Инструкция по настройке производительности PostgreSQL

Конфигурирование postgresql.conf

Пример добавленных конфигурационных строк, в файл postgresql.conf программой pg_tune (т.е. предварительный, далее модернизируем):

Параметр	Значение	Примечание
default_statistics_target	50	задаёт объём статистики по умолчанию
maintenance_work_mem	960MB	память для работы команды VACUUM
constraint_exclusion	on	оптимизация запросов при работе с партиционированием
checkpoint_completion_target	0.9	отношение периода инициализации крайней checkpoint до конца переноса данных на диск к началу следующей checkpoint
effective_cache_size	11GB	оценка объёма кэша ОС
work_mem	96MB	память для сортировки результата запроса
wal_buffers	8MB	определяет размер буфера журнала транзакций
checkpoint_segments	16	количество сегментов журнала транзакций
shared_buffers	3840MB	общий буфер сервера
max_connections	80	количество подключений

Используемая память

shared_buffers общий буфер сервера. PostgreSQL не читает данные напрямую с диска и не пишет их сразу на диск. Данные загружаются в общий буфер сервера, находящийся в разделяемой памяти, серверные процессы читают и пишут блоки в этом буфере, а затем уже изменения сбрасываются на диск. Если процессу нужен доступ к таблице, то он сначала ищет нужные блоки в общем буфере. Если блоки присутствуют, то он может продолжать работу, если нет — делается системный вызов для их загрузки. Загружаться блоки могут как из файлового кэша ОС, так и с диска, и эта операция может оказаться весьма «дорогой». Если объём буфера недостаточен для хранения часто используемых рабочих данных, то они будут постоянно писаться и читаться из кэша ОС или с диска, что крайне отрицательно скажется на производительности. Объём задаётся параметром shared_buffers в файле postgresql.conf. Единица измерения параметра — блоки величиной 8 кБ. По умолчанию значение параметра составляет 64 , что соответствует 512 кБ памяти. Это весьма мало, и для полноценной работы значение параметра следует увеличить. В то же время не следует устанавливать это значение слишком большим: PostgreSQL полагается на то, что операционная система кэширует файлы, и не старается дублировать эту работу. Кроме того, чем больше памяти будет отдано под буфер, тем меньше останется операционной системе и другим приложениям, что может привести к свопингу. Рекомендуемый параметр для БД большого объёма данных и 1-4 ГБ доступной памяти: 64-256 МБ (8192- 32768)

constraint_exclusion оптимизация запросов при работе с партиционированием. Не желательно для баз, где нет партиционирования, поскольку команда CHECK будет проверяться на всех запросах, даже простых, а значит,

производительность сильно упадет. Значения могут быть on, off и partition. По умолчанию и рекомендуется ставить partition, который будет проверять CHECK только на партиционированных таблицах.

checkpoint_completion_target отношение периода инициализации крайней checkpoint до конца переноса данных на диск к началу следующей checkpoint, т.е. если checkpoint_segments = 10, а checkpoint_completion_target = 0.5, то к моменту создания 5-го сегмента лога с момента инициализации checkpoint, запись на диск должна завершиться, для checkpoint_timeout = 300, а checkpoint_completion_target = 0.5, к 150-ой секунде с момента инициализации checkpoint, запись на диск должна завершиться. По умолчанию checkpoint_completion_target = 0.5.

work_mem память для сортировки результата запроса. Этот параметр определяет объём памяти, которую процесс может использовать для сортировки результата запроса. Такой объём может быть использован каждым процессом для каждой сортировки (в сложных запросах их может быть несколько). Если объём памяти недостаточен для сортировки некоторого результата, то серверный процесс будет использовать временные файлы. Если же объём памяти слишком велик, то это может привести к свопингу. Объём памяти задаётся параметром work_mem в файле postgresql.conf. Единица измерения параметра — 1 кБ. Значение по умолчанию — 1024. В качестве начального значения для параметра можно взять 2-4% доступной памяти.

maintenance_work_mem память для работы команды VACUUM. Этот параметр задаёт объём памяти, используемый командой VACUUM. Обычно эта команда больше нагружает диски, но увеличение vacuum_mem позволит ускорить процесс за счёт хранения в памяти больших объёмов информации об удалённых записях. Объём памяти задаётся параметром vacuum_mem в файле postgresql.conf. Единица измерения параметра — 1 кБ. Значение по умолчанию — 8192. Этот параметр может также быть задан для отдельного соединения. Можно сделать его поменьше для частых регулярных запусков VACUUM и большим для ежедневных/еженедельных запусков VACUUM FULL.

Журнал транзакций и контрольные точки. Журнал транзакций PostgreSQL работает следующим образом: все изменения в файлах данных (в которых находятся таблицы и индексы) производятся только после того, как они были занесены в журнал транзакций, при этом записи в журнале должны быть гарантированно записаны на диск. В этом случае нет необходимости сбрасывать на диск изменения данных при каждом успешном завершении транзакции: в случае сбоя БД может быть восстановлена по записям в журнале. Таким образом, данные из буферов сбрасываются на диск при проходе контрольной точки: либо при заполнении нескольких (параметр checkp oint_segments, по умолчанию 3) сегментов журнала транзакций, либо через определённый интервал времени (параметр checkpoint_timeout, измеряется в секундах, по умолчанию 300). Изменение этих параметров прямо не повлияет на скорость чтения, но может принести большую пользу, если данные в базе активно изменяются.

fsync и стоит ли его трогать. Наиболее радикальное из возможных решений — выставить значение no параметру f sync. При этом записи в журнале транзакций не будут принудительно сбрасываться на диск, что даст большой прирост скорости записи. Но в случае сбоя целостность базы будет нарушена, и её придётся восстанавливать из резервной копии!

checkpoint_segments количество сегментов журнала транзакций. Если в базу заносятся большие объёмы данных, то контрольные точки могут происходить слишком часто. При этом производительность упадёт из-за постоянного сбрасывания на диск данных из буфера. Для увеличения интервала между контрольными точками нужно увеличить количество сегментов журнала транзакций. Каждый сегмент занимает 16 МБ, так что на диске будет занято дополнительное место. Обычно на диске будет не менее одного и не более 2*checkpoint_segments+1 сегментов журнала. Следует также отметить, что чем больше интервал между контрольными точками, тем дольше будут восстанавливаться данные по журналу транзакций после сбоя.

Прочие параметры

wal_sync_method определяет метод, при помощи которого записи в журнале транзакций принудительно сбрасываются на диск. Значение по умолчанию зависит от платформы. Возможно, изменение этого параметра позволит увеличить производительность.

wal_buers (в блоках по 8 кБ, 8 по умолчанию) определяет размер буфера журнала транзакций, в котором накапливаются записи перед сбросом их на диск. Стоит увеличить буфер до 256-512 кБ, что позволит лучше работать с большими транзакциями.

commit_delay (в микросекундах, 0 по умолчанию) и commit_siblings (5 по умолчанию) определяют задержку между попаданием записи в буфер журнала транзакций и сбросом её на диск. Если при успешном завершении транзакции активно не менее commit_siblings транзакций, то запись будет задержана на время commit_delay. Если за это время завершится другая транзакция, то их изменения будут сброшены на диск вместе, при помощи одного системного вызова. Эти параметры позволят ускорить работу, если параллельно выполняется много «мелких» транзакций.

effective_cache_size оценка объёма кэша ОС. Этот параметр сообщает PostgreSQL примерный объём файлового кэша операционной системы, оптимизатор использует эту оценку для построения плана запроса. Объём задаётся параметром effective_cache_size в postgresql.conf. По умолчанию значение параметра составляет 128МБ. Пусть в компьютере 1,5 ГБ памяти, параметр shared_buffers установлен в 32 МБ, а параметр effective_cache_size в 800 МБ. Если запросу нужно 700 МБ данных, то PostgreSQL оценит, что все нужные данные уже есть в памяти и выберет более агрессивный план с использованием индексов и merge joins. Но если effective_cache_size будет всего 200МБ, то оптимизатор вполне может выбрать более эффективный для дисковой системы план, включающий полный просмотр таблицы. В качестве начального значения можно использовать 25-50% доступной памяти.

synchronous_commit
full_page_writes
effective_io_concurrency
max_prepared_transactions
max_locks_per_transaction
wal_sync_method

Сбор статистики

default_statistics_target задаёт объём по умолчанию статистики (по умолчанию 100), собираемой командой ANALYZE. Увеличение параметра заставит эту команду работать дольше, но может позволить оптимизатору строить более быстрые планы, используя полученные дополнительные данные. Объём статистики для конкретного поля может быть задан командой ALTER TABLE SET STATISTICS. У PostgreSQL также есть специальная подсистема — сборщик статистики, — которая в реальном времени собирает данные об активности сервера. Эта подсистема контролируется следующими параметрами, принимающими значения:

track_activities передавать ли сборщику статистики информацию о текущей выполняемой команде и времени начала её выполнения. По умолчанию on. Следует отметить, что эта информация будет доступна только привилегированным пользователям и пользователям, от лица которых запущены команды, так что проблем с безопасностью быть не должно.

track_counts собирать ли информацию об активности на уровне записей и блоков. По умолчанию on. Данные, полученные сборщиком статистики, доступны через специальные системные представления. Полученные данные позволят оптимизировать использование индексов.

Перенос журнала транзакций на отдельный диск

При доступе к диску изрядное время занимает не только собственно чтение данных, но и перемещение магнитной головки. Если в вашем сервере есть несколько физических дисков, то вы можете разнести файлы базы данных и журнал транзакций по разным дискам. Данные в сегменты журнала пишутся последовательно, более того, записи в журнале транзакций сразу сбрасываются на диск, поэтому в случае нахождения его на отдельном диске магнитная головка не будет лишний раз двигаться, что позволит ускорить запись. Порядок действий: Остановите сервер. Перенесите каталог pg_xlog, находящийся в каталоге с базами данных, на другой диск. Создайте на старом месте символическую ссылку. Запустите сервер. Примерно таким же образом можно перенести и часть файлов, содержащих таблицы и индексы, на другой диск, но здесь потребуется больше кропотливой ручной работы, а при внесении изменений в схему базы процедуру, возможно, придётся повторить.

Поддержание базы в порядке

VACUUM используется для «сборки мусора» в базе данных. Начиная с версии 7.2, существует в двух вариантах: VACUUM FULL пытается удалить все старые версии записей и, соответственно, уменьшить размер файла, содержащего таблицу. Этот вариант команды полностью блокирует обрабатываемую таблицу. VACUUM помечает место, занимаемое старыми версиями записей, как свободное. Использование этого варианта команды, как правило, не уменьшает размер файла, содержащего таблицу, но позволяет не дать ему бесконтрольно расти, зафиксировав на некотором приемлемом уровне. При работе VACUUM возможен параллельный доступ к обрабатываемой таблице. При использовании в форме VACUUM [FULL] ANALYZE, после сборки мусора будет обновлена статистика по данной таблице, используемая оптимизатором. В абсолютном большинстве случаев имеет смысл использовать именно эту форму. Рекомендуется достаточно частое — например, раз в несколько минут — выполнение VACUUM ANALYZE для часто обновляемых баз (или отдельных таблиц). В обыкновенных случаях достаточно ежедневного выполнения этой команды. При этом обратите внимание: если «бутылочное горлышко» сервера находится в районе дисковой подсистемы, то выполнение VACUUM параллельно с обычной работой может крайне отрицательно сказаться на производительности. Команду VACUUM FULL стоит запускать достаточно редко, не чаще раза в неделю. Её также имеет смысл запускать вручную для конкретной таблицы после удаления или обновления большой части записей в ней.

ANALYZE Служит для обновления информации о распределении данных в таблице. Эта информация используется оптимизатором для выбора наиболее быстрого плана выполнения запроса. Обычно команда используется в связке VACUUM ANALYZE. Если в базе есть таблицы, данные в которых не изменяются и не удаляются, а лишь добавляются, то для таких таблиц можно использовать отдельную команду ANALYZE. Также стоит использовать эту команду для отдельной таблицы после добавления в неё большого количества записей.

pg_autovacuum - программа, которая отслеживает изменения в таблицах и автоматически запускает команды VACUUM и/или ANALYZE для этих таблиц по достижении определённого предела. Использование этой программы позволяет отказаться от настройки периодического выполнения команд VACUUM и ANALYZE. Более того, в случае использования pg_autovacuum ресурсы не тратятся впустую на обработку таблиц, которые практически не подвергались изменениям. Для работы pg_autovacuum должен быть включён сборщик статистики и включён параметр track_counts.

REINDEX Команда REINDEX используется для перестройки существующих индексов. Использовать её имеет смысл в случае порчи индекса; постоянного увеличения его размера. Второй случай требует пояснений. Индекс, как и таблица, содержит блоки со старыми версиями записей. PostgreSQL не всегда может заново использовать эти блоки, и поэтому файл с индексом постепенно увеличивается в размерах. Если данные в таблице часто меняются, то расти он может весьма быстро. Если вы заметили подобное поведение какого-то индекса, то стоит настроить для него периодическое выполнение команды REINDEX. Команда REINDEX, как и VACUUM FULL, полностью блокирует таблицу, поэтому выполнять её надо тогда, когда загрузка сервера минимальна.

EXPLAIN [ANALYZE]. Команда EXPLAIN [запрос] показывает, каким образом PostgreSQL собирается выполнять запрос. Команда EXPLAIN ANALYZE [запрос] выполняет запрос и показывает как изначальный план, так и реальный процесс его выполнения. Использование полного просмотра таблицы (seq scan). Использование наиболее примитивного способа объединения таблиц (nested loop). Для EXPLAIN ANALYZE: нет ли больших отличий в предполагаемом количестве записей и реально выбранном? Если оптимизатор использует устаревшую статистику, то он может выбирать не самый быстрый план выполнения запроса. Следует отметить, что полный просмотр таблицы далеко не всегда медленнее просмотра по индексу. Если, например, в таблице-справочнике несколько сотен записей, умещающихся в одном-двух блоках на диске, то использование индекса приведёт лишь к тому, что придётся читать ещё и пару лишних блоков индекса. Если в запросе придётся выбрать 80% записей из большой таблицы, то полный просмотр опять же получится быстрее. При тестировании запросов с использованием EXPLAIN ANALYZE можно воспользоваться настройками, запрещающими оптимизатору использовать определённые планы выполнения. Например,

```
SET enable_seqscan=false;
```

запретит использование полного просмотра таблицы, и вы сможете выяснить, прав ли был оптимизатор, отказываясь от использования индекса. Ни в коем случае не следует прописывать подобные команды в postgresql.conf! Это может ускорить выполнение нескольких запросов, но сильно замедлит все остальные!

Использование индексов

Наиболее значительные проблемы с производительностью вызываются отсутствием нужных индексов. Поэтому столкнувшись с медленным запросом, в первую очередь проверьте, существуют ли индексы, которые он может использовать. Если нет — постройте их. Излишек индексов, впрочем, тоже чреват проблемами: Команды, изменяющие данные в таблице, должны изменить также и индексы. Очевидно, чем больше индексов построено для таблицы, тем медленнее это будет происходить. Оптимизатор перебирает возможные пути выполнения запросов. Если построено много ненужных индексов, то этот перебор будет идти дольше. Единственное, что можно сказать с большой степенью определённости — поля, являющиеся внешними ключами, и поля, по которым объединяются таблицы, индексировать надо обязательно.

Использование собранной статистики

Результаты работы сборщика статистики доступны через специальные системные представления. Наиболее интересны для наших целей следующие:

pg_stat_user_tables содержит — для каждой пользовательской таблицы в текущей базе данных — общее количество полных просмотров и просмотров с использованием индексов, общие количества записей, которые были возвращены в результате обоих типов просмотра, а также общие количества вставленных, изменённых и удалённых записей.

pg_stat_user_indexes содержит — для каждого пользовательского индекса в текущей базе данных — общее количество просмотров, использовавших этот индекс, количество прочитанных записей, количество успешно прочитанных записей в таблице (может быть меньше предыдущего значения, если в индексе есть записи, указывающие на устаревшие записи в таблице).

pg_statio_user_tables содержит — для каждой пользовательской таблицы в текущей базе данных — общее количество блоков, прочитанных из таблицы, количество блоков, оказавшихся при этом в буфере, а также аналогичную статистику для всех индексов по таблице и, возможно, по связанной с ней таблицей TOAST. Из этих представлений можно узнать, в частности. Для каких таблиц стоит создать новые индексы (индикатором служит большое количество полных просмотров и большое количество прочитанных блоков). Какие индексы вообще не используются в запросах. Их имеет смысл удалить, если, конечно, речь не идёт об индексах, обеспечивающих выполнение ограничений PRIMARY KEY и UNIQUE. Достаточен ли объём буфера сервера. Также возможен «дедуктивный» подход, при котором сначала создаётся большое количество индексов, а затем неиспользуемые индексы удаляются.

Возможности индексов

Можно построить индекс не только по полю/нескольким полям таблицы, но и по выражению, зависящему от полей. Пусть, например, в таблице foo есть поле foo_name, и выборки часто делаются по условию «первая буква foo_name = ’буква’, в любом регистре». Можно создать индекс

```
CREATE INDEX foo_name_first_idx ON foo ((lower(substr(foo_name, 1, 1))));
```

и запрос вида

```
SELECT * FROM foo WHERE lower(substr(foo_name, 1, 1)) = 'ы';
```

будет его использовать.

Частичные индексы (partial indexes) Под частичным индексом понимается индекс с предикатом WHERE. Пусть, например, у вас есть в базе таблица scheta с параметром uplocheno типа boolean. Записей, где uplocheno = false меньше, чем записей с uplocheno = true, а запросы по ним выполняются значительно чаще.

Можно создать индекс

```
CREATE INDEX scheta_neuplocheno ON scheta (id) WHERE NOT uplocheno;
```

который будет использоваться запросом вида

```
SELECT * FROM scheta WHERE NOT uplocheno AND ...;
```

Достоинство подхода в том, что записи, не удовлетворяющие условию WHERE, просто не попадут в индекс.

Оптимизация запроса

для которого по разным причинам нельзя заставить оптимизатор использовать индексы, и которые будут всегда вызывать полный просмотр таблицы. Таким образом, если вам требуется использовать эти запросы в требовательном к быстродействию приложении, то придётся их изменить.

```
SELECT max(...)/min(...)FROM <огромная таблица>.
```

Все агрегатные функции в PostgreSQL реализованы одинаково: сначала выбираются все записи, удовлетворяющие условию, а потом к полученному набору записей применяется агрегатная функция. У такого подхода есть достоинства — вы можете легко написать собственную агрегатную функцию — но есть и недостаток, который заключается в том, что для работы функций типа min() / max() весь набор записей совершенно не нужен. Для их работы рациональней было бы воспользоваться индексом по полю, для которого ищется максимум (минимум), но для этого придётся сделать реализацию этих агрегатных функций отличной ото всех остальных.

Проблема запрос вида

```
SELECT max(field) FROM foo;
```

не будет использовать существующий индекс по полю field, а будет делать полный просмотр таблицы. Если записей в таблице много, то это может занять изрядное время.

Решение запрос вида

```
SELECT field FROM foo WHERE field IS NOT NULL ORDER BY field DESC LIMIT 1;
```

вернёт то же самое значение, но при этом сможет использовать индекс по field, если таковой существует.

Конфигурирование postgresql.conf исходя из вышеописанного

Пример добавленных конфигурационных строк, в файл postgresql.conf программой pg_tune, рекомендую модернизировать следующим образом:

Параметр	Значение	Примечание
default_statistics_target	1000	для лучшей оптимизации нужно больше статистики, увеличиваем объем
maintenance_work_mem	2GB	увеличиваем для хранения в памяти больших объёмов информации об удалённых записях
constraint_exclusion	partition	неоправданно тратить время на обязательную проверку CHECK (партиционирование в нашей БД не применяется), ставим по умолчанию (на случай если партиции будут присутствовать в БД)
checkpoint_completion_target	0.9	0.9 выставляют для лучшей производительности, оставляем
effective_io_concurrency	2	равно количеству шпинделей в дисках
effective_cache_size	10GB	не будем ориентироваться только на него, немного уменьшаем
work_mem	512MB	результаты запросов достаточно объемные, увеличиваем значение
wal_buffers	8MB	для буфера журнала транзакций достаточно, оставляем
checkpoint_segments	16	оставляем
shared_buffers	1GB	это не вся необходимая память, уменьшаем
max_prepared_transactions	100	max кол-во PREPARED TRANSACTION которые установятся в состояние prepared, минимум = max_connections
max_locks_per_transaction	200	число объектов бд которые могут быть заблокированы
max_connections	100	вернем по умолчанию
commit_delay	200	задержим запись
commit_siblings	3	при трех активных транзакциях
synchronous_commit	off	производительность важнее чем точность в проведении транзакций
full_page_writes	on	отключение параметра ускоряет работу, но может привести к повреждению БД в случае сбоя