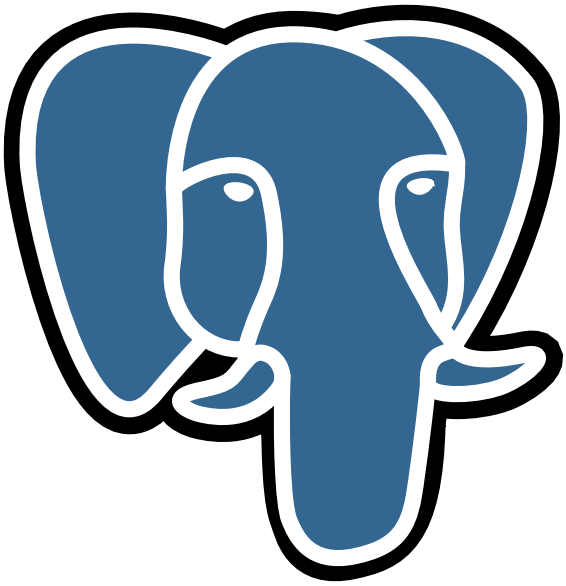

История о PostgreSQL

© Е.М. Балдин*

Эта статья была опубликована в мартовском номере русскоязычного журнала Linux Format (`http://www.linuxformat.ru`) за 2007 год. Статья размещена с разрешения редакции журнала на сайте `http://www.inp.nsk.su/~baldin/` и до конца августа месяца все вопросы с размещением статьи в других местах следует решать с редакцией Linux Format. Затем все права на текст возвращаются ко мне.

Текст, представленный здесь, не является точной копией статьи в журнале. Текущий текст в отличии от журнального варианта корректор не просматривал. Все вопросы по содержанию, а так же замечания и предложения следует задавать мне по электронной почте `mailto:E.M.Baldin@inp.nsk.su`.

Текст на текущий момент является просто *текстом*, а не книгой. Поэтому результирующая доводка в целях улучшения восприятия текста не проводилась.

*e-mail: E.M.Baldin@inp.nsk.su

Слон взят с сайта `http://pgfoundry.org/projects/graphics/`. Изображение предоставляется под лицензией BSD.

Оглавление

Глава 5

Настройка PostgreSQL

> — Между прочим, — сказал Роман громко, — уже в течение двух минут я пытаюсь его пассивизировать, и совершенно безрезультатно
>
> «Понедельник начинается в субботу» АБС

«Тюнинг» — это не операция — это образ жизни. Очевидно, что когда необходимые характеристики можно улучшить несколькими способами, то всеми способами их и надо улучшать. Но опять же следует помнить, что избыточная и ранняя оптимизация — корень многих зол. Если система работает и «не кашляет», то может быть не стоит её «настраивать»?

5.1 О железе

«Театр начинается с вешалки», а большая база данных начинается с выбора сервера. PostgreSQL умудряется работать почти при любой конфигурации, но если Вас интересует результат, то следует знать на что обращать внимание. Очевидно, что рамки на железо диктуются исключительно объёмом денежных ресурсов, но в имеющихся пределах всегда можно что-то подвигать.

Как и для всякой программы, работающей с большим объёмом данных, дисковая подсистема является для PostgreSQL определяющей. Поэтому если есть возможность, то лучше вложиться именно в неё. В противоположность дисковой подсистеме процессор нагружается не очень сильно. Поэтому для сервера достаточно, чтобы процессор просто был, хотя лучше чтобы он был не один. Если денег на покупку хорошего SCSI диска нет, то следует вложиться в память.

К вопросу о дисках можно сказать, что чем их больше — тем лучше. По возможности следует выделить отдельный диск для журнала транзакций (`pg_xlog`). Избыток дисков так же позволит собрать из них RAID. Хоть никто и не отменяет бэкапа, но дополнительная избыточность для дисковой подсистемы, как и источники бесперебойного питания, сэкономят массу сил и нервов.

Относительно недорогие системы снабжены дешёвыми встроенными RAID-контроллерами на четыре диска. Не следует использовать эти контроллеры, а лучше настроить софтверный RAID и не надо использовать RAID 5[1] для небольшого числа дисков. До 6 дисков включительно лучше RAID 1+0[2]. Избыточность во всём — это только похоже на паранойю.

На сервер, где уже работает PostgreSQL не следует «подселять» другие базы данных или программы, осуществляющие интенсивный обмен с дисковой памятью. А вот программы, которые вместо этого интенсивно используют процессор, например, Apache, вполне уживаются если памяти достаточно.

Собственно говоря, можно особо не стараться. PostgreSQL вполне себе работает и на обычном пользовательском компьютере. Более того пришло время серьёзных баз данных на десктопе. Существует куча приложений, которые занимаются индексированием и каталогизацией, при этом создавая свои доморощенные «базки». А ведь решение очевидно и PostgreSQL вполне может стать им. Да и самим данным вовсе не помешает оказаться в нормальной базе специально для этих данных созданной. Это «толстый» такой намёк, так как хорошее хранилище данных для Вашей, ещё не написанной программы, на дороге не валяется.

5.2 Конфигурационные файлы

Настройка конфигурационных файлов не единственный способ настройки сервера базы данных. Умолчания можно изменить непосредственно при сборке из исходников. Значения можно передавать непосредственно серверу `postmaster` в командной строке, используя ключик `-c`. Так же можно определить переменную окружения `PGOPTIONS`, а значения некоторых переменных можно изменить прямо в процессе выполнения запросов.

Концентрация внимания на конфигурационных файлах объясняется тем, что в любом случае их придётся настраивать. По умолчанию PostgreSQL сконфигурирован в расчёте на минимальное потребление ресурсов, и это не может не сказаться на скорости. Что подходит для локальной записной книжки — не годится для боевого сервера.

Все настройки очень подробно описаны в документации. Для любителей «выжимать из программы всё» существует специальный список рассылки psql-performance: `http://archives.postgresql.org/pgsql-performance/`. В документации на странице Power PostgreSQL `http://www.powerpostgresql.com/Docs` также можно найти некоторое количество полезных подсказок.

Для настроек PostgreSQL используются файлы:

pg_hba.conf — политика доступа и идентификации пользователей,

postgresql.conf — собственно говоря, настройки сервера.

[1] Один диск в массиве выделяется под контрольные суммы

[2] зеркалирование (1) + объединение (0)

5.2.1 pg_hba.conf

Часто для локальных нужд при использовании PostgreSQL в качестве своей личной записной книжки, склада данных или даже «помойки» не требуется открывать сетевой доступ к базе данных. По умолчанию PostgreSQL настроен так, что каждый локальный пользователь может подсоединиться к базе совпадающей по названию с регистрационным именем клиента, при условии что такая база данных уже создана.

Но это не значит, что так бывает всегда. PostgreSQL предоставляет свои механизмы для управления пользователями с помощью тройки команд CREATE USER, DROP USER и ALTER USER. В случае не совпадающих регистрационных имен в PostgreSQL и в системе или при доступе к базе данных с других компьютеров необходимо «обговорить» правила доступа к данным на сервере.

Для настройки политики доступа к серверу волей-неволей придётся заглянуть в файл `pg_hba.conf`[3]. Тело файла представляют из себя однострочные записи, каждая из которых регулирует правила получения доступа для конкретной машины или для целой группы IP. Файл `pg_hba.conf` выглядит примерно следующим образом:

```
# Разрешаем доступ через локальные unix-сокеты абсолютно
#всем пользователям к базам данным, совпадающим по названию
#с регистрационными именами
local   all             all                                     ident sameuser
# Доверяем пользователю alex с указанного IP безгранично
#в рамках базы данных photos
host    photos          alex    130.255.204.48/32   trust
# Требуем пароль от пользователя baldin при доступе с
#компьютеров из сети 128.138.242.192/27 к базам данных
#data и photos
hostssl data,photos  baldin 128.138.242.192/27  md5
```

В файле по умолчанию присутствует подробная информация о формате записей, первое поле которых определяет тип записи:

- local — эта запись определяет политику для локального доступа через локальные UNIX-сокеты.
- host — эта запись определяет политику для сетевого TCP/IP соединения и годится для соединений с использованием SSL и без него. Для того чтобы можно было доступаться к базе данных по сети необходимо правильно настроить listen_addresses в `postgresql.conf`.
- hostssl — определяется политика для сетевого соединения с обязательным использованием SSL.
- hostnossl — антипод hostssl.

[3] hba — host-based authentication.

Второе поле представляет из себя имя базы данных для которой определяется политика. Имя **all** зарезервировано для всех баз данных, а имя **sameuser** для базы данных, совпадающей с именем пользователя. Имена баз данных можно перечислять через запятую. Так же в качестве имени можно добавить имя файла со списком баз, разделённых запятой или пробелами. Для этого к имени файла следует добавить символ «@» в качестве префикса.

Третье поле — имя пользователя. Как и в случае имён баз данных можно работать со списками. Имя **all** зарезервировано для всех пользователей. Так же доступ можно открыть для группы пользователей (role), для этого перед именем группы следует поставить знак «+».

Следующие ноль (в случае записи local), одно (нотация CIDR[4]) или два поля (адрес и сетевая маска) занимает сетевой адрес компьютера или подсети для которого настраивается политика доступа.

Предпоследнее обязательное поле отведено под метод авторизации:

- trust — полностью доверяем этому клиенту.
- reject — отказ в доступе.
- ident — доступ по регистрационной записи клиента. Часто применяется для локальных соединений. RFC 1413.
- md5 — авторизация по паролю зашифрованному с помощью алгоритма md5.

  Если клиент использует библиотеку для доступа к PostgreSQL версии старше 7.2, то вместо метода md5 следует использовать метод crypt.

  Пароль при желании и значительной степени бесшабашности можно передавать открытым текстом с помощью метода password.
- pam — авторизация с помощью **P**luggable **A**uthentication **M**odules. Этот сервис предоставляется операционной системой.

  Подробнее о PAM написано здесь: `http://www.kernel.org/pub/linux/libs/pam/`.
- krb4 и krb5 — авторизация с использованием механизма Kerberos версии 4 и 5, соответственно. Это индустриальный стандарт авторизации. То есть, кому надо — тот знает что это такое.

После метода ему можно передать опции в последнем необязательном поле.

5.2.2 postgresql.conf

Настройки в `postgresql.conf` разбиты по группам и подробно задокументированы прямо в файле. Здесь будут описаны не все настройки, но важнейшие из них будут отмечены.

[4] Classless Inter-Domain Routing

Настройка соединений и авторизация (Connections and Authentication)

Политика авторизации настраивается в `pg_hba.conf`. Здесь же собраны в основном технические параметры.

- Настройка соединений (connection settings).

  listen_addresses После настройки `pg_hba.conf` можно и нужно смело устанавливать `*` — слушаем все интерфейсы, которые есть в наличии. По умолчанию (localhost) запросы принимаются только от локальных пользователей в том числе и через интерфейс обратной связи (loopback-интерфейс `127.0.0.1`).

  port Номер порта, который слушает сервер в ожидании соединений. По умолчанию он равен 5432.

  max_connections Число клиентов, которые могут подсоединяться к базе данных одновременно не может быть бесконечным. Каждое подсоединение порождает ещё один процесс postmaster, что, естественно, требует ресурсов. Средней «паршивости» современный однопроцессорный компьютер со стандартным наполнении без особых проблем может обслуживать 100-200 соединений, но, например, 600 активных соединений будут уже явной проблемой.

  Любая попытка подсоединиться сверх указанного лимита приведёт к отказу от обслуживания. Плохо написанная программа в цикле открывающая, но не закрывающая за собой соединения, легко создаст проблему.

  Если число клиентов жёстко ограничено, то имеет смысл уменьшить этот параметр до минимально возможного значения.

  superuser_reserved_connections Число соединений, которые зарезервированы для суперпользователя, чтобы он мог всегда зайти, разобраться в чём дело, а затем принять меры. Не стоит совсем отказываться зарезервированных соединений, причём одного зарезервированного соединения может оказаться не достаточно — 2 это минимум.

- Безопасность и авторизация (security and authentication)

  authentication_timeout Время ожидания для прохождения авторизации в секундах. По умолчанию это время равно одной минуте. Не позволяет клиенту «зависнуть» и заблокировать ресурс соединения на очень долгое время.

  ssl Разрешается доступ через ssl. Для работы через SSL необходимо создать публичный ключ и сертификат. Это требует некоторых усилий, зато позволяет немного успокоить себя на тему безопасности сетевых соединений.

Управление ресурсами (Resource Consumption)

Правильная оценка имеющихся ресурсов — путь к эффективном планированию. А эффективное планирование позволяет в процессе работы не сильно увеличивать окружающую нас энтропию и в то же время добиваться поставленной цели. Очевидные истины.

- Память (memory)

  shared_buffers Объём совместно используемой памяти, выделяемой PostgreSQL для кэширования данных, определяется числом страниц (shared_buffers) по 8 килобайт каждая. Естественно, данные умеет кэшировать не только сам PostgreSQL, но и операционная система сама по себе делает это очень неплохо. Поэтому нет необходимости отводить под кэш всю наличную оперативную память. Оптимальное число shared_buffers зависит от многих факторов, поэтому проще для начала принять следующие ориентиры:

  – Обычный настольный компьютер с 512 Мб и небольшой базой данных — 8–16 Мб или 1000–2000 страниц.
  – Не сильно выдающийся сервер предназначенный для обслуживания базы данных с объёмом оперативной памяти 1 Гб и базой данных около 10 Гб— 80–160 Мб или 10000–20000 страниц.
  – Сервер посерьёзнее с несколькими процессорами на борту, с объёмом памяти в 8 Гб и базой данных занимающей свыше 100 Гб обслуживающий несколько сотен активных соединений одновременно — 400 Мб или 50000 страниц.

  work_mem Под каждый запрос можно выделить личный ограниченный объём памяти для работы. Этот объём может использоваться для сортировки, объединения и других подобных операций. При превышении этого объёма сервер начинает использовать временные файлы на диске, что может существенно замедлить скорость обработки запросов. Предел для work_mem можно вычислить, разделив объём доступной памяти (физическая память минус объём занятый под другие программы и под совместно используемые страницы shared_buffers) на максимальное число одновременно используемых активных соединений.

  При необходимости, например, выполнения очень объёмных операций, допустимый лимит можно изменять прямо во время выполнения запроса. Поэтому нет нужды изначально задавать теоретический предел.

  maintenance_work_mem Эта память используется для выполнения операций по сбору статистики (ANALYZE), сборке мусора (VACUUM), создания индексов (CREATE INDEX) и добавления внешних ключей. Размер выделяемой под эти операции памяти должен быть сравним с физическим размером самого большого индекса на диске. Как и в случае

work_mem эта переменная может быть установлена прямо во время выполнения запроса.

max_prepared_transactions Определяет максимальное число подготовленных транзакций (команда PREPARE TRANSACTION). Подготовленные транзакции выполняются, но результат их не будет доступен пока их не подтвердят (COMMIT). Так же можно такие транзакции и отклонить (ROLLBACK). Если эта сущность нигде не используется, то переменную можно занулить.

- Карта неиспользованного пространства (free space map)

  При удалении записи не удаляются физически, а только помечаются как удалённые. Именно таким образом мусор и собирается.

  max_fsm_pages Для целей сборки мусора следует знать где этот мусор находится. Число страниц отведённых под эту задачу должно быть больше, чем число удалённых или изменённых записей между сборками мусора (VACUUM). Если страниц достаточно выполнение жёстких оптимизирующих операций, таких как VACUUM FULL или REINDEX никогда и не понадобится. Так как объём требуемой для этого памяти не очень большой (по 6 байт на страницу), то жадничать не стоит. Проще всего узнать необходимое число max_fsm_pages запустив, VACUUM VERBOSE ANALYZE.

  max_fsm_relations Число таблиц для которых создаются карты неиспользованного пространства. По умолчанию это число равно 1000. В случае большего числа используемых таблиц это значение можно и нужно увеличить, тем более что на каждую таблицу требуется всего по семь байт.

- Системные ресурсы (kernel resource usage)

  preload_libraries Если для исполнения запроса требуется загрузить какую-либо разделяемую библиотеку, то действует правило: «загружаем при первом использовании», что замедляет исполнение самого первого такого запроса. Это можно обойти, загрузив необходимые библиотеке при старте сервера, то есть воспользоваться формулой: память в обмен на скорость. Таким образом можно подгрузить разделяемую библиотеку для используемых в запросах процедурных языков.

- Оценка стоимости сборки мусора (cost-based vacuum delay)

  Немного подробнее про сборку мусора будет рассказываться далее. Обычно нет необходимости заглядывать в этот раздел, так как сборка мусора (VACUUM) и анализ (ANALYZE) выполняются достаточно быстро.

- Запись в фоне (background writer)

Начиная с версии PostgreSQL 8.0 вместе с основным сервером стартует процесс для записи данных в фоне. При выполнении запроса нет необходимости ждать самого акта записи, так как это гарантировано сделает background writer.

Журнал транзакций (Write Ahead Log)

Наличие журнала транзакций или WAL (write ahead log) позволяет увеличить скорость выполнения операций требующих изменения данных в следствии того, что в журнал информация об изменениях пишется последовательно, а изменения в самих таблицах могут быть отложены до «лучших времён» — своеобразный кэш только на диске. Если же база данных используется в основном для чтения, то в журнале транзакций нет особой необходимости, но это не повод от него отказываться.

- Настройки (settings)

  fsync По умолчанию эта опция включена (true). В этом случае PostgreSQL пытается записать данные на диск физически. Это на самом деле не такая уж и простая операции, так как кэши существуют не только в системе, но и в контроллерах и в дисках. Вполне можно представить себе такую ситуацию, что слишком умный диск для увеличения своей производительности в тестах рапортует о том, что данные записаны, а при перебое с электричеством выясняется, что это была просто шутка. Сбои самого сервера не приводят к порчи данных, но сервер живёт в окружении далеко не идеальной операционной системе, которая в свою очередь управляет далеко не идеальными физическими устройствами. Так что проверенное железо, источники бесперебойного питания, бэкап, бэкап и ещё раз бэкап.

  Если Вы доверяете своему железу, то эту опцию можно выключить. Здесь можно поменять немного безопасности на скорость. Хотя более оптимальным решением является перенос журнального файла на отдельный быстрый диск. То есть прикупить скорость за деньги.

- Контрольные точки (checkpoints)

  По свершению каких-то определённых условий или истечению контрольного времени сервер гарантировано переносит данные, записанные в WAL непосредственно в таблицы, даже если очень сильно занят на других запросах.

  checkpoint_segments Объём кэша на диске. Физический объём места на диске, требуемый под кэш вычисляется по формуле $(checkpoint\_segments \times \times 2+1) \times 16$ Мб. Следует выделить столько, сколько не жалко, осознавая, что 32 сегмента займёт на диске свыше 1 Гб.

  checkpoint_timeout Время, через которое WAL очищается насильно. По умолчанию 300 секунд.

checkpoint_warning Если кэш а диске заполняется быстрее чем число секунд checkpoint_warning, то посылается предупреждение, которое будет передано в журнальный файл. Это намёк, что кэш на диске следует увеличить.

- Архивация (archiving)

archive_command Для целей создания непрерывного резервного копирования (возможность для настоящих параноиков) журнал необходимо копировать куда-то ещё. Здесь должна быть команда, которая будет использоваться системой для копирования данных. Подробности следует искать в документации в разделе «On-line backup and point-in-time recovery (PITR)».

Планирование запросов (Query Planning)

Здесь можно повлиять на логику действия планировщика. Возможно конкретно для Вашей системы значения по умолчанию не оптимальны, но менять их стоит только после серии тестов для выявления более оптимальной конфигурации под конкретную платформу и конкретные запросы. В большинстве случаев углубляться в тонкости настроек из этого раздела имеет смысл только в случае очень изощрённых запросов.

- Методология планировщика (planner method configuration)

  В этом разделе перечислены алгоритмы, которые можно использовать для извлечения данных. Для целей тестирования какие-то из них можно отключить.

- Оценочные константы (planner cost constants)

effective_cache_size PostgreSQL в своих планах опирается на кэширование файлов, осуществляемое операционной системой. Этот параметр соответствует максимальному размеру объекта, который может поместиться в системный кэш. Установка этого параметра не приводит к увеличению выделяемой памяти. Это значение используется только для оценки.

effective_cache_size можно установить в 1/3 от объёма имеющейся в наличии оперативной памяти, если вся она отдана в распоряжение PostgreSQL.

Сообщения об ошибках и событиях (Error Reporting and Logging)

Оптимизация возможно только в случае обратной связи. Сервер PostgreSQL может много чего про себя рассказать. Этот раздел настроек посвящён тому, как правильно его об этом попросить.

- Местоположение журнального файла (where to log)

log_destination Здесь можно выбрать способ записи в журнальный файл: stderr или syslog. Метод stderr хорош для тестирования, но по хорошему журналированием должна заниматься специальная служба, а это нас приводит к необходимости изучить что такое syslog.

В случае выбора метода stderr придётся определить директорию для журнального файла (log_directory), имя журнального файла (log_filename) и другие параметры (log_rotation_age, log_rotation_size, log_truncate_on_rotation) свойственный службе журналирования.

В случае выбора метода syslog настроить его с помощью syslog_facility (необходимо знать как пользоваться syslog) и syslog_iden — идентификационный префикс для сообщений получаемых от PostgreSQL.

- По какому случаю создаём запись (when to log)

  Обычно, нет необходимости записывать в дневник информацию о каждом чихе, но при серьёзном разбирательстве данные о числе чихов в секунду и их классификация могут подтолкнуть в нужном направлении.

  Каждому событию можно присвоить какой-то определённый уровень. Например, уровень PANIC означает, что плохо стало всем, а уровень WARNING сообщает просто о подозрительных, но вполне законных событиях. В порядке возрастания подробности уровни имеют примерно следующую классификацию: PANIC, FATAL, LOG, ERROR, WARNING, NOTICE, INFO, DEBUG[1-5]. Уровень можно установить в процессе выполнения запроса.

  Следует осознавать, что журнал необходим при разбирательствах, но его активное использование ведёт к деградации производительности. Обычно эта уменьшение производительности находится в пределах 5% при условии, что журнальный файл расположен на другом диске нежели журнал транзакций.

  log_min_messages Характеризует уровень подробности записей в журнальный файл.

  log_error_verbosity Характеризует степень подробности делаемых в журнал записей. Различаются три степени: TERSE, DEFAULT и VERBOSE.

  client_min_messages Характеризует уровень подробности сообщений отсылаемых клиенту.

  log_min_error_statement Характеризует уровень подробностей записей в журнальный файл создаваемых в результате исполнения SQL-запросов.

- Что именно пишем в журнал (what to log)

  В этом разделе настроек перечислены различные возможные источники записей для журнала. Можно записывать информацию о делаемых соединениях (log_connections), информацию о выполняемых запросах (log_statement), информацию о времени уходящему на выполнение запросов (log_duration) и тому подобное. С помощью переменной log_line_prefix можно настроить идентификацию каждой записи по пользователю, IP, базе данных и так далее.

Сбор статистики (Run-Time Statistics)

Есть ложь, гнусная ложь и статистика. База данных врать не научена, поэтому остаётся только статистика. Этот раздел настроек отвечает за её сбор. Пока нет необходимости в мониторировании активности базы данных, нет необходимости здесь что-то править.

Для того чтобы работала автоматическая сборка мусора опции stats_start_collector и stats_row_level должны быть включены.

Автоматическая сборка мусора (Automatic Vacuuming)

Ну мусор, ну и пусть. Места много — зачем напрягаться, да ещё автоматически? В этом есть какая-то логика, но кроме сборки мусора (VACUUM) производится ещё и анализ (ANALYZE). Периодическое выполнение команды ANALYZE необходимо для нормального функционирования планировщика. Собранная с помощью этой команды статистика позволяет значительно ускорить выполнение SQL-запросов. То есть, если не хочется настраивать автоматическую сборку мусора, то в любом случае её придётся делать только теперь в ручную.

Процесс обычной сборки мусора в PostgreSQL (VACUUM без приставки FULL) не блокирует таблиц и может выполняться в фоне, не мешая выполнению запросов. Начиная с PostgreSQL версии 8.1 процесс автоматической сборки мусора выделяется в отдельный процесс. Эта группа настроек контролирует работу этого процесса.

autovacuum Если Вы лучше чем PostgreSQL знаете когда следует производить сборку мусора, то автоматику можно выключить. Хотя лучше её просто правильно настроить. С другой стороны сборка мусора оттягивает на себя ресурсы системы и если это не допустимо, то её можно отложить на некоторое время.

При настройке службы автоматической сборки мусора и анализа следует понимать, что один из зарезервированных с помощью superuser_reserved_connections слотов может оказаться в нужный момент занят.

autovacuum_naptime Время в секундах через которое база данных проверяется на необходимость в сборке мусора. По умолчанию это происходит раз в минуту.

autovacuum_vacuum_threshold Порог на число удалённых и изменённых записей в любой таблице по превышению которого происходит сборка мусора (VACUUM). По умолчанию этот порог равен 1000 и его вполне можно уменьшить.

autovacuum_analyze_threshold Порог на число вставленных, удалённых и изменённых записей в любой таблице по превышению которого запускается процесс анализа (ANALYZE). По умолчанию это порог равен 500. Никто не запрещает сделать его поменьше.

autovacuum_vacuum_scale_factor Процент изменённых и удалённых записей по отношению к таблице по превышению которого запускается сборка мусора. Значение по умолчанию равно 0.4.

autovacuum_analyze_scale_factor То же, что и предыдущая переменная, но по отношению к анализу. Значение по молчанию равно 0.2.

Настройки для пользователя по умолчанию (Client Connection Defaults)

Настройки из этой группы задают некоторые значения по умолчанию для подсоединившегося к базе данных клиента и не влияют на производительность самого сервера. Исключением является разве что переменная statement_timeout, которая выставляет ограничение в миллисекундах на исполнение запроса. Да и то эта возможность по умолчанию не активирована. Но, если хочется настроить локаль по умолчанию, хотя это личное дело клиента, или порядок выбора объектов относящихся к различным пространствам имён, то можно что-то поправить и здесь.

Управление блокировками (Lock Management)

Очевидно, что блокировок следует избегать всячески, причём делать это надо начинать на этапе проектирования базы данных. К сожалению реальная жизнь отличается от планов.

deadlock_timeout Взаимные блокировки (deadlock) — это чрезвычайно уродливое явление, при котором вошедшие в клинч процессы ожидают освобождение ресурсов, которые сами же и захватили. PostgreSQL умеет разрешать эту проблему путём насильного прерывания одного из процессов. Проверка на deadlock это довольно длительная процедура, поэтому, прежде чем начать проверку блокировки на предмет является ли она взаимной, сервер выжидает указанное время. Значение по умолчанию равно 1000 миллисекунд. Для загруженных серверов имеет смысл это значение увеличить.

max_locks_per_transaction Вопреки своему названию это не жёсткий лимит на число блокировок, осуществляемых в пределах транзакций. Этот параметр входит в формулу устанавливающую предел на число одновременно существующих блокировок, то есть это скорее максимальное среднее:
$\text{max\_locks\_per\_transaction} \times (\text{max\_connections} + \text{max\_prepared\_transactions})$
В документации сказано, что 64 (число стоящее по умолчанию) — это исторически проверенный предел и чтобы превзойти его требуются определённые усилия.

5.3 О том, что думать тоже надо

Можно идеально настроить сервер, регулярно проводить сборку мусора, можно закупить самое дорогое железо и поставить рядом с ним дизельную электростан-

цию. Но если таблицы и отношения между ними создавались без плана, а запросы задаются криво, то проблемы гарантировано будут.

Выполнение запросов следует проверять с помощью команды EXPLAIN ANALYZE, которая по полочкам разложит как ищется, сортируется, объединяется и группируется информация, какие для этого использовались алгоритмы и какие индексы были задействованы. Для любителей картинок pgAdmin III имеет графический интерфейс к этой команде. Ни в коем случае нельзя пренебрегать индексами и по возможности следует избегать блокировок.

Настройки PostgreSQL для 1C

По адресу `http://v8.1c.ru/overview/postgres_patches_notes.htm` лежат патчи. Это отличие версии сервера поставляемого с «1С:Предприятием 8» от оригинальных исходников PostgreSQL. Патч `postgresql-1c-8.1.5.patch` несёт в себе изменения в исходном файле настройки. Перечислим их:

- Допускаются сетевые соединения:

```
-#listen_addresses = 'localhost'
+listen_addresses = '*'
```

- немного увеличен размер разделяемой памяти с 8 Мб до 28 Мб:

```
-#shared_buffers = 1000
+shared_buffers = 3500
```

- оценка размера кэша системы изменилась с 8 Мб до 80 Мб:

```
-#effective_cache_size = 1000
+effective_cache_size = 10000
```

- Включён процесс автоматической сборки мусора:

```
-#stats_row_level = off
+stats_row_level = on
-#autovacuum = off
+autovacuum = on
```

- Максимальное среднее число блокировок увеличено более чем в два раза:

```
-#max_locks_per_transaction = 64
+max_locks_per_transaction = 150
```

Мне кажется, что судя по этим изменениям есть куда оптимизировать и сам продукт и настройки к PostgreSQL.