Авторы: Егор Рогов, Павел Лузанов

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

Профилирование как инструмент для поиска узких мест

Выбор подзадачи для профилирования

Средства построения профиля

Инструмент

Профилирование

- выделение подзадач
- продолжительность
- количество выполнений

Что оптимизировать?

- чем больше доля подзадачи в общем времени выполнения, тем больше потенциальный выигрыш
- необходимо учитывать затраты на оптимизацию
- полезно взглянуть на задачу шире

В предыдущих темах мы разобрались с тем, как работают запросы, из каких «кирпичиков» строится план выполнения и что влияет на выбор того или иного плана. Это самое сложное и важное. Поняв механизмы, с помощью логики и здравого смысла можно разобраться в любой возникшей ситуации и понять, эффективно ли выполняется запрос и что можно сделать, чтобы улучшить показатели.

Но как найти тот запрос, который имеет смысл оптимизировать?

В принципе, решение любой задачи оптимизации (не только в контексте СУБД) начинается с профилирования, хоть этот термин и не всегда употребляется явно. Мы должны разбить задачу, вызывающую нарекания, на подзадачи и измерить, какую часть общего времени они занимают. Также полезна информация о числе выполнения каждой из подзадач.

Чем больше доля подзадачи в общем времени, тем больше выигрыш от оптимизации именно этой подзадачи. На практике приходится учитывать и ожидаемые затраты на оптимизацию: получить потенциальный выигрыш может оказаться нелегко.

Часта ситуация, при которой подзадача выполняется быстро, но часто (например, характерно для запросов, генерируемых ORM-ами). Может оказаться невозможным ускорить выполнение отдельных запросов, но стоит задаться вопросом: должна ли подзадача выполняться *так* часто? Это может привести к необходимости изменения архитектуры, что всегда непросто, но в итоге может дать существенный выигрыш.

Нюансы

Что профилировать

отдельную задачу, вызывающую нарекания

чем точнее, тем лучше: широкий охват «размывает» проблему

Единицы измерения

время —
осмысленность для пользователя

прочитанные или записанные страницы —
стабильность по отношению ко внешним факторам

Лучше всего строить профиль для одной конкретной задачи так, чтобы измерения затрагивали только действия, необходимые для выполнения именно этой задачи. Если, например, пользователь жалуется на то, что «окно открывается целую минуту», бессмысленно смотреть на всю активность в базе данных за эту минуту: в эти цифры попадут действия, не имеющие никакого отношения к открытию окна.

(С другой стороны, профиль общей активности может многое сказать администратору, который занят не решением конкретной проблемы, а поиском наиболее ресурсоемких задач, оптимизация которых, вероятно, снизит нагрузку на систему.)

В каких единицах измерять ресурсы? Самая осмысленная для конечного пользователя характеристика — это время отклика: сколько прошло времени «от нажатия на кнопку» до «получения результата».

Однако с технической точки зрения смотреть на время не всегда удобно. Оно сильно зависит от массы внешних факторов: от наполненности кэша, от текущей загруженности сервера. Если проблема решается не на продуктивном, а на другом (тестовом) сервере с иными характеристиками, то добавляется и разница в аппаратуре, в настройках, в профиле нагрузки.

В этом смысле может оказаться удобнее смотреть, например, на число прочитанных и записанных страниц. Этот показатель более стабилен и как правило отражает объем работы при выполнении запроса, поскольку основное время уходит на чтение и обработку страниц с данными. Хотя — повторимся — такой показатель не имеет смысла для конечного пользователя.

Профиль приложения

Подзадачи

- клиентская часть
- сервер приложений
- сервер баз данных ← *проблема часто, но не всегда, именно здесь*
- сеть

Как профилировать

- технически трудно, нужны разнообразные средства мониторинга
- подтвердить предположение обычно не сложно

Для пользователя имеет смысл время отклика. Это означает, что в профиль, вообще говоря, должна входить не только СУБД, но и клиентская часть, и сервер приложений, и передача данных по сети.

Часто проблемы с производительностью кроются именно в СУБД, поскольку один неадекватно построенный план запроса может увеличивать время на порядки. Но это не всегда так. Проблема может оказаться в том, что у клиента медленное соединение с сервером, что клиентская часть долго отрисовывает полученные данные и т. п.

К сожалению, получить такой полный профиль достаточно трудно. Для этого все компоненты информационной системы должны быть снабжены подсистемами мониторинга и трассировки, учитывающими особенности именно этой системы. Но обычно несложно измерить общее время отклика (хотя бы и секундомером) и сравнить с общим временем работы СУБД; убедиться в отсутствии существенных сетевых задержек и т. п. Если же этого не проделать, то вполне может оказаться, что мы будем искать ключи там, где светлее, а не там, где их потеряли. Собственно, весь смысл профилирования и состоит в том, чтобы выяснить, что именно нуждается в оптимизации.

Далее мы будем считать установленным, что проблема именно в СУБД.

Профиль PL/pgSQL

Расширение PL Profiler

- стороннее расширение
- предназначено только для функций на PL/pgSQL
- профилирование отдельного скрипта или сеанса
- отчет о работе в html, включая flame graph

При выполнении пользователем действия, выполняется обычно не один, а несколько запросов. Как определить тот запрос, который имеет смысл оптимизировать? Для этого нужен профиль, детализированный до запросов.

Однако серверная часть приложения, выполняющаяся в СУБД, может состоять не только из SQL-запросов, но и содержать процедурный код. Если код написан на языке PL/pgSQL, то для его профилирования можно воспользоваться сторонним расширением PL Profiler (основной автор – Ян Вик, компания BigSQL: https://www.openscg.com/bigsql/docs/plprofiler/).

Расширение позволяет профилировать как отдельно выполняемые скрипты, так и снимать профиль работающего сеанса. Результат работы формируется как html-файл, содержащий необходимую информацию. В том числе он включает изображение вызовов в виде «языков пламени» (flame graph).

Поскольку курс посвящен оптимизации запросов SQL, мы не будем рассматривать это расширение, но упоминаем его для полноты картины.

Профиль запросов

Журнал сообщений сервера

включается конфигурационными параметрами:

log_min_duration_statements = 0 — время и текст всех запросов

log_line_prefix — идентифицирующая информация

сложно включить для отдельного сеанса

большой объем; при увеличении порога времени теряем информацию

не отслеживаются вложенные запросы
(можно использовать расширение auto_explain)

анализ внешними средствами, такими, как pgBadger

Для получения профиля по выполняемым SQL-запросам есть два основных средства, встроенных в PostgreSQL: журнал сообщений сервера и статистика.

В журнал сообщений с помощью конфигурационных параметров можно включить вывод информации о запросах и времени их выполнения. Обычно для этого используется параметр *log_min_duration_statement,* хотя есть и другие.

Как локализовать в журнале те запросы, которые относятся к действиям пользователя? Имело бы смысл включать и выключать параметр для конкретного сеанса, но штатных средств для этого не предусмотрено. Можно фильтровать общий поток сообщений, выделяя только нужные; для этого удобно в параметр *log_line_prefix* вывести дополнительную идентифицирующую информацию. Использование пула соединений еще больше усложняет ситуацию.

Для анализа вложенных запросов (когда в запросе вызывается функция, содержащая другие запросы) потребуется модуль auto_explain (https://postgrespro.ru/docs/postgrespro/10/auto-explain).

Следующая проблема — анализ журнала. Это осуществляется внешними средствами, из которых стандартом де-факто является расширение pgBadger (https://github.com/darold/pgbadger).

Разумеется, можно выводить в журнал и собственные сообщения, если они могут помочь делу.

Профиль запросов

Расширение pg_stat_statements

- подробная информация о запросах в представлении (в том числе о вложенных)
- ограниченный размер хранилища
- запросы считаются одинаковыми «с точностью до констант», даже если имеют разные планы выполнения
- идентификация только по имени пользователя и базе данных

Второй способ — статистика, а именно расширение pg_stat_statements (https://postgrespro.ru/docs/postgresql/10/pgstatstatements).

Расширение собирает достаточно подробную информацию о выполняемых запросах (в том числе в терминах ввода-вывода страниц) и отображает ее в представлении pg_stat_statements.

Поскольку число различных запросов может быть очень большим, размер хранилища ограничен конфигурационным параметром; в нем остаются наиболее часто используемые запросы.

При этом «одинаковыми» считаются запросы, имеющие одинаковое (с точностью до констант) дерево разбора. Надо иметь в виду, что такие запросы могли иметь разные планы выполнения и выполняться разное время.

К сожалению, есть сложности и с идентификацией: запросы можно отнести к конкретному пользователю и базе данных, но не к сеансу.

Профиль одного запроса

EXPLAIN ANALYZE

- подзадачи — узлы плана
- продолжительность — actual time
  или страничный ввод-вывод — buffers
- количество выполнений — loops

Особенности

- помимо наиболее ресурсоемких узлов, кандидаты на оптимизацию — узлы с большой ошибкой прогноза кардинальности
- любое вмешательство может привести к полной перестройке плана
- иногда приходится довольствоваться простым explain

Так или иначе, среди выполненных запросов мы находим тот, который будем оптимизировать. Как работать с самим запросом? Тут тоже поможет профиль, который выдает команда EXPLAIN ANALYZE.

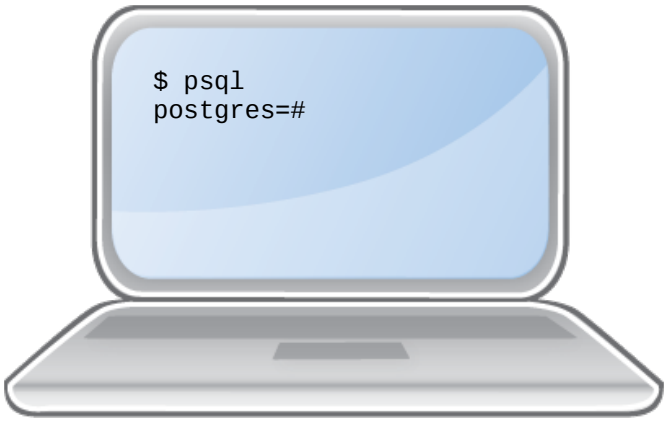
Подзадачами такого профиля являются узлы плана (надо учитывать, что это не плоский список, а дерево). Продолжительность выполнения узла покажет actual time, а число выполнений — loops. Как рассматривалось в теме «Хеш-соединение», можно получить не только время выполнения, но и объем ввода-вывода (BUFFERS).

Но здесь имеются особенности. Во-первых, кроме перечисленного, план выполнения содержит дополнительную — и крайне важную — информацию об ожидании оптимизатора относительно кардинальности каждого шага. Как правило, если нет серьезной ошибки в прогнозе кардинальности, то и план будет построен адекватный (если нет, то надо изменять глобальные настройки). Поэтому стоит обращать внимание не только на наиболее ресурсоемкие узлы, но и на узлы с большой (на порядок или выше) ошибкой. Конечно, смотреть надо на наиболее вложенный проблемный узел, поскольку ошибка будет распространяться от него выше по дереву.

Во-вторых, надо быть готовым к тому, что любое вмешательство может привести не к сокращению времени выполнения узла, а к полной перестройке плана.

Наконец, бывают ситуации, когда запрос выполняется так долго, что выполнить EXPLAIN ANALYZE не представляется возможным. В таком случае придется довольствоваться простым EXPLAIN и пытаться разобраться в причине неэффективности без полной информации.

Демонстрация



Итоги

Для поиска задач, нуждающихся в оптимизации, используется профилирование

Средства профилирования зависят от задачи:

- журнал сообщений сервера и pg_stat_statements
- EXPLAIN ANALYZE

Практика

1. Выполните первую версию отчета, показанного в демонстрации, включив вывод текста и времени выполнения запросов в журнал сообщений.
2. Посмотрите, какая информация попала в журнал сообщений.
3. Повторите предыдущие пункты, включив расширение auto_explain с выводом вложенных запросов.