Configuration de la base de données

Enterprise applications

NetApp
May 09, 2024

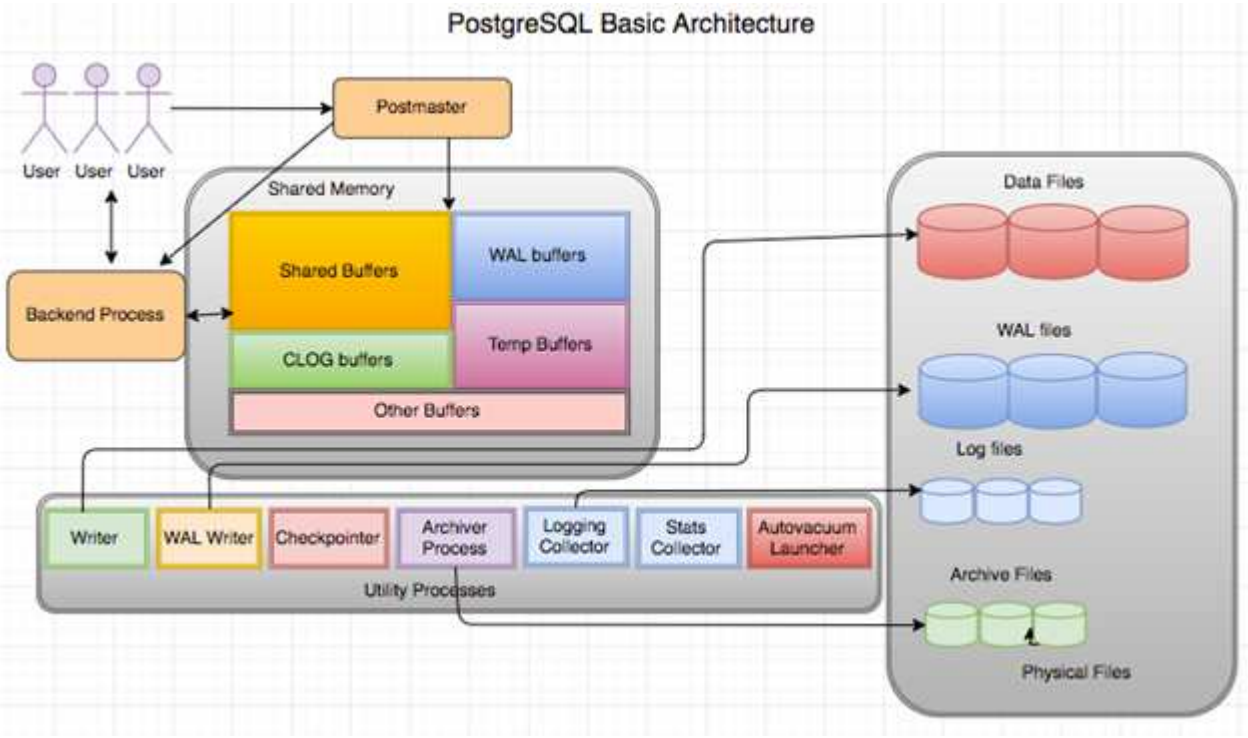
Sommaire

Configuration de la base de données . . . 1
Architecture PostgreSQL . . . 1
Paramètres d’initialisation PostgreSQL . . . 2
Configuration de la base de données PostgreSQL avec ONTAP . . . 2
Tablespaces PostgreSQL . . . 4

Configuration de la base de données

Architecture PostgreSQL

PostgreSQL est un SGBDR basé sur l'architecture client et serveur. Une instance PostgreSQL est appelée cluster de base de données, qui est une collection de bases de données par opposition à une collection de serveurs.



Il existe trois éléments principaux dans une base de données PostgreSQL : le postmaster, le front end (client) et le back end Le client envoie des demandes au postmaster avec des informations telles que le protocole IP et la base de données à laquelle se connecter. Le postmaster authentifie la connexion et la transmet au processus d'arrière-plan pour une communication plus poussée. Le processus back-end exécute la requête et envoie les résultats directement au frontal (client).

Une instance PostgreSQL est basée sur un modèle multiprocessus au lieu d'un modèle multithread. Il génère plusieurs processus pour différents travaux, et chaque processus possède sa propre fonctionnalité. Les principaux processus incluent le processus client, le processus WAL writer, le processus Background writer et le processus checkpointer :

- Lorsqu'un processus client (premier plan) envoie des demandes de lecture ou d'écriture à l'instance PostgreSQL, il ne lit pas ou n'écrit pas les données directement sur le disque. Il met d'abord en mémoire tampon les données dans des tampons partagés et des tampons WAL (Write-Ahead Logging).
- Un processus WAL writer manipule le contenu des tampons partagés et des tampons WAL pour écrire dans les journaux WAL. Les journaux WAL sont généralement des journaux de transaction de PostgreSQL et sont écrits de manière séquentielle. Par conséquent, pour améliorer le temps de réponse de la base de données, PostgreSQL écrit d'abord dans les journaux de transactions et reconnaît le client.
- Pour mettre la base de données dans un état cohérent, le processus de l'enregistreur d'arrière-plan vérifie périodiquement la présence de pages sales dans le tampon partagé. Il purge ensuite les données sur les fichiers de données stockés sur des volumes NetApp ou des LUN.

- Le processus CheckPointer s'exécute également périodiquement (moins fréquemment que le processus d'arrière-plan) et empêche toute modification des tampons. Il signale au processus d'écriture WAL d'écrire et de vider l'enregistrement de point de contrôle à la fin des journaux WAL stockés sur le disque NetApp. Il signale également au processus d'écriture d'arrière-plan d'écrire et de vider toutes les pages sales sur le disque.

Paramètres d'initialisation PostgreSQL

Vous créez un nouveau cluster de base de données à l'aide de `initdb` programme. An `initdb` script crée les fichiers de données, les tables système et les bases de données modèles (template0 et template1) qui définissent le cluster.

La base de données de modèles représente une base de données de stock. Il contient des définitions pour les tables système, les vues standard, les fonctions et les types de données. `pgdata` sert d'argument à l' `initdb` script qui spécifie l'emplacement du cluster de base de données.

Tous les objets de base de données dans PostgreSQL sont gérés en interne par les OID respectives. Les tables et les index sont également gérés par des OID individuelles. Les relations entre les objets de base de données et leurs OID respectives sont stockées dans les tables de catalogue système appropriées, selon le type d'objet. Par exemple, les OID des bases de données et des tables de segment de mémoire sont stockées dans `pg_database` et `pg_class, respectivement. Vous pouvez déterminer les OID en émettant des requêtes sur le client PostgreSQL.

Chaque base de données a ses propres tables et fichiers d'index qui sont limités à 1 Go. Chaque table a deux fichiers associés, avec le suffixe respectivement `_fsm` et `_vm`. Ils sont appelés carte de l'espace libre et carte de visibilité. Ces fichiers stockent les informations relatives à la capacité d'espace libre et ont une visibilité sur chaque page du fichier de table. Les index ne disposent que de cartes d'espace libre individuelles et ne disposent pas de cartes de visibilité.

Le `pg_xlog/pg_wal` le répertoire contient les journaux d'écriture anticipée. Des journaux d'écriture anticipée sont utilisés pour améliorer la fiabilité et les performances de la base de données. Chaque fois que vous mettez à jour une ligne dans une table, PostgreSQL écrit d'abord la modification dans le journal d'écriture anticipée, puis écrit les modifications sur les pages de données réelles sur un disque. Le `pg_xlog` le répertoire contient généralement plusieurs fichiers, mais initdb ne crée que le premier. Des fichiers supplémentaires sont ajoutés si nécessaire. Chaque fichier xlog fait 16 Mo de long.

Configuration de la base de données PostgreSQL avec ONTAP

Il existe plusieurs configurations de réglage PostgreSQL qui peuvent améliorer les performances.

Les paramètres les plus utilisés sont les suivants :

- `max_connections = <num>`: Le nombre maximal de connexions de base de données à avoir en même temps. Utilisez ce paramètre pour limiter l'échange sur le disque et l'arrêt des performances. Selon les besoins de votre application, vous pouvez également régler ce paramètre pour les paramètres du pool de connexions.
- `shared_buffers = <num>`: La méthode la plus simple pour améliorer les performances de votre serveur de base de données. La valeur par défaut est faible pour la plupart des matériels modernes. Il est défini pendant le déploiement à environ 25 % de la RAM disponible sur le système. Ce paramètre varie en

fonction de la façon dont il fonctionne avec des instances de base de données particulières ; vous devrez peut-être augmenter ou diminuer les valeurs par tâtonnement et erreur. Cependant, le réglage haut risque de dégrader les performances.

- `effective_cache_size = <num>`: Cette valeur indique à l'optimiseur de PostgreSQL la quantité de mémoire disponible pour la mise en cache des données et aide à déterminer si un index doit être utilisé. Une valeur plus élevée augmente la probabilité d'utiliser un index. Ce paramètre doit être défini sur la quantité de mémoire allouée à `shared_buffers` Plus la quantité de cache du système d'exploitation disponible. Cette valeur représente souvent plus de 50 % de la mémoire système totale.
- `work_mem = <num>`: Ce paramètre contrôle la quantité de mémoire à utiliser dans les opérations de tri et les tables de hachage. Si vous effectuez un tri important dans votre application, vous devrez peut-être augmenter la quantité de mémoire, mais soyez prudent. Ce n'est pas un paramètre à l'échelle du système, mais un paramètre par opération. Si une requête complexe comporte plusieurs opérations de tri, elle utilise plusieurs unités de mémoire Work_mem et plusieurs back end peuvent le faire simultanément. Cette requête peut souvent amener votre serveur de base de données à échanger si la valeur est trop élevée. Cette option était auparavant appelée sort_mem dans les anciennes versions de PostgreSQL.
- `fsync = <boolean> (on or off)`: Ce paramètre détermine si toutes vos pages WAL doivent être synchronisées sur le disque à l'aide de fsync() avant qu'une transaction ne soit validée. Sa désactivation peut parfois améliorer les performances d'écriture et son activation renforce la protection contre le risque de corruption en cas de panne du système.
- `checkpoint_timeout`: Le processus de point de contrôle vide les données validées sur le disque. Cela implique de nombreuses opérations de lecture/écriture sur le disque. La valeur est définie en secondes et les valeurs inférieures réduisent le temps de reprise après incident et l'augmentation des valeurs peut réduire la charge sur les ressources système en réduisant les appels au point de contrôle. En fonction de la criticité de l'application, de l'utilisation et de la disponibilité de la base de données, définissez la valeur de Checkpoint_timeout.
- `commit_delay = <num>` et `commit_siblings = <num>`: Ces options sont utilisées ensemble pour aider à améliorer les performances en écrivant plusieurs transactions qui sont exécutées simultanément. Si plusieurs objets commit_frames sont actifs à l'instant où votre transaction est validée, le serveur attend les microsecondes commit_delay pour essayer de valider plusieurs transactions à la fois.
- `max_worker_processes / max_parallel_workers`: Configurer le nombre optimal de travailleurs pour les processus. Max_Parallel_workers correspond au nombre de CPU disponibles. Selon la conception de l'application, les requêtes peuvent nécessiter un nombre réduit de collaborateurs pour les opérations parallèles. Il est préférable de conserver la même valeur pour les deux paramètres, mais d'ajuster la valeur après le test.
- `random_page_cost = <num>`: Cette valeur contrôle la façon dont PostgreSQL affiche les lectures de disque non séquentielles. Une valeur plus élevée signifie que PostgreSQL est plus susceptible d'utiliser une analyse séquentielle au lieu d'une analyse d'index, indiquant que votre serveur a des disques rapides Modifier ce paramètre après avoir évalué d'autres options telles que l'optimisation basée sur un plan, l'aspiration, l'indexation pour modifier les requêtes ou le schéma.
- `effective_io_concurrency = <num>`: Ce paramètre définit le nombre d'opérations d'E/S de disque simultanées que PostgreSQL tente d'exécuter simultanément. L'augmentation de cette valeur augmente le nombre d'opérations d'E/S que toute session PostgreSQL individuelle tente d'initier en parallèle. La plage autorisée est comprise entre 1 et 1,000, ou zéro pour désactiver l'émission de demandes d'E/S asynchrones. Actuellement, ce paramètre n'affecte que les analyses de tas bitmap. Les disques SSD et les autres systèmes de stockage basés sur la mémoire (NVMe) peuvent souvent traiter un grand nombre de requêtes simultanées. Le meilleur choix peut donc se situer dans les centaines.

Consultez la documentation PostgreSQL pour obtenir une liste complète des paramètres de configuration PostgreSQL.

TOASTS

TOAST est l'acronyme de Oversized-Attribute Storage technique. PostgreSQL utilise une taille de page fixe (généralement 8 Ko) et ne permet pas aux blocs de données de couvrir plusieurs pages. Par conséquent, il n'est pas possible de stocker directement des valeurs de champ importantes. Lorsque vous essayez de stocker une ligne qui dépasse cette taille, TOAST divise les données de grandes colonnes en « morceaux » plus petits et les stocke dans une table de TOASTS.

Les grandes valeurs des attributs toastés sont extraites (si elles sont sélectionnées) uniquement au moment où le jeu de résultats est envoyé au client. La table elle-même est beaucoup plus petite et peut contenir plus de lignes dans le cache du tampon partagé qu'elle ne le pouvait sans stockage hors ligne (TOAST).

VIDE

En mode PostgreSQL normal, les blocs de données supprimés ou rendus obsolètes par une mise à jour ne sont pas physiquement supprimés de leur table ; ils restent présents jusqu'à ce que LE VIDE soit exécuté. Par conséquent, vous devez faire fonctionner le VIDE régulièrement, en particulier sur les tables fréquemment mises à jour. L'espace qu'il occupe doit ensuite être récupéré pour réutilisation par de nouvelles lignes, afin d'éviter une panne d'espace disque. Cependant, il ne renvoie pas l'espace vers le système d'exploitation.

L'espace libre dans une page n'est pas fragmenté. VIDE réécrit le bloc entier, en empaquant efficacement les lignes restantes et en laissant un seul bloc contigu d'espace libre dans une page.

En revanche, LE VIDE COMPLET composera activement les tables en écrivant une version complètement nouvelle du fichier table sans espace mort. Cette action réduit la taille de la table mais peut prendre un certain temps. Elle nécessite également de l'espace disque supplémentaire pour la nouvelle copie de la table jusqu'à ce que l'opération soit terminée. L'objectif du VIDE DE routine est d'éviter toute activité de VIDE COMPLET. Ce processus permet non seulement de conserver les tables à leur taille minimale, mais également de conserver une utilisation régulière de l'espace disque.

Tablespaces PostgreSQL

Deux tablespaces sont créés automatiquement lorsque le cluster de base de données est initialisé.

Le `pg_global` l'espace table est utilisé pour les catalogues système partagés. Le `pg_default` tablespace est l'espace table par défaut des bases de données templage1 et template0. Si la partition ou le volume sur lequel le cluster a été initialisé est à court d'espace et ne peut pas être étendu, un espace table peut être créé sur une partition différente et utilisé jusqu'à ce que le système puisse être reconfiguré.

Un index très utilisé peut être placé sur un disque rapide et hautement disponible, comme un périphérique SSD. Par ailleurs, une table qui stocke des données archivées rarement utilisées ou non critiques pour les performances peut être stockée sur un système sur disque moins onéreux et plus lent, tel que des disques SAS ou SATA.

Les tablespaces font partie du cluster de base de données et ne peuvent pas être traités comme un ensemble autonome de fichiers de données. Elles dépendent des métadonnées contenues dans le répertoire de données principal et ne peuvent donc pas être reliées à un autre cluster de base de données ou sauvegardées individuellement. De même, si vous perdez un espace de table (suite à la suppression d'un fichier, à une panne de disque, etc.), le cluster de base de données peut devenir illisible ou ne pas démarrer. Le fait de placer un tablespace sur un système de fichiers temporaire, tel qu'un disque RAM, risque de nuire à la fiabilité de l'ensemble du cluster.

Une fois créé, un espace table peut être utilisé à partir de n'importe quelle base de données si l'utilisateur demandeur dispose de privilèges suffisants. PostgreSQL utilise des liens symboliques pour simplifier l'implémentation des tablespaces. PostgreSQL ajoute une ligne au `pg_tablespace` Tableau (table à l'échelle du cluster) et attribue un nouvel identifiant d'objet (OID) à cette ligne. Enfin, le serveur utilise l'OID pour créer un lien symbolique entre votre cluster et le répertoire donné. Le répertoire `$PGDATA/pg_tblspc` contient des liens symboliques pointant vers chacun des tablespaces non intégrés définis dans le cluster.

Informations sur le copyright

Informations sur les marques commerciales