УПРАВЛЕНИЕ ПУЛОМ СОЕДИНЕНИЙ И МАРШРУТИЗАЦИЯ ЗАПРОСОВ В POSTGRESQL

Киселёв А.С., студент

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Можей Н.П. – канд. физ.- мат. наук, доцент

Рассмотрена проблема высокой концентрации подключений к PostgreSQL и связанного с этим снижения производительности и усложнения эксплуатации. Проанализированы типовые подходы и их ограничения с точки зрения наблюдаемости и административного управления. Предложено программное средство PgPooler, объединяющее управление пулом соединений (connection pooling), маршрутизацию соединений по правилам конфигурации, сбор аналитики и интерфейсы мониторинга и управления.

PostgreSQL широко применяется в корпоративных и облачных системах [1, 2]. При росте числа одновременных соединений накладные расходы на их установление и обслуживание большого количества активных сессий становятся фактором, ограничивающим масштабирование, особенно при коротких запросах и высокой конкуренции за ресурсы БД [2]. Типовой способ снижения нагрузки – применение внешнего пуллера: ограничение числа подключений к серверу БД и их переиспользование для множества клиентских сессий.

Существующие пуллеры эффективно решают задачу ограничения числа подключений к серверу БД, однако в типовом сценарии эксплуатации часто требуют дополнительной инфраструктуры: централизованного мониторинга, хранения истории соединений и запросов, а также удобных средств оперативного управления (завершение сессий, блокировка пользователей, управление конфигурацией) [3, 4]. Это повышает трудоёмкость сопровождения и увеличивает время реакции на инциденты.

PgPooler реализован как прокси-сервер уровня соединений между клиентами PostgreSQL и серверами БД. Ядро написано на C++ (стандарт C++17) и построено на событийной модели неблокирующего ввода‑вывода (библиотека libevent), что позволяет обслуживать множество клиентских подключений при предсказуемом потреблении ресурсов.

Поддерживается два варианта развёртывания ядра: с единственным процессом (Listener) и с многопроцессной схемой «диспетчер – воркеры». В варианте с диспетчером один процесс (Dispatcher) принимает входящие TCP-соединения на заданном порту и при необходимости читает первый пакет от клиента (стартовое сообщение или SSL-запрос). По правилам маршрутизации диспетчер определяет целевой бэкенд; затем дескриптор соединения и буфер передаются одному из N воркеров через Unix domain socket (handoff). Воркер получает уже прочитанные данные и продолжает обработку сессии: аутентификация с выбранным сервером БД, работа с пулом соединений и проксирование трафика. Таким образом, приём новых соединений и их распределение по воркерам выполняет диспетчер, а каждая клиентская сессия обрабатывается одним воркером до отключения; это позволяет масштабировать нагрузку по ядрам процессора и снижать конкуренцию за общий цикл событий. Число воркеров задаётся в конфигурации.

Конфигурация задаётся в формате YAML отдельными файлами: pgpooler.yaml (адрес и порт прослушивания, пути к файлам бэкендов и правил маршрутизации, настройки воркеров, логирования и аналитики), backends.yaml (список серверов БД – хост, порт, параметры пулов по умолчанию), routing.yaml (правила маршрутизации по полям user и database – точное совпадение, список значений, префикс, регулярное выражение, правило по умолчанию). Ключевое решение – маршрутизация по параметрам стартового сообщения протокола PostgreSQL до установления полного подключения к серверу БД; выбор целевого бэкенда и пула выполняется до фазы аутентификации.

Общая схема взаимодействия приведена на рисунке 1.

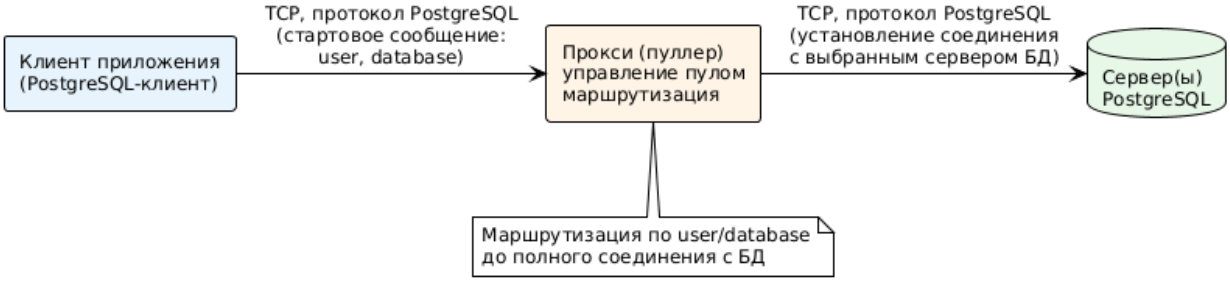

Рисунок 1 – Схема взаимодействия клиента, прокси-сервера и сервера PostgreSQL

Управление пулом настраивается режимами работы («сеанс», «транзакция», «оператор») и параметрами пулов (размеры, тайм‑ауты) [2–4]. В режиме «сеанс» выданное клиенту соединение закреплено за ним до отключения; в режиме «транзакция» соединение возвращается в пул после завершения транзакции (по сообщению ReadyForQuery с состоянием idle); в режиме «оператор» – после каждого выполненного запроса. При выдаче ранее использованного соединения на сервер БД отправляется команда сброса состояния (DISCARD ALL). При отсутствии свободного соединения в пуле клиентская сессия ставится в очередь ожидания с настраиваемым тайм‑аутом; при его истечении клиенту возвращается ошибка.

Каждому клиентскому подключению соответствует один экземпляр ClientSession в ядре: приём первого сообщения, установление соединения с бэкендом, проксирование обмена при аутентификации с кэшированием ответа до ReadyForQuery, обращение к пулу (take – получение соединения или постановка в очередь ожидания; put – возврат соединения в пул с сохранением кэша стартового ответа), определение момента возврата соединения в пул по режиму и прозрачная передача сообщений между клиентом и сервером БД. Пул соединений (BackendConnectionPool) привязан к ключу (бэкенд, пользователь, база данных); для каждого ключа хранятся свободные соединения к серверу БД и очередь сессий, ожидающих выдачи соединения (ConnectionWaitQueue). При появлении свободного соединения или по тайм-ауту ожидающие сессии уведомляются. Маршрутизатор (Router) по паре (user, database) возвращает выбранный бэкенд и параметры пула (размер, режим, тайм-ауты).

Обмен с клиентом и бэкендом осуществляется через бинарные сообщения протокола PostgreSQL (тип, длина, тело); прокси не разбирает SQL и пересылает сообщения без изменений, сохраняя совместимость с клиентами и версиями протокола. В режимах «транзакция» и «оператор» момент возврата соединения в пул задаётся ReadyForQuery ('Z') и байтом состояния транзакции I, T, E (idle / в транзакции / ошибка): в режиме «транзакция» – при переходе в idle после COMMIT/ROLLBACK, в режиме «оператор» – при каждом Z. Сессия чередует состояния Forwarding (прокси запроса и ответа) и WaitingForBackend; при данных от клиента из WaitingForBackend выполняется повторный take у пула.

Предусмотрены тайм-ауты ожидания выдачи соединения из пула (ошибка клиенту при превышении), простоя в пуле и при необходимости ограничение времени жизни соединения по возрасту. При недоступности сервера БД зависящие клиентские соединения закрываются корректно; событие фиксируется в логе и при включённой аналитике – в таблице events. Список blocked_users проверяется при подключении; блокировка и разблокировка – через REST или веб-интерфейс, хранение в базе аналитики. Конфигурация перезагружается без перезапуска процесса. Развёртывание: исполняемый файл ядра и YAML на диске; при необходимости – Docker Compose с контейнерами pgpooler, PostgreSQL, базы аналитики и веб-интерфейса в единой сети.

База аналитики (PostgreSQL, схема pgpooler) накапливает историю: connection_sessions (сессия, воркер, адрес клиента, пользователь, база, бэкенд, режим пула, время подключения и отключения), query_fingerprints и queries (отпечаток, хеш, текст запроса, длительность, тип команды), pool_snapshots (размер пула, число свободных, занятых и ожидающих соединений) – для отчётности и визуализации. Запись из ядра прокси при включённой настройке. Администрирование – отдельный сервис: REST API (HTTP, JSON) – статус, активные сессии, завершение сессии, блокировка пользователя, перезагрузка конфигурации; веб‑панель – нагрузка по пулам и серверам БД, сессии, блокировки, графики аналитики. REST и веб не входят в ядро и обращаются к базе аналитики и при необходимости к ядру.

Разработанное программное средство может применяться в организациях, где PostgreSQL – основная или вспомогательная СУБД (дата-центры, облачные провайдеры, корпоративные информационные системы). Типичные сценарии: централизованный пул соединений для множества приложений с визуализацией нагрузки и возможностью завершать долгие запросы и блокировать пользователей без консоли сервера; аналитика по подключениям и запросам для отчётности и аудита; развёртывание в Docker вместе с кластером PostgreSQL и веб-интерфейсом в едином сценарии.

Таким образом, PgPooler предлагает интегрированный подход к эксплуатации PostgreSQL: сочетание управления пулом соединений и маршрутизации по правилам конфигурации с наблюдаемостью и административным управлением в единой системе, что снижает трудозатраты сопровождения и ускоряет реакцию на эксплуатационные ситуации. Многопроцессная схема с диспетчером и воркерами позволяет масштабировать ядро прокси по ядрам процессора при росте числа одновременных соединений.

Список использованных источников:

1. Моргунов, Е. П. PostgreSQL. Основы языка SQL : учеб.-практ. пособие / Е. П. Моргунов. – СПб. : БХВ-Петербург, 2018. – 336 с.

2. Obe, R. PostgreSQL : Up and Running / R. Obe, L. Hsu. – 3rd ed. – Sebastopol : O'Reilly Media, 2017. – 314 p.

3. Schönig, H.-J. Mastering PostgreSQL 15 : Advanced techniques to build and manage scalable, reliable, and fault-tolerant database applications / H.-J. Schönig. – Birmingham : Packt Publishing, 2023. – 522 p. – ISBN 978-1-80324-834-9.

4. Riggs, S. PostgreSQL 14 Administration Cookbook : Over 175 proven recipes for database administrators to manage enterprise databases effectively / S. Riggs, G. Ciolli. – Birmingham : Packt Publishing, 2022. – 608 p. – ISBN 978-1-80324-897-4.