tellora

Administration

Architecture

PostgreSQL

Architecture PostgreSQL

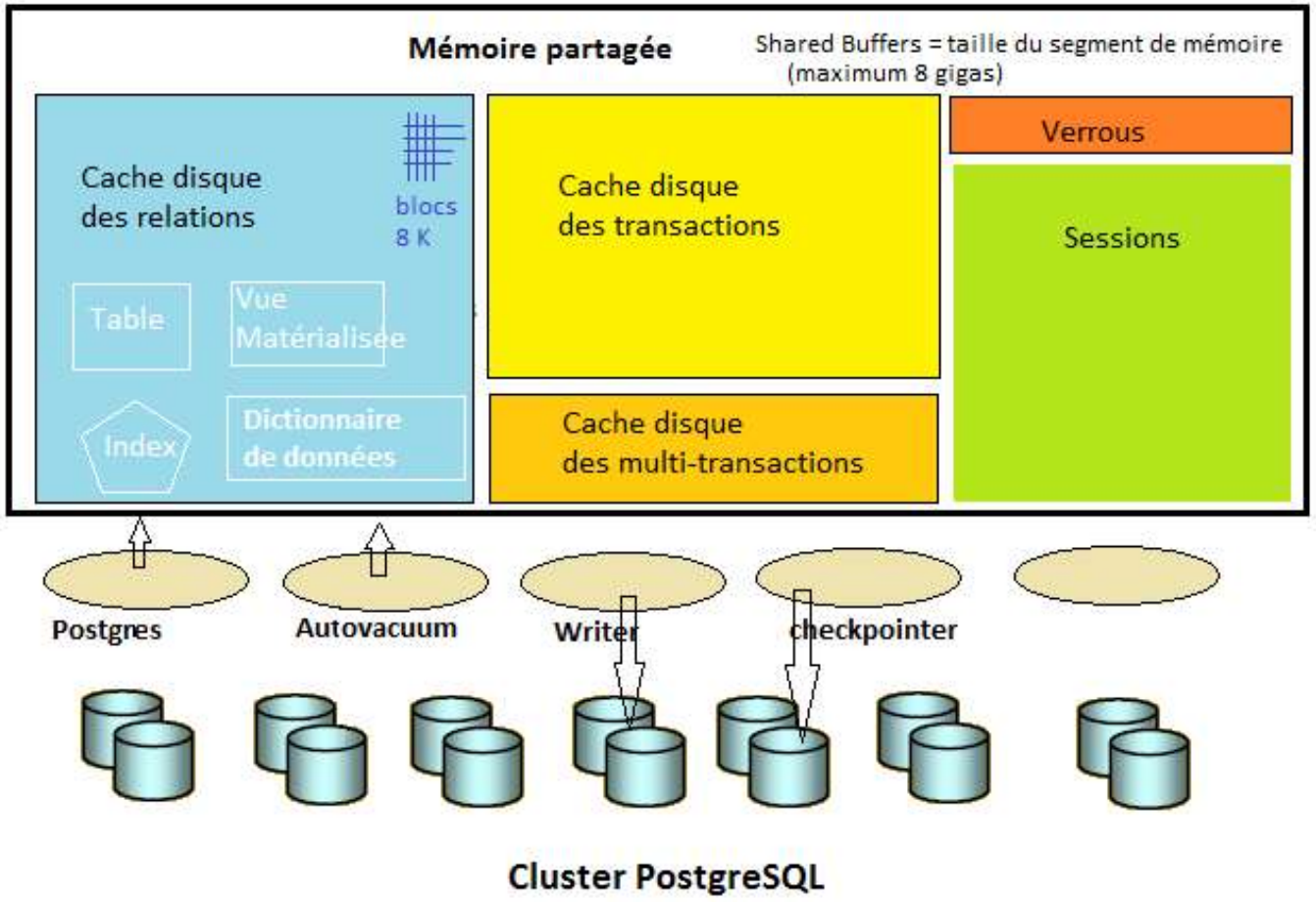
- Instance PostgreSQL
  - Une instance (ou cluster) est un groupe de base de données
    - Un cluster Postgresql peut contenir plusieurs bases de données
    - Un utilisateur
    - Des processus
    - La mémoire consommées par ces processus
    - Les ports de communication ouverts par le serveur
    - Le répertoire de fichiers, contient :
      - Fichiers de configuration
      - Fichiers de données
      - Fichiers temporaires
      - Fichiers de travail
      - Fichiers de traces

Architecture PostgreSQL

- Architecture mémoire
  - PostgreSQL utilise la mémoire partagée pour
    - Diminuer les lectures et écritures disque
    - Partager des informations entre les processus
  - La mémoire partagée est allouée au démarrage de PostgreSQL par le processus « *postmaster* »
  - Contient :
    - Le Cache disque des fichiers
    - Le Cache disque des transactions
    - Le Cache disque des multi-transactions
    - Les verrous
    - Les données des sessions
    - Des données diverses

Architecture PostgreSQL

- Paramètre « Shared_buffer » permet de tailler le cache disque des relations



Architecture PostgreSQL

- Cache disque des relations
  - PostgreSQL maintient son cache disque des relations
  - Chaque lecture et écriture sur les données passe par ce cache
  - Les processus *postgres* et *autovacuum worker* y placent les données
  - Les processus *writer* et *checkpointer* écrivent les données modifiées en cache sur le disque
  - PostgreSQL travaille par bloc de 8K

Architecture PostgreSQL

- Le cache disque des relations est composé de quatre segments
  - Les buffers Descriptors
    - descripteur de chaque bloc contenu dans le cache
  - Les buffers blocs
    - les blocs mis en cache
  - La shared buffer lookup table
    - table de recherche des blocs
  - Le buffer strategy status
    - bloc de contrôle
    - Taille fixe de 28 octets, contient le prochain élément à utiliser, le premier élément libre, dernier élément libre, nombre de blocs allouées, etc …

Architecture PostgreSQL

- Conseilles
  - Par défaut PostgreSQL a un cache disque des relations paramétré à 128 Mo
    - Paramètre shared_buffers
    - Conseillé de positionner ce paramètre à ¼ de la valeur du serveur (en serveur dédié)
    - Conseillé de ne pas dépasser 8 Go, faute de problèmes de ralentissements
    - La requête ci-dessous permet devoir le contenu du cache
      - SELECT * FROM pg_buffercache ;

Architecture PostgreSQL

- Cache disque des transactions et multi-transactions
  - Contient
    - Les journaux de transactions
    - Le statut de chaque transaction
  - Les fichiers correspondant sont stockés dans les répertoires « pg_xlog », pg_clog », « pg_subtrans » et « pg_multixact »
  - Chaque répertoire dispose de son propre segment de mémoire partagée
    - Xlog ctl pour stocker en cache les enregistrements des journaux de transactions
    - Segment Clog ctl pour stocker en cache les statuts des transactions
    - Segment subtrans ctl pour stocker en cache le contenu des fichiers du répertoire pg_subtrans

Architecture PostgreSQL

- Le cache des verrous
  - La gestion des verrous est réalisée en mémoire
  - Contient la liste des verrous posés et en attente
  - Contient la mémoire utilisée et partagée entre les différents processus
- Le cache des sessions
  - Contient des informations sur les sessions en cours d'exécution
  - Certaines informations permettent de tracer les sessions en cours et leur état

Architecture PostgreSQL

- PostgreSQL possède cinq processus d'arrière plan par défaut :
  - Postgres : processus d'écoute des connexions
  - writer process : processus d'écritures
  - wal writer process : processus d'écritures
  - autovacuum launcher process : processus de maintenance des tables
  - stats collector process : processus de collecte des statistiques
- La plupart ne sont présents qu'en un exemplaire
  - Seul celui qui se charge des communications avec les clients peut apparaître plusieurs fois
- Ils utilisent principalement la mémoire partagée, le reste dépend de leur travail mais ne représente pas beaucoup

Architecture PostgreSQL

- Processus Postgres
  - considéré comme le démon principal de PostgreSQL
    - père de tous les autres processus du serveur PostgreSQL
  - Souvent dénommé « postmaster », son but principal est d'écouter toutes les connexions entrantes,
    - soit par la socket
    - soit par le port TCP/IP
  - Au démarrage, il charge la configuration, réalise tout un ensemble de tests, supprime des fichiers temporaires qui auraient été laissés dans le répertoire des données de PostgreSQL lors d'une précédente exécution, crée un fichier verrou dans le répertoire des données, alloue la mémoire partagée, initialise certains structures
    - S'il ne peut obtenir cette mémoire, le serveur complet s'arrête.
      - Cela peut arriver par manque mémoire

Architecture PostgreSQL

- Processus Postgres (suite)
  - postmaster n'utilise pas la mémoire partagée
    - Par contre il alloue la mémoire partagée
  - En cas de crash des processus fils, il est capable de nettoyer le système en supprimant la mémoire partagée
  - Au démarrage il va lancer les différents services nécessaire à PostgreSQL dans l'ordre :
    - le processus de gestion des journaux applicatifs (si activé)
      - autovacuum launcher process
    - le processus de collecte des statistiques (si activé)
      - stats collector process
    - le « autovacuum launcher » (si activé)
    - le processus d'écriture en tâche de fond
      - Writer process
    - le processus d'écriture des journaux de transactions en tâche de fond
      - Wal Writer process
  - Quand une demande de connexion arrive, un processus fils est immédiatement créé
    - Ce dernier est responsable de l'identification et l'authentification du client et, si tout va bien, est chargé de la communication client/serveur

Architecture PostgreSQL

- Writer process
  - Processus d'écriture en tâche de fond
    - aussi appelé bgwriter (pour background writer), va de temps à autre enregistrer les blocs modifiés du cache disque (mémoire cache) dans les fichiers de données
    - Cette action d'enregistrement est généralement déclenchée par un checkpoint
    - Il n'y a qu'un seul processus activé aussi les processus postgres peuvent toujours écrire si le processus d'écriture en tâche de fond n'arrive pas à tenir le rythme
    - bgwriter est exécuté au lancement du serveur PostgreSQL et vit jusqu'à l'arrêt du serveur

Architecture PostgreSQL

- Writer process (suite)
  - Si ce processus meurt de façon inattendue, le processus postmaster traite cet événement comme la mort inattendue d'un processus postgres :
    - arrêt du serveur PostgreSQL en urgence
    - Au prochain démarrage, une procédure de restauration aura lieu
    - Quelques variables permettent de configurer le moment de l'exécution d'un checkpoint :
      - checkpoint_timeout indique la durée maximale sans CHECKPOINT
      - checkpoint_segments indique le nombre maximum de journaux de transactions utilisés sans CHECKPOINT

Architecture PostgreSQL

- Writer process (suite)
  - Par défaut, il y a un checkpoint au maximum :
    - toutes les cinq minutes
    - et à chaque fois que trois journaux de transactions ont été entièrement utilisés
  - Il s'est avéré par la suite que le checkpoint générait des pics importants d'activité
  - D'autres paramètres ont fait leur apparition pour permettre dde lisser son activité
  - Par exemple, (en 8.3),
    - le paramètre checkpoint_completion_target permet de faire en sorte que les écritures se fassent sur X% du temps entre deux checkpoints.
    - Par défaut à 0,5 (soit 50 %), il est possible d'augmenter ce paramètre jusqu'à 0,9 (90 %) pour diluer encore plus les écriture et lisser son activité
  - Plutôt que d'écrire tous les blocs modifiés en un seul coup, on peut faire en sorte que, une fois une limite dépassée, qu'un certain temps d'attente soit respecté
    - Le délai est configuré avec le paramètre bgwriter_delay (200 ms par défaut)
    - Quant à la limite, elle est calculée par rapport à deux paramètres : bgwriter_lru_maxpages, nombre maximum de blocs écrit sur disque et bgwriter_lru_multiplierut)

Architecture PostgreSQL

- Writer process (suite)
  - Lorsqu'un journal de transactions est archivé, il reste dans le répertoire pg_xlog/archive_status un fichier du nom du journal de transactions mais avec une extension .done.
    - Le processus bgwriter peut supprimer ce fichier, et traiter le journal de transactions
      - soit en le renommant
      - soit en le supprimant
  - Au niveau mémoire, ce processus utilise un peu de mémoire partagée, la quantité dépendant essentiellement de la taille de la mémoire cache de PostgreSQL
    - Par exemple sur un serveur avec 24 Mo de mémoire cache (la valeur maximale par défaut), la mémoire allouée est de 29 Ko

Architecture PostgreSQL

- Checkpointer process
  - Processus d'écriture dans les fichiers de données
    - Contrairement au processus Writer, qui exécute de nombreux et rapides nettoyages du cache disque
    - Le processus checkpointer va prendre en considération la totalité du cache et écrire sur disque tout ce qui est modifié
  - Gère les checkpoints
    - Déclenchés par le paramêtre checkpoint_timeout (durée entre chaque checkpoint) et par les demandes des processus
      - Les demandes de checkpoint viennent des processus postgres qui écrivent dans les journaux de transaction

Architecture PostgreSQL

- Wal Writer process
  - apparu en 8.3 pour gérer l'écriture en tâche de fond des journaux de transactions
  - Il enregistre les modifications dans les journaux de transactions au moment du COMMIT ou si les informations à enregistrer ne tiennent plus dans le cache
  - Ce cache spécifique aux journaux de transactions permet d'éviter les écritures tant que le COMMIT n'est pas reçu
    - Cependant, comme tout cache, il a une limite paramétré avec la variable wal_buffers. En cas de débordement du cache, le « wal writer process » écrit les données sur disque
    - De plus, dans le cas où l'enregistrement asynchrone est activé (paramètre asynchronous_commit), il garantit que l'enregistrement se fait au plus tard au bout de trois fois le délai indiqué par le paramètre wal_writer_delay

Architecture PostgreSQL

- Stats collector process
  - Processus de collecte des statistiques sur le nombre de lignes lues, insérées, modifiées et supprimées
    - Ce sont aussi des statistiques sur le nombre de blocs disque lus ou écrits
    - toutes sortes d'informations sur l'activité du serveur
    - à condition que le paramètre track_count soit activé
  - Ce processus est activé par défaut en 8.3
    - Contrairement aux deux précédents types de processus, il peut être désactivé grâce au paramètre track_activities

Architecture PostgreSQL

- Stats collector process (suite)
  - Ces informations sont récupérées via un port UDP configuré en mode non bloquant, ce qui permet, en cas de retard du collecteur, que les messages soient ignorés et ne ralentissent pas le serveur PostgreSQL
    - Les données sont temporairement enregistrées dans le fichier global/pgstat.tmp
    - Une fois l'enregistrement terminé, le fichier est renommé en global/pgstat.stat
      - Le fichier est d'autant plus gros que le nombre de bases et relations est important
      - Le fichier peut devenir très gros si un grand nombre de tables temporaires est créé
      - Cela génère une activité non négligeable étant donné que le fichier est écrit à chaque travail du collecteur de statistiques, c'est-à-dire toutes les 500 ms
    - il n'utilise pas la mémoire partagée mais comprend quelques structures qu'il conserve tout au long de son exécution
      - représente environ 128 Ko de mémoire partagée allouée jusqu'à cent processus

Architecture PostgreSQL

- Autovacuum launcher
  - Existe depuis la version8.3, exécuté dès le démarrage par le processus postmaster
    - S'arrête avec l'arrêt de l'instance
    - Si ce processus meurt de façon inattendue, le processus postmaster tente de le relancer plusieurs fois si nécessaire
  - a pour but de procéder au nettoyage des tables si elles ont eu une activité suffisante
  - le démon principal, nommé « autovacuum launcher », et des processus secondaires, appelés « autovacuum worker

Architecture PostgreSQL

- Autovacuum launcher (suite)
  - Au démarrage, il commence par allouer un petit espace de mémoire partagée entre tous les processus autovacuum. Cet espace mémoire est vraiment restreint en utilisation aux processus autovacuum et a une taille très limitée
    - 144 octets sur un serveur en moyenne
    - Cette taille dépend du paramètre autovacuum_max_workers
    - Lorsqu'un « autovacuum worker » exécute réellement un VACUUM, une mémoire supplémentaire est allouée, dont la taille dépend du paramètre maintenance_work_mem
  - L'allocation mémoire réalisée, il s'endort et se réveille à intervalle régulier dépendant du paramètre autovacuum_naptime
    - Lors de ce réveil, il indique la base de données à traiter dans l'espace en mémoire partagée qu'il a alloué à son lancement et demande au processus postmaster d'exécuter un processus « autovacuum worker »
    - Il ne peut exécuter lui-même un « autovacuum_worker » car il est moins robuste que le processus postmaster, notamment parce qu'il utilise la mémoire partagée où peut survenir une corruption
    - Le processus fils autovacuum worker se connecte à la base de données indiquée dans la mémoire partagée, recherche les tables et index à traiter, les traite puis quitte en envoyant un signal SIGUSR1 au processus « autovacuum launcher ».
      - Ainsi, il prévient ce processus pour que ce dernier puisse demander l'exécution d'un nouvel « autovacuum worker »
      - Plusieurs « autovacuum worker » peuvent fonctionner en même temps, le nombre maximum dépend du paramètre autovacuum_max_workers

Architecture PostgreSQL

- Archiver process
  - Processus d'archivage désactivé par défaut,
    - ce processus a pour but de gérer l'archivage des journaux de transactions
    - L'activation se fait en configurant le paramètre « archive_mode » à on et le paramètre « wal_level » à la valeur « archive au minimum »
  - Il n'a pas de consommation mémoire importante et il n'utilise pas non plus la mémoire partagée
  - Si jamais il meurt de façon inattendue, le serveur tentera plusieurs fois de le relancer

Architecture PostgreSQL

- Archiver process (suite)
  - Il communique avec les autres processus au moyen de signaux
  - S'il reçoit le signal SIGUSR1 ou au plus tard au bout de 60 secondes,
    - il lit le contenu du répertoire pg_xlog/archive_status.
    - Il récupère le fichier d'extension .ready le plus ancien,
    - Il en déduit le nom du journal de transactions à archiver et exécute la commande d'archivage.
    - Si cette dernière renvoie le code de statut 0 (signifiant la réussite de l'opération), il renomme le fichier .ready en .done et se rendort

Architecture PostgreSQL

- Archiver process (suite)
  - Le nom du processus affiché dépend de son état :
    - « archiver process » au départ
    - « archiving %s », lors de l'archivage du journal de transactions %s
    - « failed on %s » si l'archivage a échoué pour le journal %s
    - « last was %s » dès qu'un journal de transactions a été archivé avec succès

Architecture PostgreSQL

- Processus de communication client/serveur
  - Processus postgres (se multiplie)
  - Chaque processus s'occupe de la communication entre le client qui s'est connecté et le serveur, que ce client soit l'outil psql, l'outil de sauvegarde pg_dump ou n'importe quel autre application capable de se connecter à une base de données PostgreSQL
    - il n'y en aura jamais plus de max_connections
  - Ils utilisent principalement la mémoire partagée pour stocker les pages des tables et index qu'ils utilisent
    - Ils utilisent aussi une partie de mémoire qui leur est propre, par exemple pour les tris ou pour les créations d'index

Architecture PostgreSQL

- Processus de communication client/serveur
  - Processus postgres (suite)
  - Lorsqu'un client se connecte, le premier travail du processus postgres est de s'assurer de l'identité du client
  - Ceci fait, il effectue si nécessaire son authentification
  - Ensuite, il est en attente des requêtes du client
    - À réception d'une requête, il procède à son analyse et à son exécution
    - Pour cela, il peut avoir besoin de lire des tables qu'il placera dans la mémoire partagée
    - Une fois le résultat obtenu, il envoie le résultat au client
    - Il envoie les traces au collecteur de traces, récupère les statistiques qu'il fournir au collecteur de statistiques