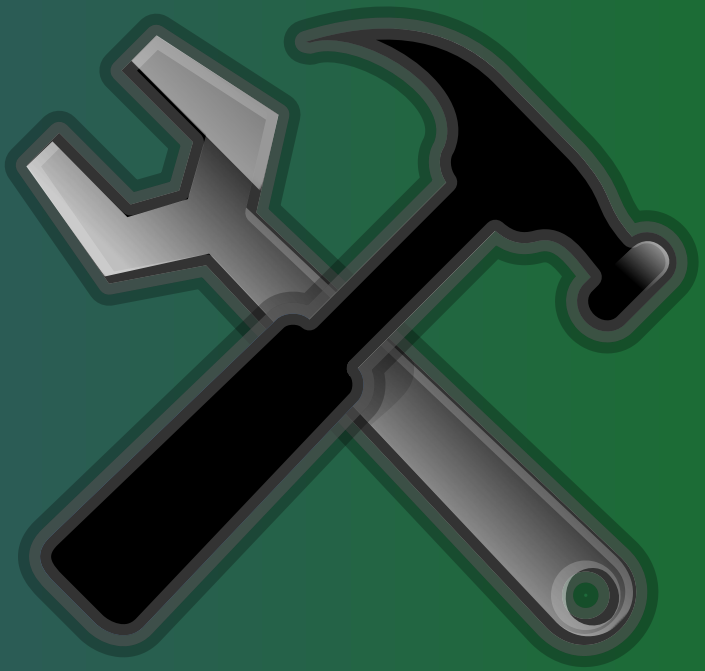
Configuration du module multi-processus (MPM) d'Apache 2.4 event

capable de gérer simultanément 2500 connexions

Mise à jour du 8 mai 2023



Sommaire

1 Introduction

La configuration ci-dessous est optimisée pour un serveur Apache 2.4 qui a une forte charge, que cette charge soit légitime ou liée à du déni de service. Elle permet de gérer simultanément 2500 connexions.

Les autres paramètres de configuration d'un serveur Apache sont décrites sur **https://lafibre.info/ubuntu/**

2 Valeurs proposées pour les directives pour le MPM event

2.1 StartServers 9 (valeur par défaut : 3)

La directive **StartServers** permet de définir le nombre de processus enfants du serveur créés au démarrage. Elle influence uniquement le nombre de slots « idle » au démarrage. La valeur par défaut est de 2 dans la configuration proposée par Ubuntu server et de 3 dans la documentation Apache.

Règle pour déterminer **StartServers** : arrondi supérieur de (MinSpareThreads / ThreadsPerChild) + (MaxRequestWorkers / (40 x ThreadsPerChild))

2.2 ServerLimit 100 (valeur par défaut : 16)

La directive **ServerLimit** définit la limite supérieure du nombre de processus.

Règle pour déterminer **ServerLimit** : MaxRequestWorkers / ThreadsPerChild

Note : Pour modifier **ServerLimit** un restart est nécessaire (donc coupure des connexions en cours). Si vous souhaitez avoir la possibilité d'augmenter la directive **MaxRequestWorkers** sans faire de « restart » d'Apache, je conseille donc de prendre une marge pour définir **ServerLimit** : 1,5 x (MaxRequestWorkers / ThreadsPerChild). Cela permet, par la suite, d'augmenter de 50 % **ThreadsPerChild** avec un simple « reload ».

2.3 MinSpareThreads 145 (valeur par défaut : 75)

La directive **MinSpareThreads** définit le nombre minimum de threads inactifs, pour être en mesure de traiter les pics de requêtes. La valeur par défaut de **MinSpareThreads** est de 25 dans la configuration proposée par Ubuntu server et de 75 dans la documentation Apache.

Règle pour déterminer **MinSpareThreads** : 20 + (MaxRequestWorkers / 20)

2.4 MaxSpareThreads 435 (valeur par défaut : 250)

La directive **MaxSpareThreads** définit le nombre maximum de threads inactifs. Si le serveur possède trop de threads inactifs, des processus enfants seront arrêtés jusqu'à ce que le nombre de threads inactifs repasse en dessous de cette limite. Objectif : libérer de la mémoire vive inutilisée après un pic de trafic. La valeur par défaut de **MaxSpareThreads** est de 75 dans la configuration proposée par Ubuntu server et de 250 dans la documentation Apache.

Règle pour déterminer **MaxSpareThreads** : 3 x MinSpareThreads

2.5 ThreadLimit 25 (valeur par défaut : 64)

La directive **ThreadLimit** permet de définir le nombre maximum que l'on peut affecter à la directive **ThreadsPerChild** pour la durée de vie du processus Apache httpd.

Règle pour déterminer **ThreadsPerChild** : 25, quel que soit **MaxRequestWorkers**, si on ne souhaite pas faire évoluer **ThreadsPerChild** (Si ThreadLimit > ThreadsPerChild, de la mémoire partagée supplémentaire sera inutilement allouée).

Note : Pour modifier la directive **ThreadLimit**, un « restart » est nécessaire (donc coupure des connexions en cours). Si vous souhaitez avoir la possibilité d'augmenter **ThreadsPerChild** sans faire de « restart » d'Apache, je conseille donc de prendre une marge pour définir **ThreadLimit**, comme définir **ThreadLimit** à 64. Cela permet, par la suite, d'augmenter de 50 % **ThreadsPerChild** avec un simple « reload ».

2.6 ThreadsPerChild 25 (valeur par défaut : 25)

La directive **ThreadsPerChild** permet de définir le nombre de threads que va créer chaque processus enfant. Un processus enfant crée ces threads au démarrage et n'en crée plus d'autres par la suite. **ThreadsPerChild** influence le nombre maximum de slots ET le nombre minimal de slots. Si la valeur est supérieure à 40, j'ai expérimenté des absences de réponse à certaines requêtes, quand il y a de nombreuses connexions asynchrones. Je recommande donc de rester sur la valeur par défaut : 25.

Règle pour déterminer **ThreadsPerChild** : 25 (valeur par défaut), quel que soit MaxRequestWorkers.

2.7 MaxRequestWorkers 2500 (valeur par défaut : 400)

La directive **MaxRequestWorkers** permet de fixer le nombre maximum de requêtes pouvant être traitées simultanément. Si la limite **MaxRequestWorkers** est atteinte, toute tentative de connexion sera normalement mise dans une file d'attente, et ceci jusqu'à un certain nombre dépendant de la directive **ListenBacklog**. Lorsqu'un processus enfant se libérera suite à la fin du traitement d'une requête, la connexion en attente pourra être traitée à son tour.

La valeur par défaut de **MaxRequestWorkers** est de 150 dans la configuration proposée par Ubuntu server et de 400 dans la documentation Apache (directive **ServerLimit** multiplié par la valeur de la directive **ThreadsPerChild**).

Ici, dans la configuration proposée, on fixe à 2500 le nombre de requêtes actives pouvant être traitées simultanément – **MaxRequestWorkers**, afin de pouvoir faire face à une forte charge ou un déni de service. La directive **MaxRequestWorkers** se détermine en fonction de la charge attendu et de la mémoire disponible sur le serveur. Il est possible de modifier **MaxRequestWorkers** par un simple « reload » d'Apache, dans la limite du paramètre **ServerLimit**.

2.8 AsyncRequestWorkerFactor 2 (valeur par défaut : 2)

Le MPM event gère certaines connexions de manière asynchrone ; dans ce cas, les threads traitant la requête sont alloués selon les besoins et pour de courtes périodes. Dans les autres cas, un thread est réservé par connexion. Ceci peut conduire à des situations où tous les threads sont saturés et où aucun thread n'est capable d'effectuer de nouvelles tâches pour les connexions asynchrones établies.

Pour minimiser les effets de ce problème, le MPM event utilise deux méthodes :

- Il limite le nombre de connexions simultanées par thread en fonction du nombre de processus inactifs ;
- Si tous les processus sont occupés, il ferme des connexions permanentes, même si la limite de durée de la connexion n'a pas été atteinte. Ceci autorise les clients concernés à se reconnecter à un autre processus possédant encore des threads disponibles.

La directive **AsyncRequestWorkerFactor** permet de personnaliser finement la limite du nombre de connexions par thread. Un processus n'acceptera de nouvelles connexions que si le nombre actuel de connexions (sans compter les connexions à l'état « closing ») est inférieur à : ThreadsPerChild + (AsyncRequestWorkerFactor * nombre de threads inactifs).

Habituellement, **AsyncRequestWorkerFactor** doit varier entre 1.5 et 3 en fonction du type de clients du site.

Règle que j'utilise pour déterminer **AsyncRequestWorkerFactor** : fixé à 2 (valeur par défaut). Si vous modifiez la valeur par défaut, vous devrez effectuer des tests approfondis à forte charge.

2.9 GracefulShutdownTimeout 3600 (valeur par défaut : 0)

La directive **GracefulShutdownTimeout** permet de spécifier le temps, en secondes, pendant lequel le serveur va continuer à fonctionner après avoir reçu un signal « graceful-stop » (« Arrêt en douceur »), afin de terminer le traitement des connexions en cours. C'est « Arrêt en douceur » est réalisé chaque nuit pour la rotation des logs.

Définir la directive **GracefulShutdownTimeout** à zéro, la valeur par défaut, signifie au serveur d'attendre jusqu'à ce que toutes les requêtes en cours aient été traitées. Le serveur va attendre jusqu'à ce que toutes les requêtes en cours aient été traitées, cela peut être très long en cas de requêtes bloquées par un « slow DDOS ». Cela peut entraîner des problèmes pour l'analyse des logs, la rotation ne pouvant s'effectuer.

Positionner le **GracefulShutdownTimeout** à une heure (3600 secondes) permet de réduire certaines attaques « slow DDOS » sans impacter les clients : une heure après la demande de rotation des logs, les requêtes encore actives seront coupées. Réduire le **GracefulShutdownTimeout** en dessous d'une heure peut poser problème pour des clients avec du bas débit qui téléchargent des gros fichiers.

Note pour donner un ordre d'idée : une connexion a 400 Kbit/s (50 Ko/s) est en mesure de télécharger 180 Mo en une heure.

2.10 MaxConnectionsPerChild 0

La directive **MaxConnectionsPerChild** permet de définir le nombre maximum de connexions qu'un processus enfant va pouvoir traiter au cours de son fonctionnement. Lorsqu'il a traité **MaxConnectionsPerChild** connexions, le processus enfant est arrêté. Si **MaxConnectionsPerChild** est définie à 0, il n'y a plus aucune limite sur le nombre de connexions que le processus pourra traiter.

Définir **MaxConnectionsPerChild** à une valeur non nulle limite la quantité de mémoire qu'un processus peut consommer à cause de fuites (accidentelles) de mémoire.

Règle pour déterminer **MaxConnectionsPerChild** : fixé à 0, sauf problème de fuite mémoire.

3 Questions fréquentes :

3.1 Nombre de slots total (busy + idle + free)

C'est la valeur la plus basse entre :

- **MaxRequestWorkers** (150 par défaut, 2500 ici) ;
- **ThreadsPerChild** (25 par défaut et ici) * **ServerLimit** (16 par défaut, 100 ici).

3.2 Nombre de slots idle au démarrage à vide

C'est la valeur la plus haute entre :

- **MinSpareThreads** (25 par défaut, 145 ici) arrondi supérieur, pour être un multiple entier de **ThreadsPerChild** ;
- **ThreadsPerChild** (25 par défaut et ici) * **StartServers** (3 par défaut, 9 ici).

3.3 Nombre minimal de slots idle quand de nombreux slots sont « busy »

C'est **MinSpareThreads** (25 par défaut, 145 ici) avec un arrondi supérieur pour que « busy » + « idle » soient un multiple entier de **ThreadsPerChild** (25 par défaut et ici). C'est donc 150 dans la configuration proposée ici.

3.4 Nombre maximal de slots idle quand de nombreux slots sont « busy »

C'est **MaxSpareThreads** (75 par défaut, 435 ici) avec un arrondi supérieur pour que « busy » + « idle » soient un multiple entier de **ThreadsPerChild** (25 par défaut et ici). C'est donc 450 dans la configuration proposée ici.

4 Exemple de fichier « mpm_event.conf »

Voici les commandes à passer (avec droits root) pour modifier la configuration proposée par défaut par Ubuntu avec le MPM Event et gérer simultanément 2500 connexions : (pour un copier-coller, les commandes sont disponibles **ici**)

1) sed -i -e "s/StartServers\t\t\t 2/StartServers\t\t\t 9\n\tServerLimit\t\t\t 100/g" /etc/apache2/mods-available/mpm_event.conf
2) sed -i -e "s/MinSpareThreads\t\t 25/MinSpareThreads\t\t 145/g" /etc/apache2/mods-available/mpm_event.conf
3) sed -i -e "s/MaxSpareThreads\t\t 75/MaxSpareThreads\t\t 435/g" /etc/apache2/mods-available/mpm_event.conf
4) sed -i -e "s/ThreadLimit\t\t\t 64/ThreadLimit\t\t\t 25/g" /etc/apache2/mods-available/mpm_event.conf
5) sed -i -e "s/MaxRequestWorkers\t 150/MaxRequestWorkers\t 2500\n\ tAsyncRequestWorkerFactor\t 2\n\tGracefulShutdownTimeout\t\t 3600/g" /etc/apache2/mods-available/mpm_event.conf

Ne pas oublier de redémarrer Apache pour la prise en compte : **systemctl restart apache2**

Tutoriel mis à jour par Vivien Guéant le 8 mai 2023.

Autres tutoriels pour Ubuntu server : **https://lafibre.info/ubuntu/**

Certaines phrases sont issues de la documentation Apache :
https://httpd.apache.org/docs/2.4/fr/mod/event.html