Configurazione del database

Enterprise applications

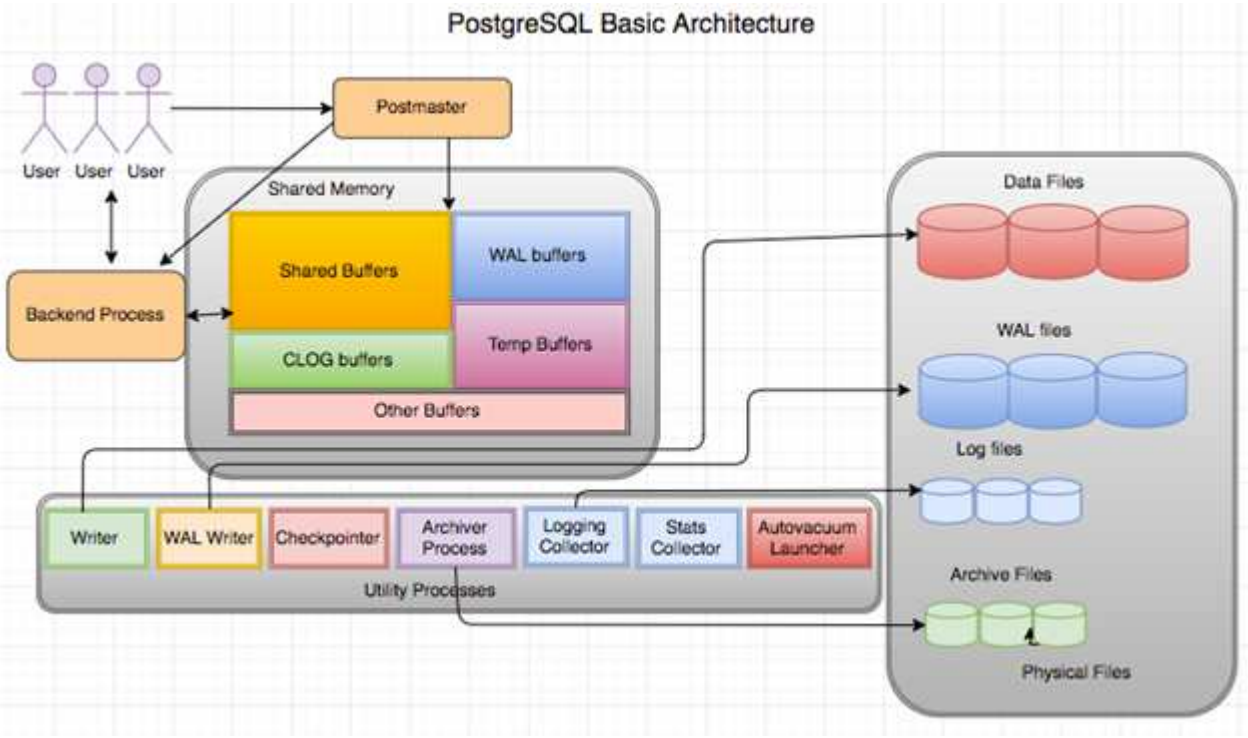
NetApp
February 10, 2026

Sommario

Configurazione del database

Architettura

PostgreSQL è un RDBMS basato su architettura client e server. Un'istanza di PostgreSQL è nota come cluster di database, ovvero una raccolta di database anziché una raccolta di server.



Un database PostgreSQL contiene tre elementi principali: Il postmaster, il front-end (client) e il back-end Il client invia richieste al postmaster con informazioni quali il protocollo IP e il database a cui connettersi. Il postmaster autentica la connessione e la passa al processo back-end per ulteriori comunicazioni. Il processo back-end esegue la query e invia i risultati direttamente al front-end (client).

Un'istanza PostgreSQL si basa su un modello multiprocesso anziché su un modello multithread. Genera più processi per diversi processi e ogni processo ha una propria funzionalità. I processi principali includono il processo client, il processo di scrittura WAL, il processo di scrittura in background e il processo di checkpointer:

- Quando un processo client (in primo piano) invia richieste di lettura o scrittura all'istanza PostgreSQL, non legge o scrive dati direttamente sul disco. Innanzitutto, memorizza i dati nei buffer condivisi e nei buffer WAL (Write-ahead logging).
- Un processo di scrittura WAL manipola il contenuto dei buffer condivisi e dei buffer WAL da scrivere nei registri WAL. I registri WAL sono in genere registri di transazioni di PostgreSQL e vengono scritti in sequenza. Pertanto, per migliorare i tempi di risposta dal database, PostgreSQL scrive prima nei registri delle transazioni e riconosce il client.
- Per impostare il database in uno stato coerente, il processo di scrittura in background verifica periodicamente la presenza di pagine sporche nel buffer condiviso. Quindi, scarica i dati sui file di dati che sono memorizzati su volumi NetApp o LUN.

- Anche il processo checkpointer viene eseguito periodicamente (meno frequentemente del processo in background) e impedisce qualsiasi modifica ai buffer. Segnala al processo di scrittura WAL di scrivere e svuotare il record del punto di verifica alla fine dei registri WAL memorizzati sul disco NetApp. Segnala inoltre al processo di scrittura in background di scrivere e scaricare tutte le pagine sporche sul disco.

Parametri di inizializzazione

È possibile creare un nuovo cluster di database utilizzando `initdb` programma. An `initdb` script crea i file di dati, le tabelle di sistema e i database dei modelli (template0 e template1) che definiscono il cluster.

Il database dei modelli rappresenta un database di stock. Contiene le definizioni per le tabelle di sistema, le viste standard, le funzioni e i tipi di dati. `pgdata` funge da argomento per il `initdb` script che specifica la posizione del cluster di database.

Tutti gli oggetti di database in PostgreSQL sono gestiti internamente dai rispettivi OID. Le tabelle e gli indici sono inoltre gestiti da singoli OID. Le relazioni tra gli oggetti del database e i rispettivi OID vengono memorizzate in tabelle di cataloghi di sistema appropriate, a seconda del tipo di oggetto. Ad esempio, gli OID dei database e delle tabelle heap vengono memorizzati in `pg_database` e `pg_class, rispettivamente. È possibile determinare gli OID eseguendo query sul client PostgreSQL.

Ogni database ha le proprie tabelle e i file di indice che sono limitati a 1GB. Ogni tabella ha due file associati, rispettivamente con il suffisso `_fsm` e. `_vm`. Sono indicate come mappa dello spazio libero e mappa di visibilità. Questi file memorizzano le informazioni sulla capacità di spazio libero e hanno visibilità su ogni pagina del file di tabella. Gli indici hanno solo mappe di spazio libero individuali e non hanno mappe di visibilità.

Il `pg_xlog/pg_wal` la directory contiene i registri write-ahead. I registri write-ahead sono utilizzati per migliorare l'affidabilità e le performance del database. Ogni volta che si aggiorna una riga in una tabella, PostgreSQL scrive prima la modifica nel registro write-ahead e successivamente scrive le modifiche alle pagine di dati effettive su un disco. Il `pg_xlog` la directory di solito contiene diversi file, ma initdb crea solo il primo. I file aggiuntivi vengono aggiunti in base alle necessità. Ciascun file xlog è lungo 16MB MB.

Impostazioni

Esistono diverse configurazioni di ottimizzazione PostgreSQL che possono migliorare le prestazioni.

I parametri più comunemente utilizzati sono i seguenti:

- `max_connections = <num>`: Il numero massimo di connessioni al database da avere contemporaneamente. Utilizzare questo parametro per limitare lo scambio sul disco e l'interruzione delle prestazioni. A seconda delle esigenze dell'applicazione, è anche possibile regolare questo parametro per le impostazioni del pool di connessione.
- `shared_buffers = <num>`: Il metodo più semplice per migliorare le prestazioni del server di database. Il valore predefinito è basso per la maggior parte dei componenti hardware moderni. Durante l'implementazione viene impostato su circa il 25% della RAM disponibile sul sistema. Questa impostazione di parametro varia in base al funzionamento con determinate istanze di database; potrebbe essere necessario aumentare e diminuire i valori per prova ed errore. Tuttavia, l'impostazione di un livello elevato potrebbe degradare le prestazioni.
- `effective_cache_size = <num>`: Questo valore indica all'ottimizzatore di PostgreSQL la quantità di

memoria disponibile per la memorizzazione nella cache dei dati e aiuta a determinare se utilizzare un indice. Un valore maggiore aumenta la probabilità di utilizzare un indice. Questo parametro deve essere impostato sulla quantità di memoria allocata a. `shared_buffers` più la quantità di cache del sistema operativo disponibile. Spesso questo valore corrisponde a più del 50% della memoria di sistema totale.

- `work_mem = <num>`: Questo parametro controlla la quantità di memoria da utilizzare nelle operazioni di ordinamento e nelle tabelle hash. Se si esegue un ordinamento pesante nell'applicazione, potrebbe essere necessario aumentare la quantità di memoria, ma prestare attenzione. Non si tratta di un parametro a livello di sistema, ma di un parametro per operazione. Se una query complessa contiene diverse operazioni di ordinamento, utilizza più unità di memoria work_mem e più backend potrebbero farlo contemporaneamente. Questa query può spesso indurre il server di database a effettuare lo swap se il valore è troppo grande. Questa opzione era precedentemente chiamata sort_mem nelle versioni precedenti di PostgreSQL.
- `fsync = <boolean> (on or off)`: Questo parametro determina se tutte le pagine WAL devono essere sincronizzate su disco utilizzando fsync() prima che venga eseguito il commit di una transazione. Disattivandolo a volte si possono migliorare le prestazioni di scrittura e attivandolo si aumenta la protezione dal rischio di danneggiamento quando il sistema si blocca.
- `checkpoint_timeout`: Il processo del punto di verifica elimina i dati sottoposti a commit sul disco. Ciò comporta numerose operazioni di lettura/scrittura su disco. Il valore è impostato in secondi e valori inferiori riducono il tempo di recupero da crash e valori crescenti possono ridurre il carico sulle risorse di sistema riducendo le chiamate al punto di verifica. In base alla criticità dell'applicazione, all'utilizzo, alla disponibilità del database, impostare il valore di checkpoint_timeout.
- `commit_delay = <num>` e. `commit_siblings = <num>`: Queste opzioni vengono utilizzate insieme per migliorare le prestazioni scrivendo più transazioni che vengono effettuate contemporaneamente. Se ci sono diversi oggetti commit_siblings attivi nel momento in cui la transazione è in fase di commit, il server attende Commit_delay microsecondi per tentare di eseguire più transazioni contemporaneamente.
- `max_worker_processes / max_parallel_workers`: Configurare il numero ottimale di lavoratori per i processi. Max_Parallel_Workers corrisponde al numero di CPU disponibili. A seconda della progettazione dell'applicazione, le query potrebbero richiedere un numero minore di lavoratori per le operazioni parallele. È meglio mantenere lo stesso valore per entrambi i parametri, ma regolare il valore dopo la verifica.
- `random_page_cost = <num>`: Questo valore controlla il modo in cui PostgreSQL visualizza le letture del disco non sequenziali. Un valore più elevato indica che PostgreSQL è più probabile che utilizzi una scansione sequenziale invece di una scansione di indice, indicando che il server dispone di dischi veloci modificare questa impostazione dopo aver valutato altre opzioni come l'ottimizzazione basata su piano, l'aspirazione, l'indicizzazione per modificare query o schemi.
- `effective_io_concurrency = <num>`: Questo parametro imposta il numero di operazioni di i/o su disco simultanee che PostgreSQL tenta di eseguire contemporaneamente. L'aumento di questo valore aumenta il numero di operazioni di i/o che una singola sessione PostgreSQL tenta di avviare in parallelo. L'intervallo consentito è compreso tra 1 e 1.000 o zero per disattivare l'emissione di richieste i/o asincrone. Attualmente, questa impostazione influisce solo sulle scansioni bitmap heap. I dischi a stato solido (SSD) e altro storage basato su memoria (NVMe) possono spesso elaborare molte richieste simultanee, cosicché il valore migliore può essere centinaia.

Consultare la documentazione di PostgreSQL per un elenco completo dei parametri di configurazione di PostgreSQL.

TOAST

TOAST è l'acronimo di OVERSIZED-Attribute Storage Technique. PostgreSQL utilizza una dimensione di pagina fissa (in genere 8KB) e non consente alle tuple di occupare più pagine. Pertanto, non è possibile memorizzare direttamente valori di campo grandi. Quando si tenta di memorizzare una riga che supera queste dimensioni, TOAST suddivide i dati delle colonne di grandi dimensioni in "pezzi" più piccoli e li memorizza in

una tabella TOAST.

I valori elevati degli attributi tostati vengono estratti (se selezionati) solo quando il set di risultati viene inviato al client. La tabella stessa è molto più piccola e può contenere più righe nella cache buffer condivisa di quanto non possa fare senza alcuna archiviazione out-of-line (TOAST).

VUOTO

Nelle normali operazioni PostgreSQL, le tuple eliminate o rese obsolete da un aggiornamento non vengono fisicamente rimosse dalla tabella; rimangono presenti fino all'esecuzione di VACUUM. Pertanto, è necessario eseguire il VUOTO periodicamente, soprattutto nelle tabelle aggiornate di frequente. Lo spazio occupato deve quindi essere recuperato per essere riutilizzato da nuove righe, per evitare di esaurire lo spazio su disco. Tuttavia, non restituisce lo spazio al sistema operativo.

Lo spazio libero all'interno di una pagina non è frammentato. L'ASPIRAPOLVERE riscrive l'intero blocco, comprimendo in modo efficiente le righe rimanenti e lasciando un singolo blocco contiguo di spazio libero in una pagina.

Al contrario, VACUUM FULL comprime attivamente le tabelle scrivendo una versione completamente nuova del file di tabella senza spazio morto. Questa azione riduce al minimo le dimensioni della tabella, ma può richiedere molto tempo. Richiede inoltre ulteriore spazio su disco per la nuova copia della tabella fino al completamento dell'operazione. L'obiettivo del VUOTO DI routine è di evitare l'attività di VUOTO PIENO. Questo processo non solo mantiene le tabelle alla loro dimensione minima, ma mantiene anche l'utilizzo costante dello spazio su disco.

Tablespace

Due tablespace vengono create automaticamente al momento dell'inizializzazione del cluster di database.

Il `pg_global` tablespace viene utilizzato per i cataloghi di sistema condivisi. Il `pg_default` tablespace è la tablespace predefinita dei database template1 e template0. Se la partizione o il volume su cui il cluster è stato inizializzato esaurisce lo spazio e non può essere esteso, è possibile creare uno spazio di tabella in un'altra partizione ed utilizzarlo fino a quando il sistema non può essere riconfigurato.

Un indice molto utilizzato può essere collocato su un disco veloce e altamente disponibile, come un dispositivo a stato solido. Inoltre, una tabella che memorizza i dati archiviati utilizzati raramente o non critici per le prestazioni può essere archiviata su un sistema su disco meno costoso e più lento, come le unità SAS o SATA.

Gli spazi di tabella fanno parte del cluster di database e non possono essere trattati come una raccolta autonoma di file di dati. Dipendono dai metadati contenuti nella directory dei dati principale e pertanto non possono essere collegati a un cluster di database diverso o sottoposti a backup individuale. Analogamente, se si perde uno spazio di tabella (a causa dell'eliminazione dei file, del guasto del disco e così via), il cluster del database potrebbe diventare illeggibile o non avviarsi. Posizionando una tablespace su un file system temporaneo come un disco RAM si rischia l'affidabilità dell'intero cluster.

Una volta creato, è possibile utilizzare un tablespace da qualsiasi database se l'utente richiedente dispone di privilegi sufficienti. PostgreSQL utilizza collegamenti simbolici per semplificare l'implementazione di tablespace. PostgreSQL aggiunge una riga al `pg_tablespace` Tabella (una tavola a livello di cluster) e assegna un nuovo identificatore di oggetto (OID) a quella riga. Infine, il server utilizza l'OID per creare un collegamento simbolico tra il cluster e la directory specificata. La directory `$PGDATA/pg_tblspc` contiene collegamenti simbolici che puntano a ciascuno degli spazi di tabella non incorporati definiti nel cluster.

Informazioni sul copyright

Informazioni sul marchio commerciale