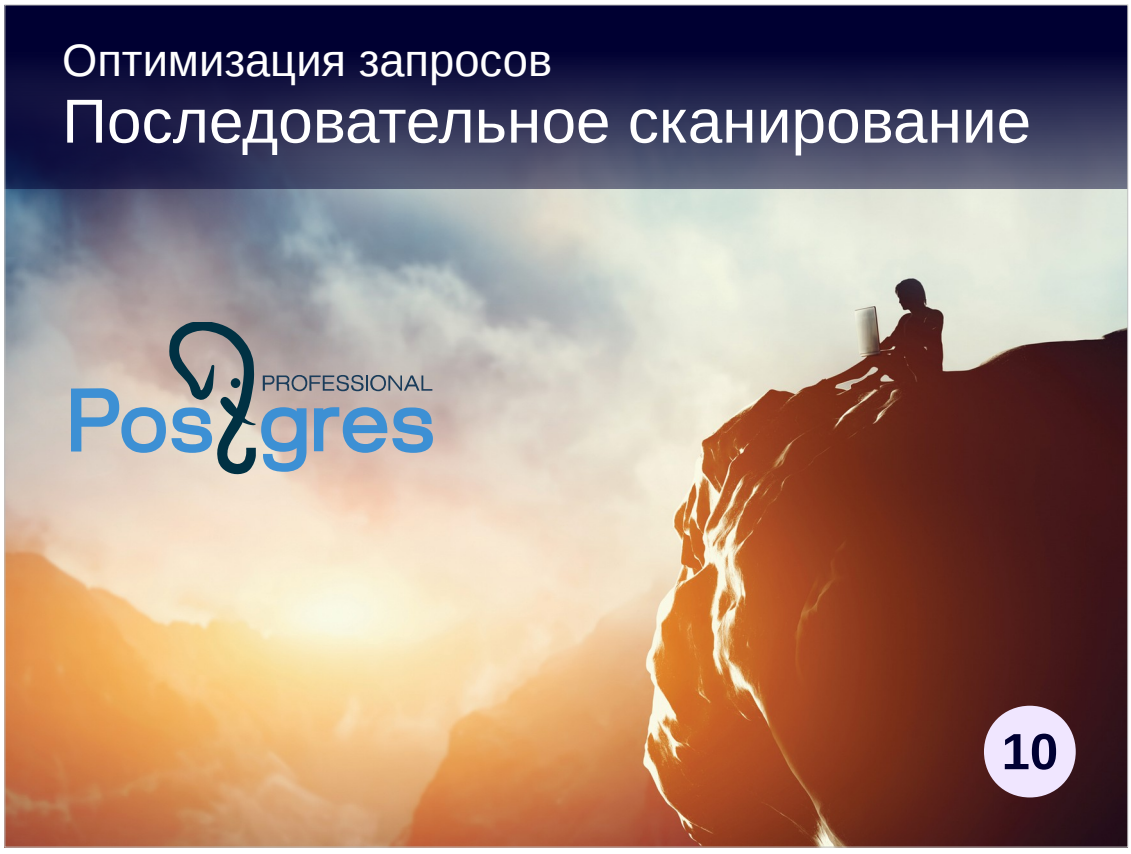


Авторы: Егор Рогов, Павел Лузанов

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

Темы

Последовательное сканирование (Seq Scan)

Параллельные планы выполнения

Параллельное сканирование (Parallel Seq Scan)

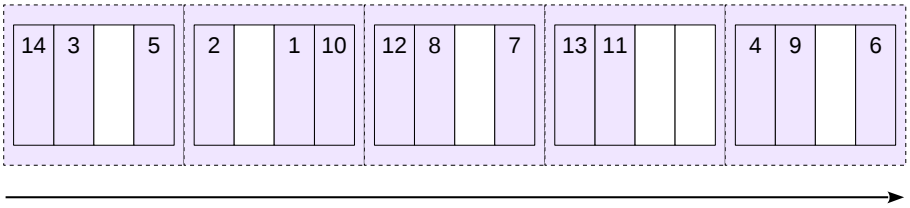
Агрегация при параллельном выполнении

Команда EXPLAIN

Seq Scan

Последовательное чтение всех страниц

- страницы читаются в кэш (используется буферное кольцо)
- проверяется видимость версий строк
- данные возвращаются в произвольном порядке
- время сканирования зависит от физического размера файла



В распоряжении оптимизатора имеется несколько способов доступа к данным. Самый простой из них — последовательное сканирование таблицы. Файл (или файлы) таблицы читается постранично от начала до конца. При этом рассматриваются все версии строк на каждой странице: удовлетворяют ли они условиям запроса и соблюдены ли правила видимости.

Напомним, что чтение происходит через буферный кэш; чтобы большая таблица не вытеснила все полезные данные, для последовательного сканирования создается «кольцо буферов» небольшого размера. При этом другие процессы, одновременно выполняющие последовательное сканирование той же таблицы, «присоединяются» к тому же кольцу и тем самым экономят дисковые чтения (если процесс присоединился не сразу, он затем отдельно дочитывает начальные страницы таблицы).

Последовательное чтение файла позволяет использовать тот факт, что операционная система обычно читает данные порциями больше, чем размер страницы: с большой вероятностью несколько следующих страниц уже окажутся в кэше ОС.

Последовательное сканирование эффективно работает, когда надо прочитать всю таблицу или значительную ее часть (если селективность условия низка). Если же из всей таблицы нужна только небольшая часть записей, более предпочтительными являются методы доступа, использующие индекс.

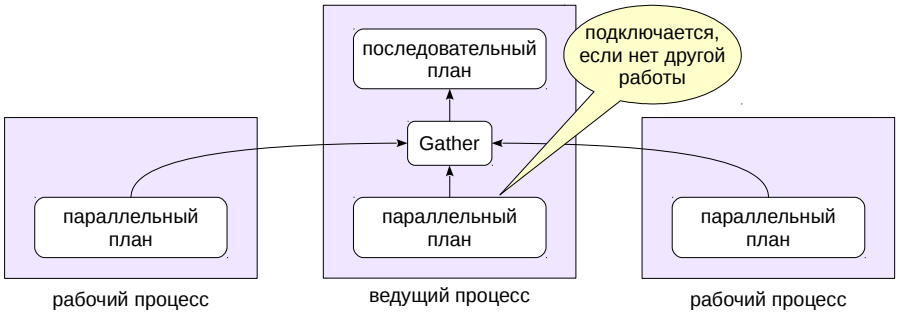
Параллельные планы

Ведущий процесс

- выполняет последовательную часть плана
- запускает рабочие процессы и получает от них данные

Рабочие процессы

- одновременно работают над параллельной частью плана



Начиная с версии 9.6 PostgreSQL поддерживает параллельное выполнение запросов. Идея состоит в том, что ведущий процесс, выполняющий запрос, порождает (с помощью postmaster, конечно) несколько рабочих процессов, которые одновременно выполняют одну и ту же «параллельную» часть плана. Результаты этого выполнения передаются ведущему процессу, который собирет их в узле Gather.

Если рабочие процессы не успевают загрузить ведущего данными, ведущий процесс тоже подключается к выполнению того же самого параллельного плана.

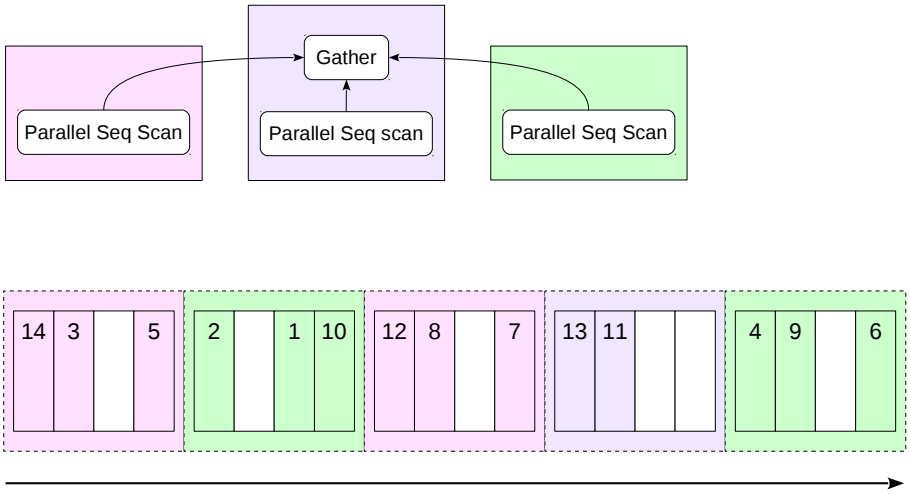
Разумеется, запуск процессов и пересылка данных требуют определенных ресурсов, поэтому далеко не каждый запрос выполняется параллельно.

Кроме того, даже при параллельном выполнении не все шаги плана запроса могут быть распараллелены. Часть операций может выполняться ведущим процессом в одиночку, последовательно.

https://postgrespro.ru/docs/postgresql/10/parallel-query

Заметим, что в PostgreSQL нет другого теоретически возможного режима распараллеливания, при котором несколько процессов составляют конвейер для обработки данных (грубо говоря, отдельные узлы плана выполняются отдельными процессами). Разработчики PostgreSQL сочли такой режим неэффективным.

Parallel Seq Scan



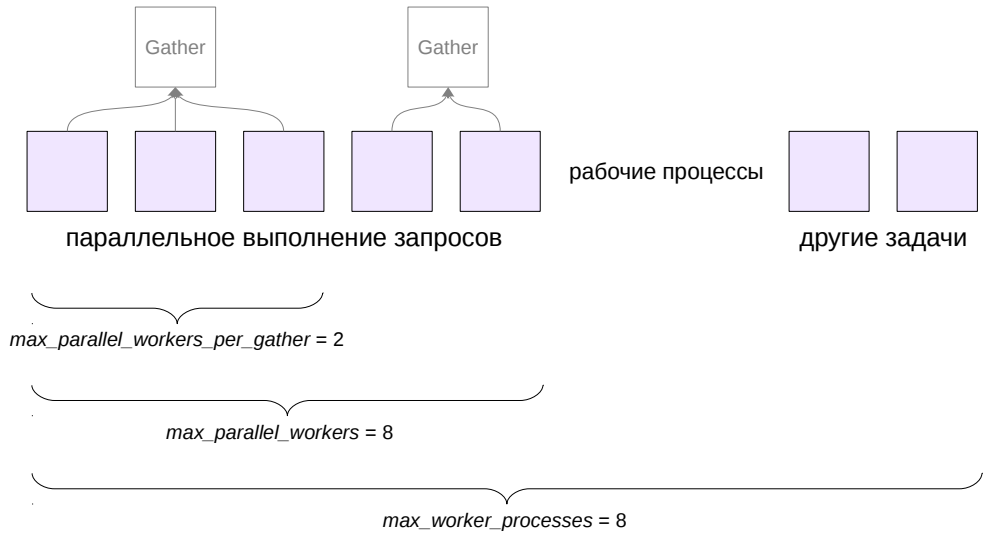
страницы читаются последовательно, но разными процессами

Примером параллельного выполнения является Parallel Seq Scan — «параллельное последовательное сканирование».

Название звучит противоречиво, но, тем не менее, отражает суть операции. Страницы таблицы *читаются последовательно,* в том же самом порядке, в котором они читались бы при обычном последовательном сканировании. Однако чтение выполняется несколькими *параллельно* работающими процессами. Процессы синхронизируются между собой, чтобы запросы на чтение шли в правильном порядке.

Сама по себе такая операция не имеет большого смысла, поскольку к обычным затратам на чтение страниц добавляются накладные расходы на пересылку данных от процесса к процессу. Но если рабочие процессы выполняют какую-то обработку прочитанных строк (например, агрегацию), то суммарное выполнение запроса может занять существунно меньше времени. Такой пример будет рассмотрен в демонстрации.

Общие настройки



Параллельным выполнением управляет довольно много параметров. Сначала рассмотрим те, что ограничивают число рабочих процессов.

Вообще механизм рабочих процессов используется не только для параллельного выполнения запросов. Например, рабочие процессы задействованы в логической репликации, ими могут пользоваться расширения. Общее число одновременно выполняющихся рабочих процессов ограничено параметром *max_worker_processes* (по умолчанию 8).

Число одновременно выполняющихся рабочих процессов, занимающихся параллельными планами, ограничено параметром *max_parallel_workers* (по умолчанию тоже 8).

Число одновременно выполняющихся рабочих процессов, обслуживающих один ведущий процесс, ограничено параметром *max_parallel_workers_per_gather* (по умолчанию 2).

Значения этих параметров следует изменить в зависимости от возможностей аппаратуры, объема данных и загруженности системы. Например, даже если в базе данных есть большие таблицы и запросы могли бы выиграть от распараллеливания, но при этом в системе нет свободных ядер, то параллельное выполнение не будет иметь смысла.

Число рабочих процессов

Равно нулю (параллельный план не строится)

если *размер таблицы* < *min_parallel_table_scan_size* = 8MB

Фиксировано

если для таблицы указан параметр хранения *parallel_workers*

Вычисляется по формуле

$1 + \lfloor \log_3(\textit{размер таблицы} / \textit{min\_parallel\_table\_scan\_size}) \rfloor$

но не больше, чем *max_parallel_workers_per_gather*

Сколько рабочих процессов будет использоваться?

Планировщик вообще не будет рассматривать параллельное сканирование, если физический размер таблицы меньше значения параметра *min_parallel_table_scan_size*.

Обычно число процессов вычисляется по формуле, приведенной на слайде. Она означает, что чем больше таблица, тем больше будет создаваться параллельных процессов, но очередное изменение числа процессов происходит при увеличении таблицы в три раза. Например, для стандартного значения *min_parallel_table_scan_size* = 8MB:

таблица	процессы	таблица	процессы
8MB	1	216MB	4
24MB	2	648MB	5
72MB	3	1.9GB	6

Число процессов можно и явно указать в параметре хранения *parallel_workers* таблицы.

При этом число процессов в любом случае не будет превышать значения параметра *max_parallel_workers_per_gather*. Если при выполнении запроса доступное число процессов окажется меньше запланированного, будут использоваться только доступные (вплоть до последовательного выполнения, если пул полностью исчерпан).

Не распараллеливаются

Запросы на запись

- а также запросы с блокировкой строк

Курсоры

- в том числе запросы в цикле FOR в PL/pgSQL

Запросы с функциями PARALLEL UNSAFE

Запросы в функциях,
вызванных из распараллеленного запроса

Запросы при уровне изоляции SERIALIZABLE

Не каждый запрос может выполняться в параллельном режиме.

Не распараллеливаются запросы, изменяющие или блокирующие данные (UPDATE, DELETE, SELECT FOR UPDATE и т. п.).

Не распараллеливаются запросы, выполнение которых может быть приостановлено — это относится к запросам в курсорах, в том числе в циклах FOR PL/pgSQL.

Не распараллеливаются запросы, содержащие функции, помеченные как PARALLEL UNSAFE (все основные стандартные функции безопасны; список небезопасных можно получить из таблицы pg_proc по условию proparallel = 'u').

Не распараллеливаются запросы, содержащиеся в функциях, которые вызываются из распараллеленного запроса (чтобы не допустить рекурсивного разрастания).

Не поддерживается уровень изоляции SERIALIZABLE.

Часть из этих ограничений может быть снята в следующих версиях PostgreSQL.

Только последовательно

Чтение результатов общих табличных выражений (CTE)

Чтение результатов нераскрываемых подзапросов

Обращения к временным таблицам

Вызовы функций PARALLEL RESTRICTED

Функции, использующие вложенные транзакции

В целом, чем большую часть плана удается выполнить параллельно, тем больший возможен эффект. Однако есть ряд операций, которые в целом не препятствуют распараллеливанию, но сами могут выполняться только последовательно в ведущем процессе.

К ним относятся:

- чтение результатов общих табличных выражений (подзапросов в предложении WITH);

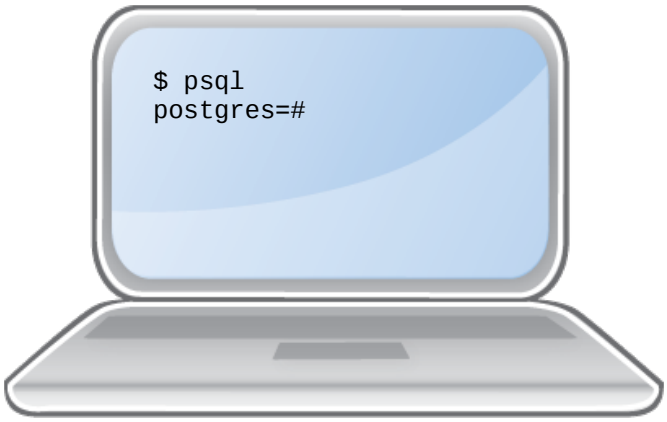
- чтение результатов других нераскрываемых подзапросов (которые представляются в плане узлами InitPlan или SubPlan);

- обращения ко временным таблицам (так как они доступны только ведущему процессу);

- вызовы функций, помеченных как PARALLEL RESTRICTED (список можно получить из таблицы pg_proc по условию proparallel = 'r').

Если в запросе вызывается функция, использующая вложенные транзакции (например, функции на PL/pgSQL с обработкой исключений), запрос завершится с ошибкой. Такие функции должны быть помечены как PARALLEL RESTRICTED.

Демонстрация



Итоги

Последовательное сканирование читает всю таблицу

- эффективно при низкой селективности

Параллельное выполнение в ряде случаев позволяет ускорить запрос

Практика

1. Убедитесь, что запрос, вычисляющий среднюю стоимость перелета, выполняется параллельно.
2. Проверьте, как выполняется тот же запрос, если чтение из таблицы перелетов оформлено как общее табличное выражение.
3. В демонстрации запрос всех строк из таблицы рейсов выполнялся последовательно. Почему? Может ли такой запрос в принципе быть распараллелен или он запрещает параллельное выполнение?
   Воспользуйтесь параметром *force_parallel_mode*, чтобы ответить на этот вопрос.