−Table des matières

Rapide configuration de PostgreSQL

Cet article, écrit par Guillaume Lelarge, a été publié dans le hors-série 44 du magazine GNU/Linux Magazine France, hors-série dédié à PostgreSQL. Il est disponible maintenant sous licence Creative Commons.

PostgreSQL et son fichier de configuration de 17 Ko sur 503 lignes. Rien de moins que 180 et quelques paramètres. Cela n'aide clairement pas un débutant à se lancer. Pourtant, il faut savoir que seule une grosse dizaine de paramètres sont essentiels à configurer. Le reste n'a pour cible que les cas très particuliers.

Paramètres de connexion

Par défaut, PostgreSQL n'écoute que sur l'interface locale. Donc personne ne peut se connecter sur le serveur PostgreSQL à partir d'une autre machine. Si vous avez besoin d'autoriser l'accès externe (ce qui est généralement le cas), il vous faudra indiquer les interfaces (adresses IP) sur lesquelles PostgreSQL doit attendre les connexions. Pour cela, vous devez modifier le paramètre listen_addresses. Généralement, ce paramètre est passé à * pour indiquer toutes les interfaces. Mais si vous voulez n'autoriser qu'une seule interface, c'est tout à fait possible.

Si vous voulez autoriser les connexions SSL, il vous faut activer le paramètre du même nom. Attention, cela ne veut pas dire ensuite que toutes les connexions seront en SSL. Pour forcer l'utilisation du SSL, vous devrez tout d'abord le configurer dans le fichier des authentifications (dont le nom est pg_hba.conf). Enfin, dernier paramètre de cette catégorie : max_connections. PostgreSQL bloque le nombre maximum de connexions simultanées grâce à ce paramètre. Par défaut, il est à 100, ce qui est une valeur généralement correcte. Il ne faut pas hésiter à l'augmenter, tout en sachant que des valeurs supérieures à 1000 ne sont pas conseillées. Il vaudra mieux utiliser un pooler de connexions dans ce cas. Les deux disponibles pour PostgreSQL vous seront présentés dans le reste de ce hors-série.

Paramètres mémoire

Un serveur de bases de données doit ses performances principalement à sa gestion de la mémoire. En effet, le but est de fournir des données et le meilleur moyen de les donner rapidement, c'est de les avoir déjà en mémoire.

Il faut considérer avec PostgreSQL deux types de mémoire : la mémoire partagée par les processus serveur et la mémoire individuelle de chaque processus.

La mémoire partagée concerne principalement le cache disque : celui des fichiers de données et celui des journaux de transactions. La taille du premier dépend du paramètre shared_buffers, et la taille du second concerne le paramètre wal_buffers. Pour un serveur dédié, un quart de la mémoire est un bon départ pour le paramètre shared_buffers. Pour les autres, il faudra voir la mémoire utilisée par les autres processus pour en déduire ce qui est laissé à PostgreSQL. Si vous l'augmentez (ce qui est conseillé), vous devrez aussi modifier le paramètre noyau shmmax (voir le chapitre « Configuration du noyau »). wal_buffers est beaucoup plus simple à configurer. Pour une instance qui sera fortement utilisée par de nombreux utilisateurs, il est intéressant de l'augmenter à une valeur généralement comprise entre 1 et 8 Mo.

La mémoire individuelle est utilisée dans deux contextes : les tris et hachages, et les opérations de maintenance. Pour les opérations de tris et de hachages, le paramètre work_mem permet d'indiquer la quantité de mémoire utilisable par chaque processus. Si jamais le processus a besoin de plus, les données déjà triées/hachées sont enregistrées sur disque avant de ré-utiliser la mémoire déjà attribuée. Augmenter cette valeur permet donc d'éviter l'utilisation du disque mais il ne faut pas l'augmenter trop car chaque client peut l'utiliser. La valeur est donc à mettre en adéquation avec le nombre de clients maximum possibles (paramètre max_connections) et avec la complexité des requêtes (si une même requête demande plusieurs opérations de tris et/ou de hachage, la mémoire réclamée sera d'autant plus importante). Quant aux opérations de maintenance, maintenance_work_mem permet d'augmenter la mémoire disponible. Les opérations en question sont le VACUUM, la création d'index et la création de clés étrangères. Dans ce cas, on peut mettre des valeurs plus importantes que pour le work_mem. La principale raison est que ces opérations ont généralement besoin de beaucoup plus de mémoire que les tris et les hachages. La seconde raison est que le nombre d'utilisateurs qui exécuteront des opérations de maintenance au même instant est tellement limité qu'il est difficile de croire que plus de deux personnes le feront en même temps.

Journaux de transactions

Pour des raisons de performances, il est souvent intéressant d'augmenter le nombre des journaux de transactions. En effet, PostgreSQL ne déclenche les écritures sur les fichiers de données (une opération appelée CHECKPOINT) qu'à partir du moment où un certain délai est écoulé ou quand un certain nombre de journaux sont écrits. Un paramètre permet de configurer cela: checkpoint_segments. Il vaut 3 par défaut, et il est généralement raisonnable de l'augmenter au moins à 10.

Du coup, comme nous repoussons le moment du CHECKPOINT, donc de l'écriture dans les fichiers de données, la quantité de données à écrire sera beaucoup plus importante. Du coup, pour éviter un pic d'écriture au moment du CHECKPOINT, nous allons demander à diluer les écritures dans le temps jusqu'au prochain CHECKPOINT. Disons qu'il faut 4 minutes entre deux CHECKPOINT, plutôt que d'écrire les x Mo de données en quelques secondes, bloquant en gros le reste de l'activité, nous allons faire les écritures de façon à ce que ces x Mo soient écrits en un peu moins de 4 minutes. Le paramètre pour cela est checkpoint_completion_target et nous indiquons le pourcentage du temps d'écriture entre deux CHECKPOINT. Il est par défaut à 50% (soit une valeur de 0,5). Nous augmentons généralement la durée pour l'écriture à 90% (donc 0,9).

Planificateur

Deux paramètres sont forcément à considérer pour une meilleure planification des plans de requêtes.

effective_cache_size est la taille du cache disque du système d'exploitation. Cela permet au planificateur d'estimer la probabilité pour qu'une table et un index soient en cache. Plus la valeur de ce paramètre est important, plus les parcours d'index sont valorisés.

random_page_cost est le coût d'accès à une page aléatoire sur disque. Pour les disques récents et rapides, diminuer cette valeur jusqu'à 2 peut apporter un plus très conséquent car là-aussi cela favorisera les parcours d'index.

Journalisation

Même s'il ne s'agit pas de performances, une bonne journalisation des traces est essentielle pour savoir ce qu'il se passe sur le serveur. Il n'y a que deux paramètres essentiels. log_destination doit être placé à syslog pour en tirer toutes les possibilités. La rotation des journaux de traces sera réalisée avec un outil comme logrotate. lc_messages doit être configuré à C pour s'assurer que les messages seront en anglais. Ceci est nécessaire pour trouver facilement, via un moteur de recherche sur Internet, des informations sur les messages enregistrées dans les traces. C'est aussi nécessaire quand on veut de l'aide des développeurs.

Récapitulatif

Le tableau Tab.1 présente les quelques paramètres discutés ci-dessus d'une façon plus directe.

Tab.1 : les paramètres essentiels à configurer

Type	Paramètre	Nouvelle valeur	Commentaires
Connexions	listen_addresses	'*'	autorise les connexions par toutes les interfaces réseau du serveur
	ssl	'on'	autorise l'utilisation de connexions chiffrées avec SSL
	max_connections	100	ne jamais dépasser 1000
Mémoire partagée	shared_buffers	¼ de la RAM	pour un serveur dédié
	wal_buffers	8MB	
Mémoire par processus	work_mem	10MB	ne pas dépasser 100 Mo
	maintenance_work_mem	au maximum 1GB	pour un serveur dédié ayant au moins 4 Go
Journaux de transactions	checkpoint_segments	10	
	checkpoint_completion_target	0.9	
Planificateur	effective_cache_size	2/3 de la RAM	pour un serveur dédié
	random_page_cost	entre 2 et 4	2 pour les disques rapides, 4 pour les disques lents
Traces	log_destination	'syslog'	
	lc_messages	'C'	

Configuration du noyau

La mémoire partagée utilisée par PostgreSQL est une partie de la mémoire assez restreinte par défaut sous Linux. Cependant, elle est configurable grâce au paramètre shmmax. Le mieux est d'utiliser l'outil sysctl pour cela.

Il faut ajouter la ligne suivante pour passer la limite à environ 512 Mo:

```
kernel.shmmax = 540000000
```

Pour que la modification soit pris en considération immédiatement, utilisez la commande suivante:

```
debian1:~$ sysctl -p
```

Comment vérifier l'intérêt de cette nouvelle configuration

Deux types de tests sont possibles: soit utiliser pgbench, soit utiliser votre applicatif.

Installons pgbench. Ce dernier fait partie des modules contrib de PostgreSQL. Il convient donc d'installer le paquet des modules contrib relatifs à votre version:

```
debian1:~$ aptitude install postgresql-contrib-8.4
```

pgbench dispose de deux modes d'exécution: le mode initialisation et le mode tests. Le premier sert à remplir une base de données, le second à exécuter un scénario qui donnera lieu à un petit rapport indiquant le nombre de transactions exécutées par seconde.

La première chose à faire est de créer une base:

```
postgres@debian1:~$ createdb benchs
Puis d'exécuter pgbench pour qu'il y crée ses tables et pour qu'il les remplisse:
postgres@debian1:~$ /usr/lib/postgresql/8.4/bin/pgbench -i -s 10 benchs
NOTICE:  la table « pgbench_branches » n'existe pas, poursuite du traitement
NOTICE:  la table « pgbench_tellers » n'existe pas, poursuite du traitement
NOTICE:  la table « pgbench_accounts » n'existe pas, poursuite du traitement
NOTICE:  la table « pgbench_history » n'existe pas, poursuite du traitement
```

```
creating tables...
10000 tuples done.
20000 tuples done.
[… suite du remplissage de la base …]
990000 tuples done.
1000000 tuples done.
set primary key...
NOTICE:  ALTER TABLE / ADD PRIMARY KEY créera un index implicite « pgbench_branches_pkey » pour la table « pgbench_branches »
NOTICE:  ALTER TABLE / ADD PRIMARY KEY créera un index implicite « pgbench_tellers_pkey » pour la table « pgbench_tellers »
NOTICE:  ALTER TABLE / ADD PRIMARY KEY créera un index implicite « pgbench_accounts_pkey » pour la table « pgbench_accounts »
vacuum...done.
```

L'option -s permet d'indiquer un facteur d'échelle pour la taille de la base. Pour avoir une idée approximative de la taille finale, il faut multiplier le facteur d'échelle par 16 Mo. Dans ce cas, nous devons avoir une base d'environ 160 Mo.

La base benchs comprend maintenant quatre tables et fait un total de 152 Mo, soit plus de 5 fois plus que la taille du cache disque de PostgreSQL (shared_buffers est à sa valeur par défaut pour ce test, soit 28 Mo sur une 8.4 sous Linux).

Maintenant, il est possible de l'exécuter en mode tests:

```
postgres@debian1:~$ /usr/lib/postgresql/8.4/bin/pgbench -T 60 -c 8 -S
starting vacuum...end.
transaction type: SELECT only
scaling factor: 10
query mode: simple
number of clients: 8
duration: 60 s
number of transactions actually processed: 77881
tps = 1296.451439 (including connections establishing)
tps = 1299.336099 (excluding connections establishing)
```

L'option -T n'apparaît qu'avec la version 8.4. Pour les versions antérieures de pgbench, utilisez l'option -t en indiquant le nombre de transactions à exécuter par chaque client. L'option -c indique justement le nombre de clients simulés.

Maintenant que nous avons un chiffre pour la configuration par défaut, nous pouvons modifier la configuration, relancer PostgreSQL et relancer pgbench pour voir si les modifications ont apporté une amélioration.

Évidemment, cet outil ne reflétera pas le nombre de transactions par seconde dont pourra bénéficier votre application. Néanmoins, il fournit une tendance entre deux versions de la configuration de PostgreSQL.

Je ne saurais que trop vous conseiller de lire les différents articles de Greg Smith sur pgbench. Il est très facile de mal utiliser cet outil et de se retrouver avec des résultats inintéressants, voire inexploitables. Voici quelques liens pour vous y aider:

- Determining the right pgbench database size scale (http://www.westnet.com/~gsmith/content/postgresql/pgbench-scaling.htm)
- Graphing pgbench results (http://www.westnet.com/~gsmith/content/postgresql/pgbench.htm)
- pgbench-tools (http://www.westnet.com/~gsmith/content/postgresql/pgbench-tools.htm)

Si vous pouvez utiliser votre applicatif pour les tests de performances, vous aurez une meilleure idée du comportement de PostgreSQL. Le mieux dans ce cas est d'utiliser pgfouine. Cet outil PHP a été écrit par Guillaume Smet, de la société OpenWide. Il est disponible maintenant en version 1.1 avec de belles améliorations, notamment dans la gestion de la mémoire.

Commençons donc par configurer PostgreSQL pour qu'il trace toutes les requêtes exécutées. Pour cela, il suffit d'initialiser le paramètre log_min_duration_statement à 0. Ceci fait, il faut demander le rechargement de la configuration à PostgreSQL:

```
debian1:~$ /etc/init.d/postgresql reload
```

Ensuite, vous devez lancer vos tests sur votre applicatif. Pendant ce temps, vous pouvez installer le paquet Debian (toujours en version 1.0 actuellement):

```
debian1:~# aptitude install pgfouine
```

Ceci fait, les traces PostgreSQL contiendront toutes les requêtes exécutées avec leur durée d'exécution. Il faut récupérer ce journal applicatif et le fournir à pgfouine:

```
debian1:~# pgfouine -file requetes.log -top 40 \
  -report queries.html=overall,bytype,slowest,n-mosttime,n-mostfrequent,n-slowestaverage \
  -report hourly.html=overall,hourly \
  -report errors.html=overall,n-mostfrequenterrors \
  -format html-with-graphs \
  -logtype syslog \
  -memorylimit 512
```

Cette commande créera trois fichiers:

- queries.html, une liste des requêtes les plus lentes, les plus fréquentes, etc;
- hourly.html, un décompte des requêtes par heure et par type de requêtes (lecture, écriture);
- errors.html, une liste des requêtes erronées.

Ces rapports indiquent notamment le temps d'exécution totale des requêtes. Après modification de la configuration, prise en compte par PostgreSQL et ré-exécution des tests, il est intéressant de comparer les chiffres de durée d'exécution totale.

Conclusion

Ce très court article n'est là que pour indiquer les quelques paramètres essentiels à une bonne configuration d'un serveur PostgreSQL. Il sera toujours possible dans un second temps d'affiner le paramétrage, notamment pour les paramètres non présentés dans cet article.