clef étrangère vers les tables des utilisateurs et des produits :

Utilisateurs	
Id	Name
1	John
2	Bob

Achats	
User_id	Product_id
1	2
2	1

Produits	
Id	Name
1	TV
2	Radio

Si on souhaite obtenir la liste des achats, on effectuera une requête avec probablement deux jointures pour récupérer le nom des utilisateurs et des produits.

La forme dénormalisée serait :

Utilisateurs	
Id	Name
1	John
2	Bob

Achats	
User	Product
John	Radio
Bob	TV

Produits	
Id	Name
1	TV
2	Radio

Cette forme sera plus efficace à requêter, puisque la base de données n'aura pas de jointures à faire.

En revanche, sous cette forme, la base de données est beaucoup plus dure à maintenir. Si un produit change de nom par exemple, il faudra modifier son nom dans tous les champs dénormalisés de la base, ce qui est complexe, et aussi long !

Il vaut mieux toujours partir d'un modèle normalisé et ne dénormaliser que lorsque les problèmes de performance sont constatés. En effet, la dénormalisation complexifie la maintenance du code et devrait être évitée si possible.

La dénormalisation sera par contre la règle pour accéder à des techniques plus avancées comme les bases de données multidimensionnelles. Dans ce cas, on devra en effet commencer à construire une vue entièrement dénormalisée de la base avant de remplir la base de données multidimensionnelle.

D'autres bases de données comme les bases de données NoSQL reposent sur des modèles moins stricts et moins normés (donc plus dénormalisés). Nous n'aborderons cependant pas le sujet dans ce livre blanc, le sujet méritant à lui seul son propre livre !

Base de données multidimensionnelles

L'agrégation de données dans des tables d'agrégats permet d'offrir des statistiques rapidement. Cependant, cette approche peut être limitée si les statistiques à produire sont très diverses.

Prenons l'exemple classique suivant : vous disposez de plusieurs magasins dans le monde. Chacune des ventes est loguée dans une table qui contient le produit vendu, le prix, le vendeur, la date, le magasin avec sa localisation géographique. A partir de cette table, vous créez une table d'agrégat pour connaître le nombre de ventes par vendeur.

Puis vous souhaitez créer un nouvel indicateur (donc une nouvelle table d'agrégat) pour connaître les ventes par magasin. Deux mois plus tard, nouvel indicateur : les ventes par magasin, par catégorie de produit et par mois.... Au bout de quelques mois, votre base de données est encombrée de tables d'agrégats difficiles à maintenir.

Pour ces cas ou vous ne pouvez savoir à priori quelles sont les statistiques que vous souhaitez produire, les bases de données multidimensionnelles sont la solution. Une base de données multidimensionnelle (ou base OLAP) permet d'analyser un grand nombre de données sous des axes différents. Dans le cas de notre magasin, la base de données multidimensionnelle effectuera l'intégralité des agrégats possibles en une passe (ce que l'on appelle *calculer le cube multidimensionnel*).

On pourra alors interroger la base instantanément sous toutes ses dimensions (c'est-à-dire sous tous les axes d'agrégation possibles). Si vous êtes habitués de Excel, vous pouvez voir une table multidimensionnelle comme un tableau croisé dynamique pouvant contenir des milliards de lignes.

Maintenir une base multidimensionnelle demande cependant des efforts. Il faut mettre en place un script de chargement de la base à partir de votre base relationnelle (MySQL).

En résumé, utilisez une base de données multidimensionnelle si :

- Vous avez des problèmes de performance sur un affichage de statistiques
- Ces problèmes de statistiques sont dus à la volumétrie de vos données
- Vous ne savez pas à priori quelles statistiques seront nécessaires ou si la diversité des statistiques empêche l'utilisation de tables d'agrégats

Si vous décidez d'utiliser une base de données multidimensionnelle, vous devrez choisir la base à utiliser. Citons les plus connues :

- Mondrian : une base de données multidimensionnelle open- source
- Microsoft Analysis Services: la base de données multidimensionnelle de Microsoft
- Essbase : la base de données multidimensionnelle de Oracle
- SAS OLAP Server: le serveur de base de données multidimensionnelle fournit par la suite SAS d'analyse BI

Configuration MySQL

Votre site web n'effectue plus que des requêtes indexées. Vous avez pré calculé toutes vos statistiques. Bref, vous avez épuisé toutes les optimisations recommandées dans ce document, et vous avez encore un problème de performances.

Alors (et seulement à ce moment), vous pouvez regarder les paramètres d'optimisation de MySQL. MySQL est une base proposant plusieurs moteurs de gestion des tables en son sein, les plus connus étant MyISAM et InnoDB. Chacun de ces moteurs possède des performances et des paramètres d'optimisation qui lui sont propres.

1. MyIsam vs InnoDB

Le choix entre MyISAM et InnoDB n'est pas anodin. La plupart du temps, le choix est dicté par les fonctionnalités de la base. Par exemple, seul InnoDB permet de déclarer des contraintes d'intégrité sur la base de données. InnoDB dispose aussi d'un journal, qui garantit que les données seront toujours dans un état stable, même si le serveur de base de données s'arrête en plein milieu d'une opération d'écriture.

A contrario, MyISAM est le seul à proposer un index full-text. Si vous avez besoin de cette fonctionnalité, le choix seront donc imposé directement.

Vous avez bien sûr la possibilité de mélanger les moteurs, chaque table pouvant utiliser un moteur qui lui est propre. Cette solution est à utiliser avec précaution. En effet, chaque moteur dispose de ces propres restrictions, notamment en termes de backup. En utilisant les deux moteurs, vous cumulez les deux limitations.

En termes de performance, chaque moteur possède ses caractéristiques propres. Si vous avez choisi MyISAM ou InnoDB pour l'une des fonctionnalités qui lui est propre, mieux vaut rester à votre choix initial que décider de changer pour un problème de performances. Si par contre, votre choix a été effectué « parce que c'est le moteur par défaut », vous pouvez alors légitimement vous poser la question d'un changement de moteur.

InnoDB semble être le moteur préféré de la plupart des développeurs. D'un point de vue des performances uniquement, il se différencie sur deux grands points : il dispose d'un journal, et il supporte le « *row-level locking* ».

Pour une base de données, un journal est un espace dans lequel la base va écrire toutes les actions d'écriture effectuées. Dans la pratique, cela signifie que les actions sont écrites 2 fois. Bien sûr, d'un point de vue des performances, cela a un coût puisque les données sont écrit en double.

Cependant, c'est la seule manière de permettre de gérer des transactions (d'effectuer des « rollbacks »), et surtout, cela garanti de toujours pouvoir récupérer la base de données dans un état stable, quelques soit les coupures du serveur.

Le fait que MyISAM ne dispose pas de journal ne le rend cependant pas systématiquement plus performant. En effet, lors d'une opération d'écriture en base avec MyISAM, MyISAM bloque toute la table.

A contrario, InnoDB ne bloque que les lignes (le fameux *row-level locking*).

Ainsi, InnoDB est plus efficace que MyISAM pour les applications à haute concurrence d'accès (typiquement une application web). InnoDB possède cependant des limitations qu'il est bon de savoir.La plus connue étant que la récupération du compte de lignes dans une table effectue un parcours de table complet (très lent).

Il faut donc éviter les requêtes du type :

```
SELECT COUNT(*) FROM books
```

Enfin, InnoDB est activement développé alors que MyISAM est plus stable. On peut donc s'attendre à voir de nombreuses améliorations de performance arriver sur InnoDB.

InnoDB étant un système plus complet que MyISAM, il dispose également de plus de paramètres. Alors que MyISAM est assez facile à configurer, InnoDB demandera plus de temps pour obtenir un système optimal.

La liste de tous les paramètres InnoDB est présente sur cette page :

http://dev.mysql.com/doc/refman/5.0/en/innodb-parameters.html

Même si cette liste est longue, si vous souhaitez faire du fine-tuning MySQL, nous vous conseillons de la parcourir en entier, certains paramètres ayant des impacts sur d'autres. Nous ne parcourrons pas l'intégralité des paramètres dans ce livre blanc (ce n'est pas le sujet), mais allons présenter les principaux.

2. Taille du cache InnoDB

Le paramètre principal influençant les performances MySQL sera bien sûr la taille du cache. Plus le cache est grand, moins MySQL devra aller chercher de données sur le disque lors de requêtes, et donc plus la base sera rapide.

Sous InnoDB, la taille du cache est gérée par le paramètre innodb_buffer_pool_size. Par défaut, il est initialisé à 128 Mo. Sur un serveur de base de données dédié, on peut monter jusqu'à 80% de la RAM du serveur.

Attention cependant, MySQL aura besoin de RAM également pour stocker le programme, la structure de la base, etc... De plus, si vous utilisez MyISAM en parallèle de InnoDB, il faudra également laisser de la place en RAM pour le cache MyISAM !

3. Nombre de connexions simultanées

Le nombre de connexions simultanées à la base est limité par le paramètre max_connections. Si le nombre de connexions maximal est atteint, la base de données retournera une erreur (Too many connections).

Il est donc très important que ce paramètre ne soit pas trop bas. Chaque thread PHP pouvant ouvrir une connexion à la base de données, le nombre de connexions maximum demandées à la base de données devrait être environ égal au nombre de threads PHP autorisés dans Apache (voir à ce sujet la section Autres paramétrages Apache- PHP)

La commande ‹ SHOW STATUS › MySQL permet de voir le nombre maximal de connections simultanées ouvertes sur le serveur, ce qui permet de voir en production si on se rapproche ou non du maximum autorisé.

Attention, ne modifiez pas le paramètre max_connections sans modifier le paramètre table_cache qui lui est associé.

table_cache représente le nombre de fichiers ouverts simultanément par MySQL. En MyISAM, pour chaque connexion, lorsque MySQL accède à une table, MySQL ouvre un fichier. Si MySQL fait une requête avec un join sur 2 tables, il ouvre 2 fichiers, etc...

Dans l'idéal, on devrait donc avoir table_cache = max_connections * nombre de join maximum. Dans le cas de InnoDB, ce nombre peut être différent puisque selon la configuration de InnoDB, celui-ci écrit toutes les tables dans le même fichier, ou écrit une table par fichier (suivant la valeur du paramètre innodb_file_per_table).

4. Nombre de threads

SYMPTOMES : Vous disposez d'un serveur très puissant (plus de 8 cœurs). MySQL ne sature pas le CPU, mais celui-ci plafonne à 20% ou 40% de la puissance maximum. Les autres ressources (disque / RAM) ne saturent pas non plus. Le serveur semble ne pas donner tout son potentiel.

Par défaut, MySQL n'ouvre pas plus de 8 threads pour InnoDB. Le paramètre régissant ce comportement est innodb_thread_concurrency :

http://dev.mysql.com/doc/refman/5.0/en/innodb-parameters.html#sysvar_innodb_thread_concurrency

Si vous disposez d'un serveur dédié puissant (par exemple avec 24 processeurs), vous pourriez vouloir adapter ce paramètre. La recommandation MySQL est de configurer ce paramètre à « deux fois le nombre de processeurs + le nombre de disques ».

Nous vous recommandons cependant d'effectuer des tests avec des valeurs très différentes pour voir le comportement de votre application. En effet, en augmentant le nombre de threads, vous augmentez la concurrence et donc le risque de locks.

5. Log file

SYMPTOMES : Le disque dur de votre serveur MySQL sature en écriture à cause d'un grand nombre de commandes INSERT / UPDATE / DELETE.

Lorsqu'un utilise InnoDB, MySQL écrit un journal. Comme expliqué précédemment, l'idée du journal est d'écrire l'ancien état de la base avant de l'écraser avec le nouvel état. En cas de crash, on peut ainsi toujours revenir en arrière.

L'écriture sur le disque est effectuée en double. Ceci est inhérent à toutes les bases de données respectant les normes ACID (c'est à dire quasiment toutes les bases SQL sauf MySQL en mode MyISAM).

Il existe dans MySQL des paramètres permettant d'assouplir l'écriture dans le journal (par exemple de faire l'écriture une fois par seconde plutôt qu'à chaque UPDATE ou INSERT).

Le paramètre innodb_flush_log_at_trx_commit régit ce fonctionnement :

http://dev.mysql.com/doc/refman/5.1/en/innodb-parameters.html#sysvar_innodb_flush_log_at_trx_commit
http://www.mysqlperformanceblog.com/2007/11/01/innodb- performance-optimization-basics/

5 Rationaliser votre infrastructure

Lorsque la mise en place d'une des solutions précédentes est coûteuse, il est intéressant d'examiner l'architecture physique de la solution.

Les solutions associées à l'infrastructure matérielle sont nombreuses et la plupart consistent à augmenter le nombre de serveurs. Cette partie présente quels sont les moyens d'améliorer les performances de votre application grâce à l'augmentation des capacités matérielles.

Upgrade

Le principe de l'upgrade des serveurs est simple. Il s'agit d'augmenter de façon constante les performances physiques de sa machine : rajouter de la RAM, augmenter la taille des disques ou ajouter des disques, augmenter la puissance du ou des processeurs, utiliser un système multi-cœurs ...

Ces actions permettent d'augmenter les capacités du serveur pour accueillir un plus grand nombre de connexions utilisateurs et/ou de calculs serveur. Une fois *upgradé*, le serveur web sera capable d'absorber une plus grande charge.

BON A SAVOIR : augmenter la puissance d'un serveur coûte relativement peu tant qu'on reste dans une puissance « standard », mais le rapport coût/performance croît exponentiellement dès que l'on sort de la norme.

Il est préférable de rapidement s'orienter vers des solutions de clustering qui fournissent une architecture plus robuste notamment en cas de panne. Aussi, il est intéressant de définir un seuil à partir duquel il convient de stopper l'upgrade, pour démarrer une stratégie de clustering.

Clustering

1. Séparer Apache-MySQL

Pour beaucoup d'applications PHP- MySQL, l'accès aux données représente un point majeur de l'utilisation des ressources serveurs (goulet d'étranglement). Or, dans le même temps, l'application a un besoin non négligeable de ressources pour faire fonctionner l'application PHP. Dans de nombreux cas, ces deux demandes de ressources simultanées peuvent créer des lenteurs, voire des défaillances.

Une solution pour résoudre ce problème est de séparer le serveur Apache, contenant l'application PHP, du serveur MYSQL contenant le Système de Gestion de Base de Données (SGBD).

Ainsi, les actions effectuées sur l'un des serveurs n'impacteront pas l'autre et les performances en seront améliorées. Le SGBD étant dans de nombreux cas le goulet d'étranglement d'un site web, il sera ici traité de façon dédiée par un unique serveur qui n'aura que cette charge à gérer.

BON A SAVOIR : Dans ce cas, la latence augmente. L'application ne doit pas effectuer de trop nombreuses requêtes par page.

2. Load Balancing des serveurs APACHE

Le load balancing Apache permet de distribuer le trafic sur plusieurs serveurs. L'objectif est de multiplier le nombre de serveurs capables d'héberger le site cible et ainsi de partager la charge relative aux actions utilisateurs sur plusieurs serveurs.

La distribution sera effectuée par un serveur dédié (le *load-balancer*). Pour les sites à très fort trafic, on aura recours à un boitier physique (proposé par les hébergeurs) afin de rediriger les accès au site web sur les différents serveurs.

Un paramétrage fin est également nécessaire pour partager les données en cache, ainsi que les données sessions (APC, Memcached). La base de données doit également être partagée pour être accessible des différents serveurs web.

En plus d'améliorer les performances (plusieurs serveurs répondent en même temps aux requêtes utilisateurs), il y a une autre raison pour mettre en place cette optimisation : la solution est beaucoup plus robuste.

En effet, comme plusieurs serveurs fonctionnent en parallèle, si l'un d'entre eux est défaillant, les autres sont susceptibles de prendre le relais instantanément.

En revanche, cela demande un suivi constant de l'activité des serveurs en utilisant Nagios par exemple. Dans le cas contraire, un serveur peut devenir indisponible sans toutefois alerter l'administrateur du site.

NOTE : la maintenance de la solution devient plus complexe du fait que les modifications effectuées sur un serveur doivent être diffusées vers tous les autres.

3. SQL Master/Slave

Dans la plupart des cas d'utilisation d'un serveur MYSQL, les requêtes effectuées sont des requêtes de lecture, de récupération de listes de produits, d'informations personnelles, ...

Ces demandes fréquentes surchargent les serveurs MYSQL qui doivent gérer par ailleurs les requêtes d'insertion (INSERT) et de mise à jour (UPDATE). Une nouvelle solution d'optimisation d'architecture matérielle est de dédier différents serveurs MYSQL à des actions précises.

Un serveur maître gère alors toutes les requêtes d'INSERT et d'UPDATE, et un serveur esclave gère toutes les requêtes de sélection. Ainsi la charge du SGBD sera partagée entre plusieurs serveurs. Pour conserver une cohérence de données, le serveur esclave est répliqué du serveur maître en temps réel.

BON A SAVOIR : Une solution qui n'améliore pas les performances mais qui augmente la robustesse de la solution est le SQL Clustering. Le principe est de mettre à disposition un nouveau serveur SQL si le principal n'est plus disponible (crash, maintenance, ...).

Cette architecture est basée sur une réplication automatique des données du Système de Gestion de Base de Données. Le serveur courant est alors utilisé pour gérer l'ensemble des actions du serveur web, pendant que le serveur de backup ne fait que recopier les actions effectuées sur le principal. L'avantage est que ce système peut être couplé à une utilisation des instances master/slave.

Cloud Computing

Pour simplifier, le ‹Cloud Computing› peut être défini comme le moyen d'exécuter des tâches de calcul en parallèle sur différents serveur. Ce qui est interessant dans le cadre de l'amélioration des performances, est la façon d'utiliser cette puissance de calcul dynamiquement, en fonction de la charge serveur.

Il est très fréquent qu'un site Internet soit soumis à des pics d'activité ; de ce fait, il n'est pas forcément nécessaire de posséder une structure matérielle maximale pendant les périodes d'activité faible, ou moindre, de votre site Internet.

Le principe évolutif du ‹Cloud Computing› comme le propose Amazon (AWS – EC2), est de mettre à disposition de l'application une force de calcul supplémentaire (image de votre site web) quand vous en avez besoin.

Ce besoin très spécifique peut être défini de différentes façons : l'utilisation du CPU du serveur, la charge mémoire du serveur ou toutes autres données relative à l'utilisation et la charge du serveur web. Toutes ces données peuvent être surveillées afin de déclencher l'utilisation de nouvelles ressources matérielles. Ainsi, votre infrastructure s'adaptera automatiquement à votre besoin.

BON A SAVOIR : Ce mouvement s'accompagne des bases de données NoSQL. Ces bases de données sont moins simples à requêter mais elles se dupliquent facilement. En fait, ces besoins sont tirés par les géants du Web (Google, Amazon ...) qui ont besoin de rechercher des informations qui ne sont pas nécessairement structurées.

SUIVEZ-NOUS SUR LINKEDIN

Retrouvez notre **actualité**, nos **articles,**
nos **livres blancs**, nos **innovations**
et bien plus encore !

+33 (0)1 85 08 34 98
communication@thecodingmachine.com

56 rue de Londres
75008 Paris

TheCodingMachine
TCM://