Введение

В 2022 году ни одна сфера жизни не обходится без компьютеризации. Организации используют базы и СУБД для перевода данных в электронный вид. Необходимость перевода заключается не столько в потребности сократить временные и материальные (кадровые) расходы, сколько в поддержании конкурентоспособности. Переход компании в электронный вид дает возможность приобретения принципиально новых качеств, позволяющих иметь существенные преимущества над другими.

Из-за высокой популярности СУБД возникает вопрос об оптимизации ее работы. Так как один из самых распространенных способов увеличения производительности — параллельное выполнение, следует рассмотреть оптимизацию многопоточной программы.

Многопоточность

С ростом объема БД наблюдается преимущество по времени многопоточной реализации [1]. Так, при работе с базой данных, состоящей из 100.000 записей, время выполнения запросов примерно в 1000 раз выше у однопоточной программы.

Тем не менее, многопоточная реализация имеет ряд недостатков. Так, приложения, использующие PostgreSQL, вынуждены открывать новое соединение в каждом потоке. Поскольку операция подключения — одна из самых дорогостоящих (процесс подключения к БД занимает от 2 до 3 МБ [2]), рост количества потоков может привести к замедлению работы программы: повышенная нагрузка на системные ресурсы и значительное снижение производительности, особенно в многоядерных системах.

Соединение в PostgreSQL

PostgreSQL содержит инструменты для реализации многопоточности. Один из них — библиотека libpq. Однако при многопоточности существует ограничение: «Два потока не должны пытаться одновременно работать с

одним объектом PGconn. В частности, не допускается параллельное выполнение команд из разных потоков через один объект соединения» [3].

Один из подходов сокращения числа подключений к БД — использование пула соединений. В PostgreSQL отсутствует встроенный пул [3], но допускается использование внешнего. Внешний пул может быть разработан с помощью средств библиотеки libpq или подключен в качестве дополнительной службы (PGBouncer или PGPool). Преимущество использование пула — увеличение пропускной способности транзакции до 60% [4]. Однако каждая из реализаций имеет ряд ограничений, которые не позволяют достичь требуемой эффективности в многопоточной программе.

Задача разрабатываемого метода заключается в сокращении числа открытых соединений путем реализации возможности параллельной работы из разных потоков с одним объектом соединения.

Метод параллельного выполнения запросов в рамках одного соединения

Разрабатываемый метод состоит из двух этапов обработки запроса: отправка запроса серверу, подразумевающий формирование очереди запросов и отправку запроса из очереди, и получение результата от сервера.

Работа функции отправки запроса начинается с блокировки мьютекса. Блокировка необходима для защиты параметров соединения от изменений другими потоками, т.к. результат выполнения напрямую зависит от состояния параметров.

После того как была произведена блокировка, выполняется проверка корректности самих параметров и инициализация соответствующих полей. В том числе, определяется точка входа очереди команд. И, после указания протокола запросов, команда может быть добавлена в очередь.

Ожидание поступления результата от сервера реализовано с помощью бесконечного цикла, на каждой итерации которого выполняется проверка состояния сервера. Пока сервер готов вернуть данные, выполняется их чтение и запись во входной буфер. Как только состояние сервера меняется на

«свободен», выполняется обработка полученных данных. Обработка включает в себя чтение типа сообщения, его длины, и непосредственно сообщение результата.

Перед выходом из функции выполняется проверка корректности полученных данных и разблокировка мьютекса для обработки следующего запроса.

Исследование временной эффективности

Было проведено исследование временной эффективности метода параллельного выполнения простых запросов в пределах одного соединения без нагрузки и с нагрузкой БД. Под простым запросом подразумевается выборка всех данных посредством одного оператора ‘SELECT’ и одного оператора ‘FROM’.

Выполнение простого запроса без нагрузки БД

В данном эксперименте выполнялось сравнение времени выполнения простого запроса для 4 реализаций: последовательная, параллельная, внешний пул на основе библиотеки libpq и разработанный метод.

Для каждого опыта учитывалось время создания соединения, выполнения запроса, очистки результата выполнения и закрытия соединения. В случае многопоточности, время создания и ожидания синхронизации потоков также влияло на итоговый результат.

Согласно результатам (рисунок 1), наибольшее время выполнения было зафиксировано у однопоточной реализации: 0.42 секунды при 500 потоках, что в 1.7 раз превосходит многопоточную реализацию.

Время работы модуля с использованием внешнего пула и с использованием разработанного метода разнится незначительно в данном эксперименте. Но стоит отметить, что разработанный метод в 4.55 раза работает быстрее однопоточной реализации при большом количестве создаваемых соединений.

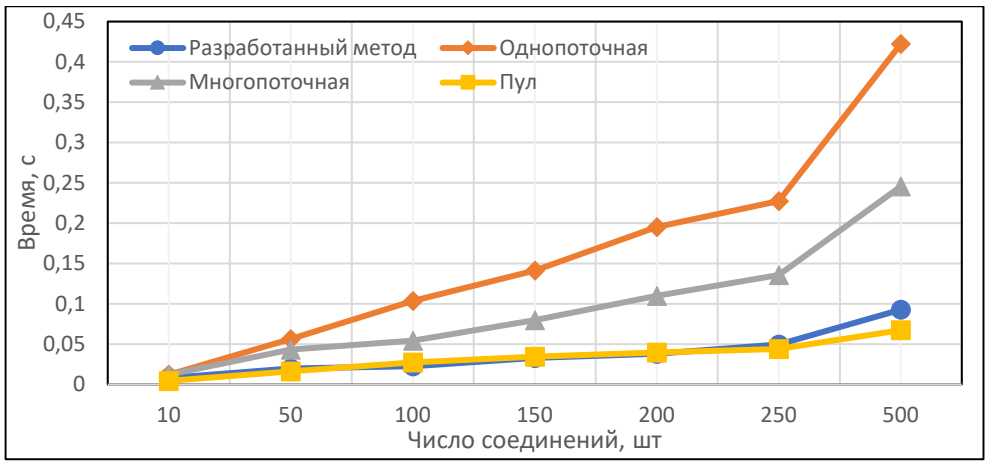

Рисунок 1 — График зависимости времени выполнения запроса от количества создаваемых соединений для каждой реализации.

Сравнение разработанного метода с пулом соединений

На основе результатов опыта, описанного выше, было установлено, что конкурентоспособность по времени разработанному методу составляет только пул соединений.

В данном эксперименте проводилось сравнение времени работы пула, использующего библиотеку libpq и пула, реализованного в качестве внешней службы (PGBouncer), с разработанным методом.

Согласно результатам эксперимента (рисунок 2), при более 100 соединений PGBouncer теряет временную эффективность: в 2.74 раза работает дольше, чем внешний пул. Однако при 10 соединениях работает быстрее разработанного метода в 1.24 раза.

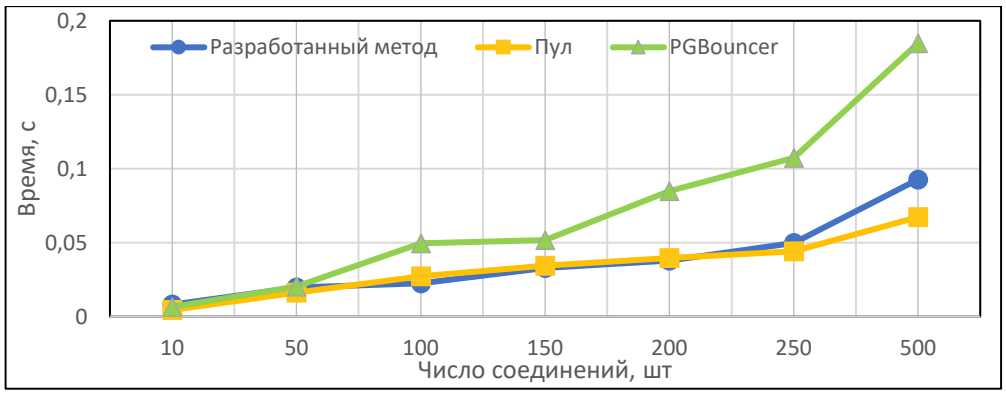

Рисунок 2 — Сравнение времени работы метода и пула соединений.

Выполнение простого запроса с нагрузкой БД

В данном эксперименте выполнялось сравнение времени выполнения простого запроса для разработанного метода и пула соединений при нагрузке на сервер БД. Для нагрузочного тестирования использовался инструмент Apache Jmeter.

Для нагрузки сервера были созданы пул соединений размером 200 и 150 потоков (пользователей), каждый из которых выполнял один запрос к БД. Запрос подразумевал декартовое произведение (CROSS JOIN) таблиц размером 200 и 500 строк. Таким образом, результат состоял из 100.000 строк.

Результаты эксперимента приведены на рисунке 3. Следует отметить, что время работы реализованного метода сильно разнилось для каждого опыта. На графике представлены средние значения для 30 опытов.

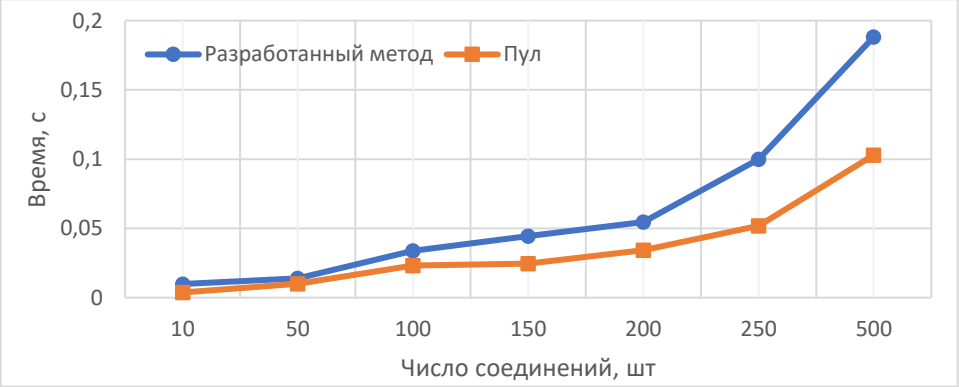

Рисунок 3 — График зависимости времени работы реализаций в зависимости от количества соединений при нагрузке БД.

Согласно графику, разработанный метод уступает по времени реализации, основанной на внешнем пуле соединении. Так, при 500 соединениях пул работает в 1.83 раза быстрее. Однако при 50 соединениях разница минимальна и составляет 3.79 мс.

Анализ требуемых ресурсов

Помимо временного анализа, было выполнено исследование затрат памяти для каждой реализации в случае создания 10 соединений и выполнения простого запроса. Результаты эксперимента представлены в таблице 1.

Таблица 1 — Затраты памяти каждой из реализаций.

Реализация	Число раз выделения памяти	Суммарный объем используемой памяти
Однопоточная	729	588,870 байт
Многопоточная	812	593,508 байт
Внешний пул	831	586,212 байт
Разработанный метод	182	180,794 байт

Согласно результатам, наибольшее потребление памяти (593 байта) у многопоточной реализации, что в 3.28 раза больше, чем память, потребляемая разработанным методом. Также, метод требует в 3.24 раза меньше памяти, чем внешний пул.

Сильное отклонение реализуемого метода от среднего значения объясняется тем, что что при создании соединения, серверный процесс *postmaster* создает новый процесс для обслуживания данного клиента. Поскольку метод использует только 1 открытое соединение, то суммарный объем потребляемой памяти меньше, чем в других реализациях.

Вывод

Таким образом, использование той или иной реализации зависит от условия задачи. Применение метода оправдано в тех задачах, где объем предоставляемой памяти ограничен, а скорость выполнения запросов не критична. В случае, когда в задаче важно время выполнения, следует использовать внешний пул. Однако при использовании следует учитывать затраты на его разработку, тестирование, конфигурацию и встраиваемость в код. В то время, как использование разработанного метода подразумевает вызов одной функции.

Разработанный метод имеет перспективу дальнейшего развития. Так, может быть выполнена реализация пользовательского вывода информации об ошибках в случае конкатенации нескольких запросов в одну команду. Другим направлением развития может стать разработка метода управления ресурсным пулом в случае потери потоком соединения с базой данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Пан К. С., Цымблер М. Л. Разработка параллельной СУБД на основе по-следо-вательной СУБД PostgreSQL с открытым исходным кодом // Вестник ЮУрГУ. Сер. Математическое моделирование и программирование. – 2012. – №18. – С. 277. Режим доступа: https://cyberleninka.ru/article/n/razrabotka-parallelnoy-subd-na-osnove-posledovatelnoy-subd-postgresql-s-otkrytym-ishodnym-kodom.
2. Воронова, Н. М. Алгоритмы оценки производительности модуля работы с данными / Н. М. Воронова, А. С. Кованова, Н. С. Корж // Инновации. Наука. Образование. – 2021. – № 38. – С. 647-658.
3. PostgreSQL: Документация: 14.2. [Электронный ресурс]. Режим доступа: https://postgrespro.ru/docs/postgresql/14.
4. Шиндов Д. А. Разработка пула соединений для работы с СУБД MYSQL на языке программирования C++ // ББК 1 А28. – 2021. – С. 112-113.