Configuration d'Apache pour des performances maximales

Gazette Linux n°123 — Février 2006

Vishnu Ram V

Article paru dans le n°123 de la Gazette Linux de février 2006.
Traduction française par Deny `<deny CHEZ monaco POINT net>`**.**
Relecture de la traduction française par Joëlle Cornavin `<jcornavi CHEZ club TIRET internet POINT fr>`**.**

Table des matières

1. Introduction

Apache, implémentation *open source* du serveur HTTP, est le serveur *web* le plus populaire sur l'Internet. L'enquête sur les serveurs *web* de décembre 2005 menée par Netcraft (http://news.netcraft.com/archives/web_server_survey.html) 1 montre qu'environ 70 % des sites sur Internet utilisent Apache.

Les performances du serveur Apache peuvent être améliorées en ajoutant des ressources matérielles complémentaires telles que de la mémoire vive, un processeur plus rapide, etc. Mais la plupart du temps, on peut obtenir le même résultat avec une configuration personnalisée du serveur. Cet article explique comment obtenir les performances maximales d'Apache avec les ressources matérielles existantes, en particulier sur des systèmes Linux. Naturellement, on part du principe qu'il y a assez de ressources matérielles — notamment assez de mémoire vive pour éviter que le serveur ne « swappe » fréquemment. Les deux premières sections présentent les diverses options de configuration `compile-time` et `run-time`. La section run-time suppose qu'Apache est compilé avec le MPM (*Multi-Processing Module*) prefork (http://httpd.apache.org/docs/2.2/mod/prefork.html). La compression HTTP et la mise en cache (*caching*) est abordée plus loin. Pour finir, nous verrons l'utilisation de serveurs séparés pour la prise en charge de contenus statiques et dynamiques est abordée. Des connaissances de base sur la compilation et la configuration d'Apache et de Linux sont requises.

2. Options de configuration `compile-time`

2.1. Chargement des seuls modules requis :

Le serveur Apache HTTP est un programme modularisé dans lequel l'administrateur peut choisir les fonctions à inclure dans le serveur en sélectionnant un ensemble de modules 2. Les modules peuvent être compilés soit statiquement en tant qu'éléments du binaire HTTP, soit en tant qu'objets partagés dynamiques (DSO, *Dynamic Shared Objects*). Les modules DSO peuvent être compilés lors de la construction du serveur ou ajoutés ultérieurement via l'utilitaire apxs (http://httpd.apache.org/docs/2.2/programs/apxs.html), ce qui permet une compilation à une date ultérieure. Le module mod_so doit être compilé statiquement dans le noyau d'Apache pour activer la prise en charge des DSO.

Exécutez Apache avec les seuls modules requis. Vous réduisez ainsi l'utilisation de la mémoire, ce qui améliore les performances du serveur. Les modules compilés statiquement économiseront la mémoire vive employée pour la prise en charge des modules chargés dynamiquement, mais vous devrez recompiler Apache pour ajouter ou supprimer un module. C'est ici que le mécanisme DSO prend tout son sens. Une fois que le module mod_so est compilé statiquement, n'importe quel autre module peut être ajouté ou supprimé à l'aide de la commande **LoadModule** dans le fichier `httpd.conf`. Naturellement, vous devrez compiler les modules à l'aide d'apxs s'ils ne l'ont pas été lors de la construction du serveur.

2.2. Choix des MPM appropriés

Le serveur Apache est livré avec une sélection de modules MPM qui sont chargés de lier les ports réseau sur la machine, en acceptant des requêtes et en répartissant les processus enfants pour traiter les requêtes 3. Un seul MPM peut être chargé à tout moment dans le serveur.

Le choix d'un MPM dépend de divers facteurs, selon que le système d'exploitation prend en charge ou non les fils d'exécution, la quantité de mémoire disponible, l'extensibilité par rapport à la stabilité, selon que des modules tiers avec des fils d'exécution non synchronisés sont utilisés ou non, etc.

Les systèmes Linux peuvent choisir d'employer un MPM avec fil d'exécution comme worker (http://httpd.apache.org/docs/2.2/mod/worker.html) ou un MPM sans fil d'exécution comme prefork (http://httpd.apache.org/docs/2.2/mod/prefork.html) :

- Le MPM worker emploie de multiples processus enfants. Il est multiprocessus à l'intérieur de chaque processus enfant, et chaque fil gère une seule connexion. Worker est rapide, hautement extensible et comparativement peu exigeant en mémoire. Il convient parfaitement pour des processeurs multiples. Par ailleurs, worker est moins tolérant aux modules défectueux, et un module défectueux peut affecter tous les fils d'un processus enfant.
- Le MPM prefork emploie de multiples processus enfants, dont chacun gère une connexion à la fois. Prefork est bien adapté aux systèmes à un ou deux processeurs, la rapidité est comparable à celle de worker et il est hautement tolérant aux modules défectueux et aux « plantages » de processus enfants — mais il consomme beaucoup de mémoire. De plus, davantage de trafic entraîne une plus grande utilisation de mémoire.

3. Options de configuration `run-time`

3.1. Recherche de DNS

La directive `HostnameLookups` permet la recherche des DNS de façon à journaliser des noms d'hôtes plutôt que des adresses IP. Cette option ajoute un délai d'attente à chaque requête, puisque la recherche de DNS doit être réalisée avant que la requête soit terminée. `HostnameLookups` est positionnée sur `Off` par défaut dans Apache 1.3 et supérieur. Laissez-le à `Off` et utilisez un programme de post-traitement tel que logresolve (http://httpd.apache.org/docs/2.2/programs/logresolve.html) pour résoudre les adresses IP dans les fichiers journaux `access` d'Apache. Logresolve est livré avec Apache.

Lorsque vous faites appel aux directives `Allow from` ou `Deny from`, employez une adresse IP au lieu d'un nom de domaine ou d'un nom d'hôte. Autrement, une double recherche de DNS est effectuée pour s'assurer que le nom de domaine ou le nom d'hôte n'est pas usurpé.

3.2. `AllowOverride`

Si `AllowOverride` n'est pas positionné sur `None`, alors Apache tentera d'ouvrir le fichier `.htaccess` (comme spécifié par la directive `AccessFileName`) dans chaque répertoire qu'il visite. Par exemple :

Si une requête est effectuée pour l'URI `./index.html`, alors Apache tentera d'ouvrir `.htaccess`, `/var/.htaccess`, `/var/www/.htaccess` et `/var/www/html/.htaccess`. Ces consultations supplémentaires du système de fichiers allongent le délai d'attente. Si `.htaccess` est exigé pour un répertoire particulier, alors adaptez-le pour ce seul répertoire.

3.3. `FollowSymLinks` et `SymLinksIfOwnerMatch`

Si l'option `FollowSymLinks` est cochée, alors le serveur suivra les liens symboliques dans ce répertoire. Si `SymLinksIfOwnerMatch` est cochée, alors le serveur ne suivra les liens symboliques que si le fichier ou le répertoire cible appartient au même utilisateur que le lien.

Si `SymLinksIfOwnerMatch` est cochée, alors Apache devra émettre des appels système supplémentaires pour vérifier si l'appartenance du lien correspond et du fichier cible correspondent. Des appels système supplémentaires sont également necessaires quand `FollowSymLinks` n'est PAS cochée. Par exemple :

Pour une requête effectuée pour l'URI `/index.html`, Apache exécutera `lstat()` sur `/var`, `/var/www`, `/var/www/html` et `/var/www/html/index.html`. Ces appels système supplémentaires allongeront le délai d'attente. Les résultats de **lstat** n'étant pas mis en cache, ils apparaitront lors de chaque requête.

Pour obtenir des performances maximales, cochez `FollowSymLinks` partout et ne cochez jamais `SymLinksIfOwnerMatch`. Ou sinon, si l'option `SymLinksIfOwnerMatch` est exigée pour un répertoire, alors cochez-la pour ce seul répertoire.

3.4. Négociation de contenu

Évitez la négociation de contenu pour obtenir une réponse rapide. Si la négociation de contenu est requise pour le site, utilisez des fichiers `type-map` plutôt que la directive `Options MultiViews`. Avec `MultiViews`, Apache doit balayer le répertoire pour rechercher les fichiers, ce qui allonge le délai d'attente.

3.5. `MaxClients`

La directive `MaxClients` fixe la limite maximale de requêtes simultanées que le serveur peut prendre en charge ; aucun processus enfant au-delà de ce nombre n'est engendré. Il ne devrait pas être défini à une valeur trop basse, sinon un nombre toujours croissant de connexions sont reportées dans la file d'attente et occasionnent finalement un dépassement du temps imparti, alors que les ressources du serveur restent inutilisées. Si vous lui donnez une valeur trop élevée, en revanche, le serveur commencera à « swapper », ce qui fera diminuer considérablement le temps de réponse. La valeur appropriée pour `MaxClients` peut être calculée ainsi :

4 `MaxClients` = Mémoire vive totale dédiée au serveur *web* / Taille maximale des processus enfants.

La taille des processus enfants destinés à prendre en charge des fichiers statiques est d'environ 2 à 3 Mo. Pour du contenu dynamique tel que PHP, elle peut être aux environs de 15 Mo. La colonne RSS dans **ps -ylC httpd --sort:rss** affiche l'utilisation de la mémoire physique non permutée par des processus Apache en kilooctets.

S'il y a plus d'utilisateurs simultanés que `MaxClients`, les requêtes seront mises en file d'attente jusqu'à un nombre défini en fonction de la directive ListenBacklog. Augmentez `ServerLimit` pour régler `MaxClients` au-dessus de 256.

3.6. `MinSpareServers`, `MaxSpareServers` et `StartServers`

`MaxSpareServers` et `MinSpareServers` déterminent combien de processus enfants doivent rester actifs tout en attendant des requêtes. Si `MinSpareServers` est réglé trop bas et qu'un grand nombre de demandes survient, Apache devra engendrer des processus enfants supplémentaires pour satisfaire les requêtes. Créer des processus enfants est relativement exigeant en mémoire. Si le serveur est occupé à créer des processus enfants, il ne sera pas en mesure de prendre en charge les requêtes des clients immédiatement. `MaxSpareServers` ne devrait pas être réglé trop haut : trop de processus enfants consommeront des ressources inutilement.

Affinez `MinSpareServers` et `MaxSpareServers` de façon à ce qu'Apache n'ait pas à engendrer plus de 4 processus enfants par seconde (Apache peut engendrer un maximum de 32 processus enfants par seconde). Si plus de 4 processus enfants par seconde sont engendrés, un message est journalisé dans `ErrorLog`.

La directive `StartServers` fixe le nombre de processus enfants que le serveur crée au démarrage. Apache continuera à créer des processus enfants jusqu'à ce que le paramètre `MinSpareServers` soit atteint. Cela n'aura pas grand effet sur les performances si le serveur n'est pas redémarré fréquemment. S'il y a beaucoup de requêtes et qu'Apache est souvent redémarré, donnez à cette directive une valeur relativement élevée.

3.7. `MaxRequestsPerChild`

La directive `MaxRequestsPerChild` fixe la limite du nombre de requêtes que gérera un seul processus enfant du serveur. Après les requêtes `MaxRequestsPerChild`, le processus enfant mourra mais comme il est fixé à 0 par défaut, le processus enfant n'expirera jamais. Il est approprié de donner à cette directive une valeur de quelques milliers. Cela peut aider à empêcher des pertes de mémoire, puisque le processus meurt après avoir satisfait un certain nombre de requêtes. Ne fixez pas une valeur trop basse, car la création de nouveaux processus est exigeante en temps système.

3.8. `KeepAlive` et `KeepAliveTimeout`

La directive `KeepAlive` autorise l'envoi de requêtes multiples sur la même connexion TCP. C'est particulièrement utile pour la prise en charge de pages HTML comportant de nombreuses images. Si `KeepAlive` est positionnée sur `Off`, alors une connexion TCP séparée doit être créée pour chacune des images. La surcharge causée par l'établissement des connexions TCP peut être éliminée en positionnant `KeepAlive` sur `On`.

`KeepAliveTimeout` détermine la durée d'attente de la prochaine requête. Donnez-lui une valeur basse, peut-être entre deux et cinq secondes. Si la valeur fixée est trop élevèe, les processus enfants sont immobilisés, attendant le client alors qu'ils peuvent être utilisés pour satisfaire de nouveaux clients.

4. Compression HTTP et mise en cache

La totalité des spécifications de la compression HTTP figure dans HTTP/1.1. Le serveur utilise la méthode d'encodage gzip ou deflate pour la charge utile de réponse avant qu'elle ne soit envoyée au client. Le client décompresse les données utiles. Il n'y a aucun besoin d'installer un logiciel supplémentaire côté client puisque tous les navigateurs majeurs prennent en charge ces méthodes. Utiliser la compression économisera de la bande passante et améliorera le temps de réponse. Des études ont relevé un gain moyen de 75,2 % grâce à la compression 5.

La compression HTTP peut être activée dans Apache à l'aide du module mod_deflate (http://httpd.apache.org/docs/2.2/mod/mod_deflate.html). La charge utile n'est compressée que si le navigateur demande la compression, sinon le contenu décompressé est pris en compte. Un navigateur prenant en charge la compression informe le serveur qu'il préfère du contenu compressé via l'en-tête de requête HTTP `Accept-Encoding: gzip,deflate`. Alors le serveur répond avec la charge utile compressée et l'en-tête de réponse positionnés sur `Content-Encoding: gzip`.

L'exemple suivant utilise telnet pour afficher la requête et les en-têtes de réponse :

Lors de la mise en cache, une copie des données est stockée chez le client ou sur un serveur mandataire, de sorte que le serveur n'a pas à la rechercher fréquemment. Cela économisera de la bande passante, diminuera la charge sur le serveur et réduira le temps d'attente. Le contrôle du cache est effectué à l'aide d'en-têtes HTTP : Apache permet de le faire par le biais des modules mod_expires (http://httpd.apache.org/docs/2.2/mod/mod_expires.html) et mod_headers (http://httpd.apache.org/docs/2.2/mod/mod_headers.html). On peut aussi utiliser la mise en cache côté serveur, dans laquelle le contenu le plus souvent affiché est stocké en mémoire, ce qui permet sa prise en charge rapide. Le module mod_cache (http://httpd.apache.org/docs/2.2/mod/mod_cache.html) peut servir à la mise en cache côté serveur ; il est stable en production dans la version 2.2. d'Apache.

5. Serveur séparé pour du contenu statique et dynamique

Les processus d'Apache prenant en charge du contenu dynamique consomment de 3 Mo à 20 Mo de mémoire vive. La taille croît pour s'adapter au contenu pris en charge et elle ne décroît jamais jusqu'à ce que le processus meure. À titre d'exemple, supposons qu'un processus d'Apache croisse jusqu'à 20 Mo pendant le chargement de contenu dynamique. Une fois cette requête achevée, il est libre de satisfaire n'importe laquelle autre. Si une requête pour une image survient, alors ce processus de 20 Mo prend en charge du contenu statique, ce qu'un processus de 1 Mo pourrait tout aussi bien faire. Par conséquent, la mémoire est employée de façon inefficace.

Utilisez un Apache léger (avec un minimum de modules statiquement compilés) comme serveur frontal pour prendre en charge du contenu statique. Les requêtes concernant le contenu dynamique devraient être transférées au serveur Apache puissant (compilé avec tous les modules requis). Utiliser un serveur frontal léger offre l'avantage d'une prise en charge rapide du contenu statique sans pour autant consommer trop de mémoire, et seul le contenu dynamique est transféré vers le serveur complet.

Le transfert de requête peut être réalisé à l'aide des modules mod_proxy (http://httpd.apache.org/docs/2.2/mod/mod_proxy.html) et mod_rewrite (http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html). Supposez qu'il y ait un serveur Apache léger qui écoute le port 80 et un Apache complet qui écoute le port 8088. Alors la configuration suivante dans le serveur Apache léger peut être employée pour transférer toutes les requêtes (sauf les requêtes pour les images) sur le serveur complet 9 :

Toutes les requêtes, sauf celles pour des images, seront transférées via le serveur mandataire au serveur dorsal. La réponse est reçue par le serveur frontal et fournie au client. En ce qui concerne le client, toutes les réponses semblent provenir d'un seul serveur.

6. Conclusion

Configurer Apache pour des performances maximales est complexe ; il n'y pas de règles ardues et rapides. Cela dépend beaucoup de la compréhension des exigences du serveur *web* et de l'expérimentation des diverses options disponibles. Faites appel à des outils comme ab (http://httpd.apache.org/docs/2.2/programs/ab.html) et httperf (http://www.hpl.hp.com/research/linux/httperf/) pour mesurer les performances du serveur *web*. Des serveurs légers tels que tux (http://www.redhat.com/docs/manuals/tux/) ou thttpd (http://www.acme.com/software/thttpd/) peuvent aussi être utilisés comme serveur frontal. Si un serveur de base de données est employé, assurez-vous qu'il est optimisé de façon à ne pas créer de goulots d'étranglement. Dans le cas de MySQL, mtop (http://sourceforge.net/projects/mtop) permet de surveiller les requêtes lentes. Les performances des *scripts* PHP peuvent être ameliorées par le biais d'un produit de cache PHP tel que Turck MMCache (http://sourceforge.net/projects/turck-mmcache/). Il élimine la surcharge due à la compilation en mettant en cache les *scripts* PHP dans un état compilé.

7. Bibliographie

1. http://news.netcraft.com/archives/web_server_survey.html
2. http://httpd.apache.org/docs/2.2/dso.html
3. http://httpd.apache.org/docs/2.2/mpm.html
4. http://modperlbook.org/html/ch11_01.html
5. http://www.speedupyoursite.com/18/18-2t.html
6. http://www.xs4all.nl/~thomas/apachecon/PerformanceTuning.html
7. http://www.onlamp.com/pub/a/onlamp/2004/02/05/lamp_tuning.html
8. http://httpd.apache.org/docs/2.2/misc/perf-tuning.html
9. Administration Linux à 200 % (http://www.oreilly.fr/catalogue/2841772950.html), par Rob Flickenger

Je suis titulaire d'un *MTech.* (maîtrise) en *Communication Systems* (systèmes de communication) de l'*IIT Madras* (Institut Indien de Technologie de Madras). J'ai rejoint Poornam Info Vision Pvt Ltd. en 2003 et travaille pour Poornam depuis lors.

Mes centres d'intérêt sont l'optimisation de performances, la surveillance de serveurs et la sécurité. Pendant mon temps libre, je pratique le karaté, je lis et j'écoute de la musique.