ВЫСОКАЯ ПРОИЗВОДИТЕЛЬНОСТЬ*, АНАЛИЗ И ПРОЕКТИРОВАНИЕ СИСТЕМ*, SQL*, POSTGRESQL*, ORACLE*

История успеха «Яндекс.Почты» с PostgreSQL

TM_content 14 февраля 2017 в 00:46 40,8k

Владимир Бородин (на «Хабре» dev1ant), системный администратор группы эксплуатации систем хранения данных в «Яндекс.Почте», знакомит со сложностями миграции крупного проекта с Oracle Database на PostgreSQL. Это — расшифровка доклада с конференции HighLoad++ 2016.

Всем привет! Меня зовут Вова, сегодня я буду рассказывать про

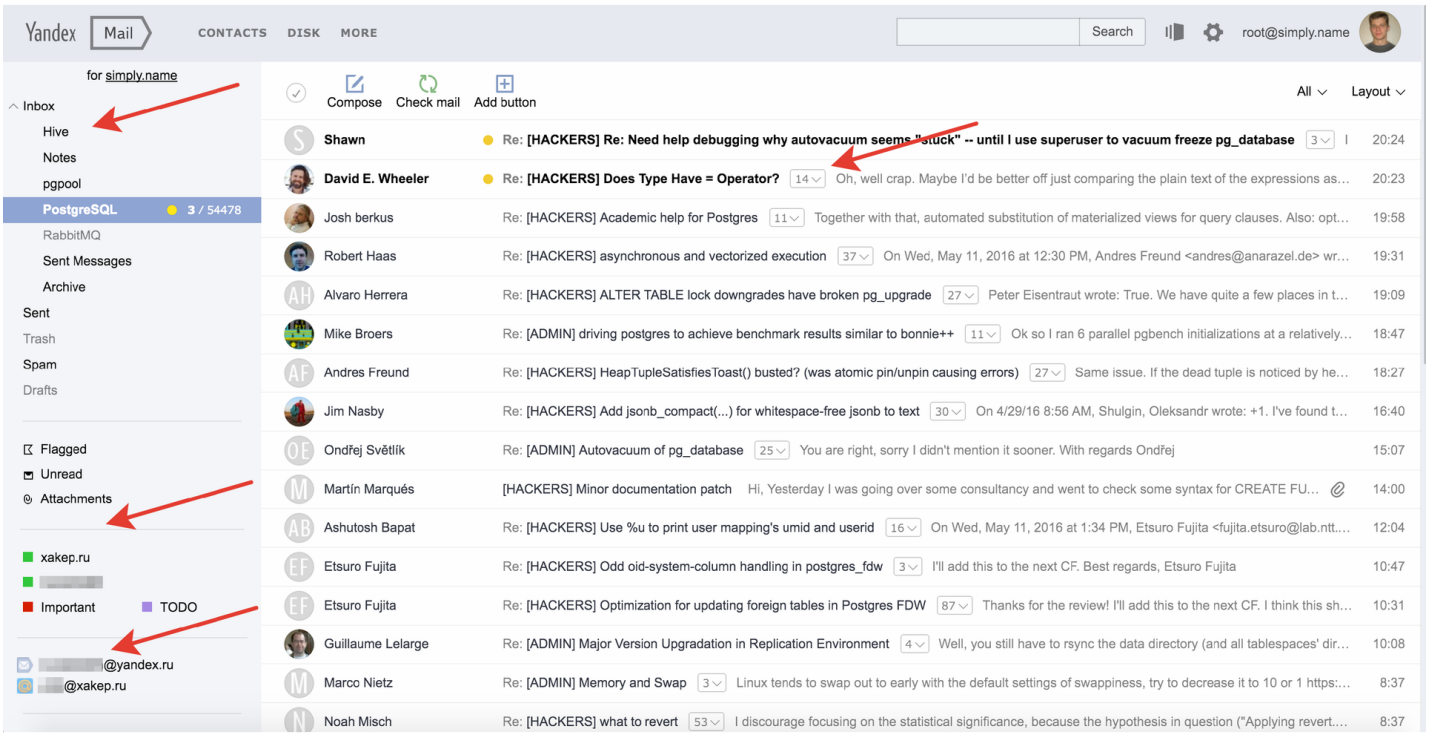
базы данных «Яндекс.Почты».

Сначала несколько фактов, которые будут иметь значение в будущем. «Яндекс.Почта» — сервис достаточно старый: он был запущен в 2000 году, и потому мы накопили много legacy. У нас — как это принято и модно говорить — вполне себе highload-сервис, больше 10 миллионов пользователей в сутки, какие-то сотни миллионов всего. В бэкенд нам прилетает более 200 тысяч запросов в секунду в пике. Мы складываем более 150 миллионов писем в сутки, прошедших проверки на спам и вирусы. Суммарный объём писем за все 16 лет — больше 20 петабайт.

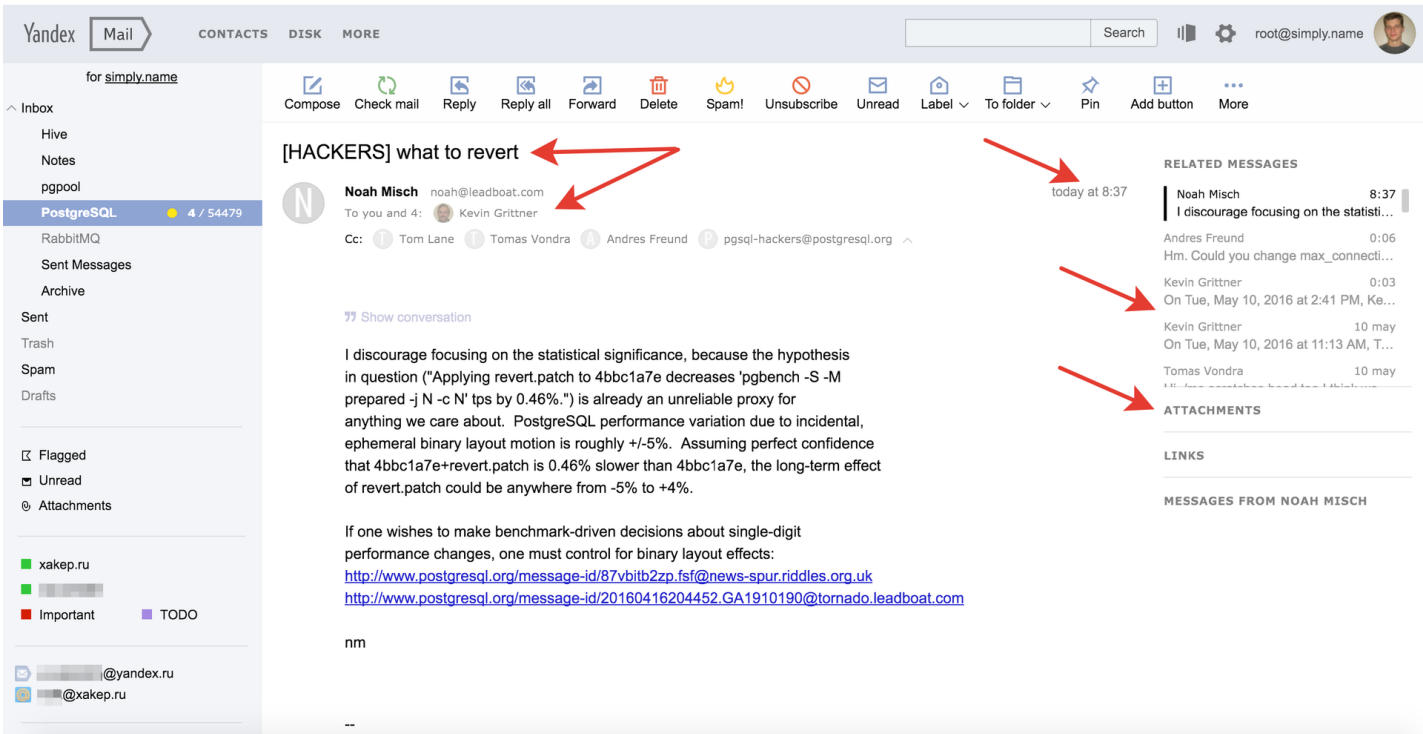
О чем пойдет речь? О том, как мы перевезли метаданные из Oracle в PostgreSQL. Метаданных там не петабайты — их чуть больше трехсот терабайт. В базы влетает более 250 тысяч запросов в секунду. Надо иметь в виду, что это маленькие OLTP-запросы, по большей части чтение (80%).

Это — не первая наша попытка избавиться от Oracle. В начале нулевых была попытка переехать на MySQL, она провалилась. В 2007 или 2008 была попытка написать что-то своё, она тоже провалилась. В обоих случаях был провал не столько по технически причинам, сколько по организационным.

Что является метаданными? Здесь они выделены стрелочками. Это папки, которые собой представляют какую-то иерархию со счётчиками, метки (тоже, по сути, списки со счётчиками), сборщики, треды и, конечно же, письма.



Мы не храним сами письма в метабазах, тела писем находятся в отдельном хранилище. В метабазах мы храним конверты. Конверты — это некоторые почтовые заголовки: от кого, кому, тема письма, дата и подобные вещи. Мы храним информацию о вложениях и цепочках писем.



Назад в 2012 год

Всё это лежало в Oracle. У нас было очень много логики в самой хранимой базе. Оракловые базы были самым эффективным железом по утилизации: мы складывали очень много данных на шард, больше 10 терабайт. Условно говоря, при 30 ядрах у нас нормальный рабочий load average был 100. Это не когда всё плохо, а при штатном режиме работы.

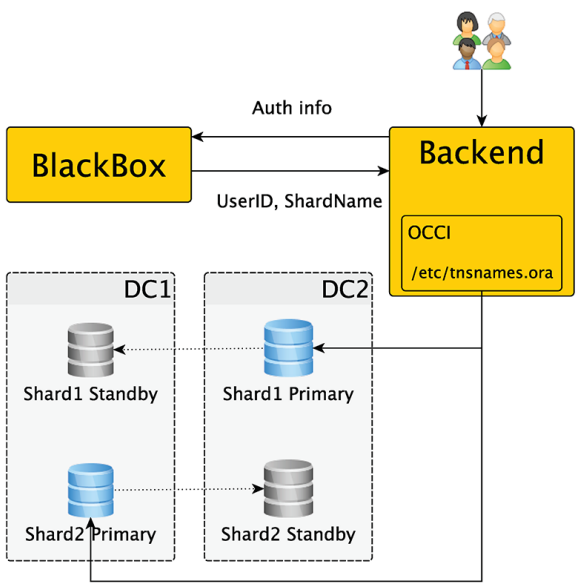
Баз было мало, поэтому многое делалось руками, без автоматизации. Было много ручных операций. Для экономии мы делили базы на «тёплые» (75%) и «холодные» (25%). «Тёплые» — это для активных пользователей, они с SSD. «Холодные» — для неактивных пользователей, с SATA.

Шардирование и отказоустойчивость — важная тема в «Яндексе». Шардирование — потому что в один шард всё не запихаешь, а

отказоустойчивость — потому что мы регулярно берем и отключаем один из наших дата центров, чтобы увидеть, что всё работает.

Как это было реализовано? У нас есть внутренний сервис BlackBox (чёрный ящик). Когда один запрос прилетает на один из наших бэкендов, бэкенд обменивает аутентификационные данные — логин, пароль, cookie, token или что-то подобное. Он идет с этим в BlackBox, который в случае успеха возвращает ему идентификатор пользователя и имя шарда.

Шардирование и отказоустойчивость



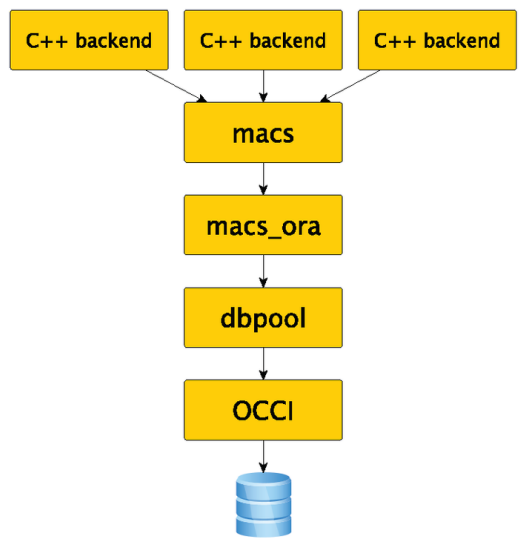
9

Затем бэкенд скармливал это имя шарда в оракловый драйвер OCCI, дальше внутри этого драйвера была реализована вся логика отказоустойчивости. То есть, грубо говоря, в специальном файлике /etc/tnsnames.ora были записаны shardname и список хостов, которые в этот шард входят, его обслуживают. Oracle сам решал, кто из них мастер, кто реплика, кто жив, кто мертв и т. д.

Итого, шардирование было реализовано средствами внешнего сервиса, а отказоустойчивость — средствами драйвера Oracle.

Большая часть бэкендов была написана на C++. Для того, чтобы не плодить «велосипедов», у них долгое время существовала общая абстракция macs meta access. Это просто абстракция для хождения в базы. Практически всё время у неё была одна реализация macs_ora для хождения непосредственно в Oracle. В самом низу, конечно же, OCCI. Еще была небольшая прослойка dbpool, которая реализовывала пул соединения.

Внутри приложения



10

Вот так это когда-то давно задумывалось, дизайнилось и реализовывалось. С течением времени абстракции протекли, бэкенды начали использовать методы из реализации macs_ora, еще хуже если из dbpool. Появились Java и всякие другие бэкенды, которые не могли использовать эту библиотеку. Всю эту лапшу потом пришлось мучительно разгребать.

Oracle — прекрасная база данных, но и с ней были проблемы. Например, выкладка PL/SQL кода — это боль, потому что есть library cache. Если база под нагрузкой, то нельзя просто взять и обновить код функции, который сейчас используется какими-то сессиями.

Остальные проблемы связаны не столько с Oracle, сколько с тем подходом, который мы использовали. Опять же: множество ручных операций. Переключение мастеров, наливка новых баз, запуск переносов пользователей — всё делалось руками, потому что баз было немного.

С точки зрения разработки есть недостаток в том, что плюсовый [C++] оракловый драйвер имеет только синхронный интерфейс. То есть нормальный асинхронный бэкенд поверх написать не получится. Это вызывало некоторую боль в разработке. Вторую боль в разработке вызывало то, что поднять тестовую базу проблематично. Во-первых, потому что руками, во-вторых, потому что это деньги.

Кто бы что ни говорил, поддержка у Oracle есть. Хотя поддержка enterprise-компаний зачастую далека от идеала. Но главная причина перехода — это деньги. Oracle стоит дорого.

Хронология

В октябре 2012 года, больше 4 лет назад, было принято решение о том, что мы должны избавиться от Oracle. Не звучало слов

PostgreSQL, не звучало каких-либо технических подробностей — это было чисто политическое решение: избавиться, срок в 3 года.

Спустя полгода мы начали первые эксперименты. На что были потрачены эти полгода, я могу чуть попозже рассказать. Эти полгода были важные.

В апреле 2013 мы поэкспериментировали с PostgreSQL. Тогда был очень модный тренд на всякие NoSQL-решения, и мы попробовали много всякого разного. Мы вспомнили, что у нас все метаданные уже хранятся в бэкенде поиска по почте, и может быть, можно использовать его. Это решение тоже попробовали.

Первый успешный эксперимент был со сборщиками, о нем я рассказывал на митапе в «Яндексе» в 2014 году.

Мы взяли небольшой кусочек (2 терабайта) довольно нагруженных (40 тысяч запросов в секунду) почтовых метаданных и унесли их из Oracle в PostgreSQL. Тот кусочек, который не очень связан с основными метаданными. У нас получилось, и нам понравилось. Мы решили, что PostgreSQL — наш выбор.

Далее мы запилили прототип почтовой схемы уже для PostgreSQL и начали складывать в него весь поток писем. Делали мы это асинхронно: все 150 миллионов писем в день мы складывали в PostgreSQL. Если покладка не удалась бы, то нам было бы всё равно. Это был чистый эксперимент, продакшн он не задевал.

Это позволило нам проверить первоначальные гипотезы со

схемой. Когда есть данные, которые не жалко выкинуть — это очень удобно. Сделал какую-то схему, напихал в неё писем, увидел, что не работает, всё дропнул и начал заново. Отличные данные, которые можно дропать, мы такие любим.

Также благодаря этому получилось в некоторой степени провести нагрузочное тестирование прямо под живой нагрузкой, а не какой-то синтетикой, не на отдельных стендах. Так получилось сделать первоначальные прикидки по железу, которое понадобится для PostgreSQL. И конечно же, опыт. Основная цель предыдущего эксперимента и прототипа — это опыт.

Дальше началась основная работа. Разработка заняла примерно год календарного времени. Задолго до того, как она закончилась, мы перенесли из Oracle в PostgreSQL свои ящики. Мы всегда понимали, что никогда не будет такого, что мы всем покажем на одну ночь «извините, технические работы», перенесем 300 терабайт и начнем работать на PostgreSQL. Так не бывает. Мы бы обязательно сломались, откатывались, и всё было бы плохо. Мы понимали, что будет довольно длительный период времени, когда часть ящиков будет жить в Oracle, а часть — в PostgreSQL, будет идти медленная миграция.

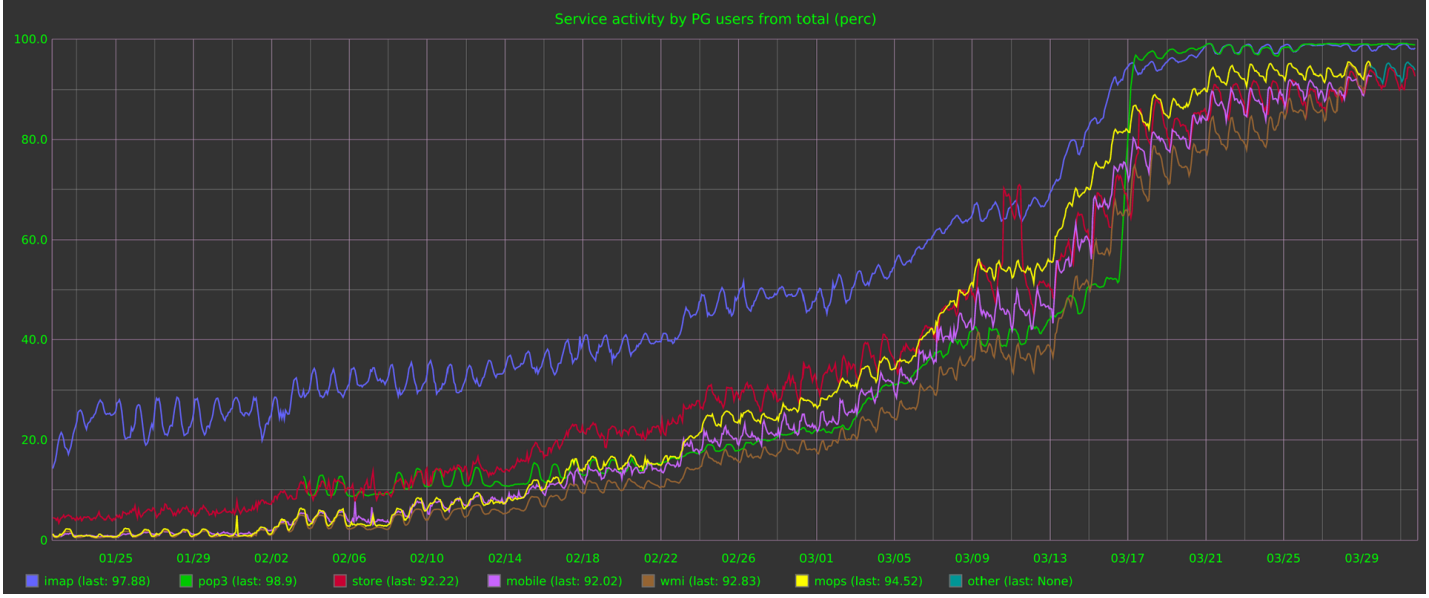
Летом 2015 года мы перенесли свои ящики. Команда «Почты», которая её пишет, тестирует, администрирует и так далее, перенесла свои ящики. Это очень сильно ускорило разработку. Страдает абстрактный Вася, или страдаешь ты, но можешь это поправить, — это две разные вещи.

Даже до того, как мы дописали и реализовали все фичи, мы начали нести неактивных пользователей. Неактивным мы называем такого пользователя, которому почта приходит, мы складываем письма, но он их не читает: ни вебом, ни с мобильным, ни IMAP — ему они неинтересны. Есть такие пользователи, к несчастью. Мы начали нести таких неактивных пользователей, когда у нас, допустим, ещё не полностью был реализован IMAP, или не работала половина ручек в мобильном приложении.

Но это не потому, что мы такие смелые и решили всем ящики сломать, а потому что у нас был план Б в виде обратного переноса, и он нам очень сильно помог. Была даже автоматизация. Если мы перенесли пользователя, и он вдруг попробовал, например, зайти в веб-интерфейс — проснулся и стал активным — мы обратно переносили его в Oracle, чтобы не ломать ему никакие фичи. Это позволило нам поправить кучу багов в коде трансфера.

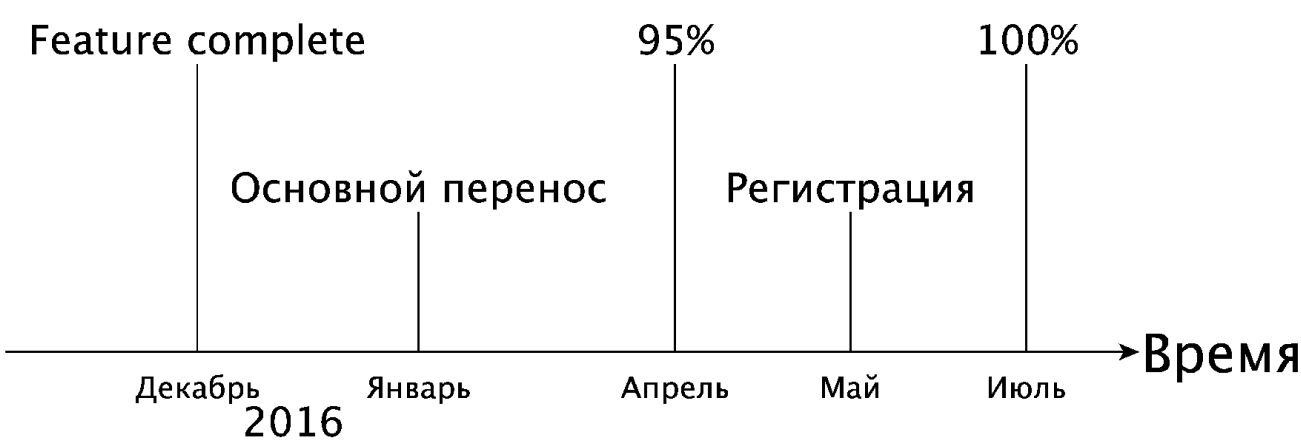
Затем последовала миграция. Вот несколько интересных фактов. 10 человеко-лет мы потратили на то, чтобы переписать всю нашу лапшу, которую мы накопили за 12—15 лет.

Сама миграция при этом прошла очень быстро. Это график за 4 месяца. Каждая линия — это процент нагрузки, который сервис отдаёт из PostgreSQL. Разбито на сервисы: IMAP, web, POP3, покладка, мобильные и так далее.



К сожалению, пропасть перепрыгнуть на 95% нельзя. Мы не смогли всех перенести к апрелю, потому что регистрация оставалась в Oracle, это довольно сложный механизм. Получилось так, что мы регистрировали новых пользователей в Oracle и сразу же ночью их переносили в PostgreSQL. В мае мы запилили регистрацию, и в июле погасили уже все базы Oracle.

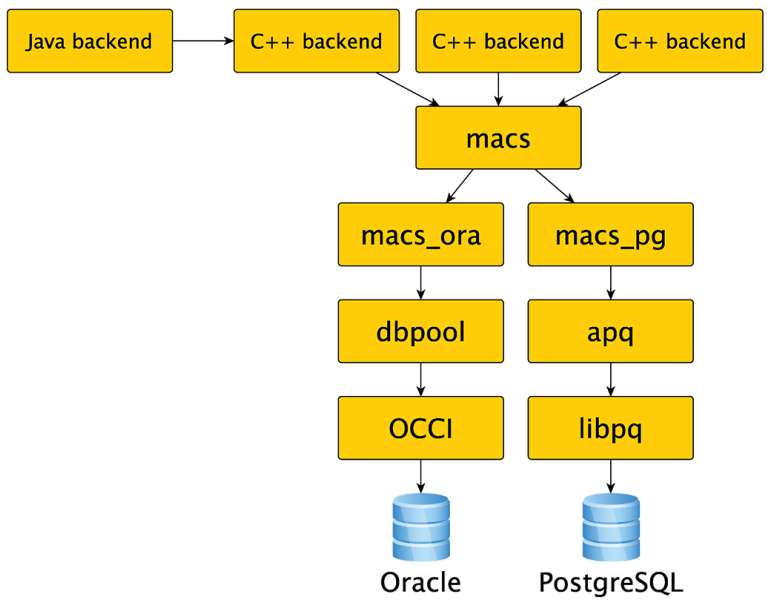
Завершение



Основные изменения

В нашей абстракции появилась еще одна реализация macs_pg, и мы распутали всю лапшу. Все те протекшие абстракции пришлось аккуратно переписать. Внизу у нее libpq, сделали еще небольшую прослойку apq, где реализован пул соединений, таймауты, обработка ошибок, и всё это асинхронно.

macs

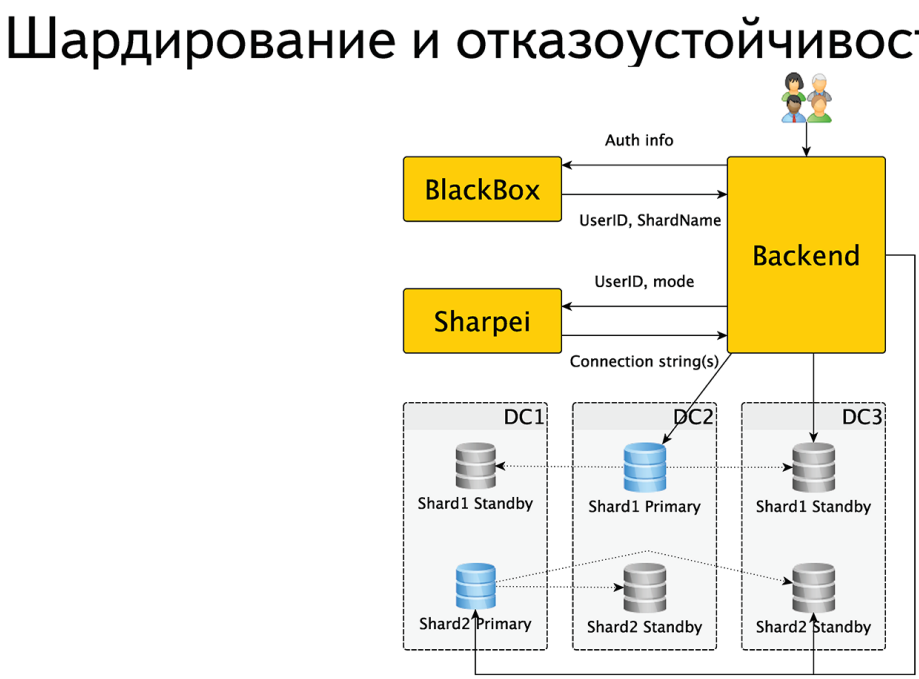


22

Шардирование и отказоустойчивость — всё то же самое. Бэкенд получает аутентификационные данные от пользователя, обменивает их в BlackBox на идентификатор пользователя и имя шарда. Если в имени шарда есть буква pg, то дальше он делает еще один запрос в новый сервис, который мы назвали Sharpei. Бэкенд передаёт туда идентификатор этого пользователя и режим, в котором он хочет получить базу. Например, «я хочу мастер», «я хочу синхронную реплику» или «я хочу ближайший хост». Sharpei возвращает ему строки подключения. Далее бэкенд открывает

соединение, его держит и использует.

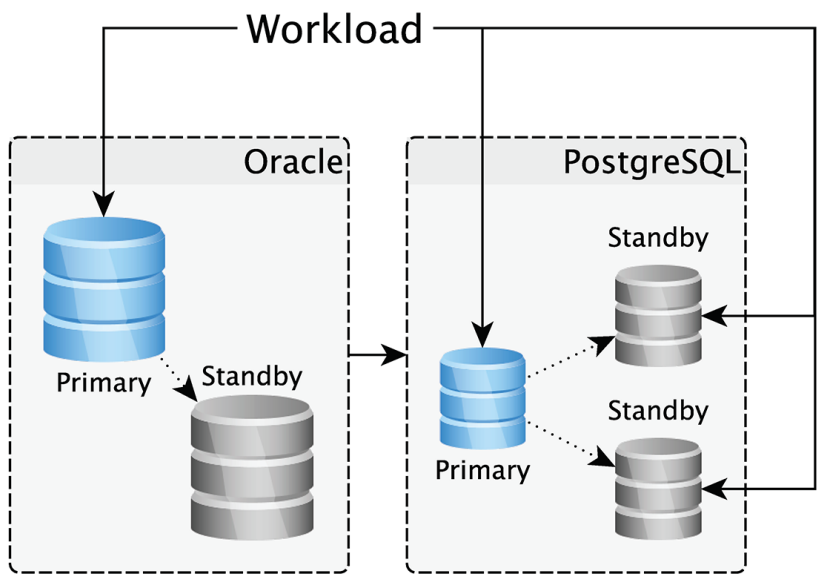
Чтобы знать информацию, кто мастер, кто реплика, кто жив, кто мертв, кто отстал, кто нет, Sharpei раз в секунду ходит в конечные базы и спрашивает их статусы. В этом месте появился компонент, который взял на себе обе функции: и шардирования, и отказоустойчивости.



В плане железа мы сделали несколько изменений. Поскольку Oracle лицензируется по процессорным ядрам, мы были вынуждены масштабироваться вертикально. На одно процессорное ядро мы напихивали много памяти, много SSD-дисков. Было небольшое количество баз с небольшим количеством процессорных ядер, но с огромными массивами памяти и дисков. У нас всегда была строго одна реплика для отказоустойчивости, потому что все последующие — это деньги.

В PostgreSQL мы поменяли подход. Мы стали делать базы поменьше и по две реплики в каждом шарде. Это позволило нам заворачивать читающие нагрузки на реплики. То есть в Oracle всё обслуживалось с мастера, а в PostgreSQL — три машины вместо двух поменьше, и чтение заворачиваем в PostgreSQL. В случае с Oracle мы масштабировались вертикально, в случае с PostgreSQL масштабируемся горизонтально.

Аппаратное обеспечение



24

Помимо «тёплых» и «холодных» баз появились еще и «горячие». Почему? Потому что мы внезапно обнаружили что 2% активных пользователей создают 50% нагрузки. Есть такие нехорошие пользователи, которые нас насилуют. Под них мы сделали отдельные базы. Они мало чем отличаются от теплых, там тоже SSD, но их меньше на одно процессорное ядро, потому что процессор там активнее используется.

Разумеется, мы запилили автоматизацию переноса пользователей

между шардами. Например, если пользователь неактивный, сейчас живет в саташной [с SATA-накопителем] базе и вдруг начал использовать IMAP, мы его перенесём в «тёплую» базу. Или если он в теплой базе полгода не шевелится, то мы его перенесём в «холодную».

Перемещение старых писем активных пользователей с SSD на SATA — это то, что мы очень хотим сделать, но пока не можем. Если ты активный пользователь, живешь на SSD и у тебя 10 миллионов писем, они все лежат на SSD, что не очень эффективно. Но пока что в PostgreSQL нормального секционирования нет.

Мы поменяли все идентификаторы. В случае с Oracle они были все глобально-уникальными. У нас была отдельная база, где было написано, что в этом шарде такие диапазоны, в этом — такие. Разумеется, у нас был факап, когда в силу ошибки пересеклись идентификаторы, а на их уникальность была завязана примерно половина всего. Это было больно.

В случае с PostgreSQL мы решили перейти к новой схеме, когда у нас идентификаторы уникальны в пределах одного пользователя. Если раньше уникальным был mid идентификатор письма, то теперь уникальной является пара uid mid. Во всех табличках у нас первым полем uid, им всё префиксивано, он является частью пока везде.

Кроме того, что это меньше места, есть ещё один неочевидный плюс. Поскольку все эти идентификаторы берутся из сиквенсов, у

нас меньше конкуренция за последнюю страничку индекса. В Oracle мы для решения этой проблемы вкрячивали реверсивные индексы. В случае PostgreSQL так как вставки идут в разные страницы индекса, мы используем обычные B-Tree, и у нас есть range-сканы, все данные одного пользователя в индексе лежат рядом. Это очень удобно.

Мы ввели ревизии для всех объектов. Это позволило читать с реплик, во-первых, неустаревшие данные, во-вторых, инкрементальные обновления для IMAP, мобильных. То есть ответ на вопрос «что изменилось в этой папке с такой-то ревизии» за счет этого сильно упростился.

В PostgreSQL всё хорошо с массивами, композитами. Мы сделали денормализацию части данных. Вот один из примеров:

Пример

```
xdb01g/maildb M # \dS mail.box
                        Table "mail.box"
    Column     |           Type           |       Modifiers
---------------+--------------------------+------------------------
 uid           | bigint                   | not null
 mid           | bigint                   | not null
 lids          | integer[]                | not null
<...>
Indexes:
    "pk_box" PRIMARY KEY, btree (uid, mid)
    "i_box_uid_lids" gin (mail.ulids(uid, lids)) WITH (fastupdate=off)
```


Это наша основная табличка mail.box. Она содержит по строчке на

каждое письмо. Первичным ключом у нее является пара uid mid. Ещё там есть массив меток lids, потому что на одном письме может быть больше одной метки. При этом есть задача отвечать на вопрос «дай мне все письма с такой-то меткой». Очевидно, что для этого нужен какой-то индекс. Если построить B-Tree индекс по массиву, то он не будет отвечать на такой вопрос. Для этого у нас используется хитрый функциональный индекс gin по полю uid и lids. Он позволяет нам отвечать на вопрос «дай мне все письма такого-то пользователя с такими-то метками или с такой-то меткой».

Хранимая логика

- Поскольку с Oracle было очень много боли с хранимой логикой, мы зареклись что в PostgreSQL хранимой логики не будет вообще, никакой. Но в процессе наших экспериментов и прототипов мы поняли, что **PL/pgSQL очень даже хорош**. Он не страдает проблемой с library cache, и мы не нашли других сильно критичных проблем.
- При этом **количество логики сильно сократили**, оставили только ту что нужна для логической целостности данных. Например, если вы кладёте письмо, то увеличиваете счётчик в табличке с папками.
- Поскольку нет undo, **цена ошибки стала сильно выше**. В undo мы залазили пару раз после выкладки плохого кода, про это мой коллега Александр делал отдельный доклад у нас на митапе.
- Из-за отсутствия library cache оно сильно **проще деплоится**. Мы катаемся по пару раз в неделю вместо раза в квартал, как

было раньше.

Подход к обслуживанию

- Поскольку мы поменяли аппаратное обеспечение и стали масштабироваться горизонтально, то и подход к обслуживаю баз мы поменяли. Базами мы теперь рулим **SaltStack**. Самая главная его киллер-фича для нас — это возможность видеть детальный diff между тем, что сейчас есть на базе, и тем, что мы от неё ожидаем. Если наблюдаемое устраивает, то человек нажимает кнопку «выкатить», и оно катится.
- Схему и код мы теперь изменяем **через миграции**. У нас был отдельный доклад про это.
- От ручного обслуживания мы ушли, всё, что можно, **автоматизировали**. Переключение мастеров, переносы пользователей, наливки новых шардов и так далее — всё это по кнопке и очень просто.
- Поскольку развернуть новую базу — это одна кнопка, мы получили **репрезентативные тестовые окружения** для разработки. Каждому разработчику по базе, по две, сколько захочет — это очень удобно.

Проблемы

Такие вещи гладко не проходят никогда.

Это список тредов в комьюнити с проблемами, которые мы самостоятельно решить не смогли.

- Problem with ExclusiveLock on inserts
- Checkpoint distribution
- ExclusiveLock on extension of relation with huge shared_buffers
- Hanging startup process on the replica after vacuuming on master
- Replication slots and isolation levels
- Segfault in BackendIdGetTransactions

То есть мы пошли в комьюнити, и нам помогли. Это была проверка, что делать, когда у тебя нет enterprise-поддержки: есть комьюнити, и оно работает. И это очень здорово. Разумеется, сильно больше проблем мы решили сами.

Например, у нас была очень популярна вот такая шутка: «В любой непонятной ситуации виноват autovacuum». Эти проблемы мы тоже порешали.

Нам очень не хватало способов диагностики PostgreSQL. Ребята из Postgres Pro запилили нам wait-интерфейс. Об этом я уже рассказывал на PG Day в 2015 году Питере. Там можно почитать, как это работает. С помощью ребят из Postgres Pro и EnterpriseDB оно вошло в ядро 9.6. Не всё, но какая-то часть этих наработок вошла в 9.6. Дальше эта функциональность будет улучшаться. В 9.6 появились столбцы, которые позволяют сильно лучше понимать, что происходит в базе.

Сюрприз. Мы столкнулись с проблемой с бэкапами. У нас recovery window 7 дней, то есть мы должны иметь возможность восстановиться на любой момент в прошлом за последние 7 дней. В Oracle размер места под все бэкапы и архивлоги был равен

примерно размеру базы. База 15 терабайт — и её бэкап за 7 дней занимает 15 терабайт.

В PostgreSQL мы используем barman, и в нём под бэкапы нужно места минимум в 5 раз больше, чем размер базы. Потому что WAL сжимаются, а бэкапы нет, там есть File-level increments, которые толком не работают, вообще всё однопоточное и очень медленное. Если бы мы бэкапили as is эти 300 терабайт мета-данных, у нас понадобилось бы примерно 2 петабайта под бэкапы. Напомню, всего хранилище «Почты» — 20 петабайт. То есть 10% мы должны были бы отрезать только под бэкапы мета-баз за последние 7 дней, что довольно плохой план.

Мы не придумали ничего лучше и запатчили barman, вот pull request. Уже почти год прошел, как мы их просим запилить эту киллер-фичу, а они просят с нас денег, чтобы замержить её. Очень наглые ребята. Мой коллега Евгений, который всё это и запилил, рассказывал об этом на PGday в 2016 году. Оно правда сильно лучше жмёт бэкапы, их ускоряет, там честные инкременты.

По опыту эксперимента, прототипа, других баз, которые к тому времени появились у нас на PostgreSQL, мы ожидали кучу граблей во время переноса. А их не было. Было много проблем, но с PostgreSQL они связаны не были, что было для нас удивительно. Было полно проблем с данными, потому что за 10 лет накопилось много всякого legacy. Внезапно обнаружилось, что в каких-то базах данные лежат в кодировке KOI8-R, или другие странные вещи. Разумеется, были ошибки в логике переноса, поэтому данные тоже

приходилось чинить.

Завершение

Есть вещи, которых нам очень не хватает в PostgreSQL.

Например, **секционирование**, чтобы двигать старые данные с SSD на SATA. Нам не хватает хорошего встроенного **recovery manager**, чтобы не использовать форк batman, потому что до ядра barman это, наверное, не доедет никогда. Мы уже устали: почти год их пинаем, а они не очень-то торопятся. Кажется, это должно быть не в стороне от PostgreSQL, а именно в ядре.

Мы будем развивать **wait-интерфейс**. Думаю, в 10-й версии случится **quourum commit**, там патч в хорошем состоянии. Ещё мы очень хотим **нормальную работу с диском**. В плане дискового I/O PostgreSQL сильно проигрывает Oracle.

Что в итоге? Если учитывать рейды-реплики, то у нас в PostgreSQL больше 1 петабайта. Недавно я считал, там чуть больше 500 миллиардов строк. Туда влетает 250 тысяч запросов в секунду. Всего у нас это заняло 3 календарных года, но мы потратили больше 10 человеко-лет. То есть усилие всей команды довольно внушительное.

Что мы получили? Стал быстрее деплой, несмотря на то, что баз стало сильно больше, а число DBA уменьшилось. DBA на этот проект сейчас меньше, чем когда был Oracle.

Хотели того мы или нет, но нам пришлось порефакторить весь код бэкенда. Всё то legacy, которое копилось годами, выпилилось. Наш код сейчас почище, и это очень хорошо.

Без ложек дёгтя не бывает. У нас сейчас в 3 раза больше железа под PostgreSQL, но это ничто по сравнению со стоимостью Oracle. Пока у нас не было крупных факапов.

Небольшое замечание от меня. В «Почте» мы используем много open source библиотек, проектов и готовых решений. К трём стульям, на которых мы плотно сидели, которые у нас есть почти везде, — Linux, nginx, postfix — добавился PostgreSQL. Сейчас мы его используем под многие базы в других проектах. Он нам понравился. Четвёртый — хороший, надёжный стул. Я считаю, это история успеха.

У меня всё. Спасибо!

Владимир Бородин — История успеха «Яндекс.Почты» с PostgreSQL

Проголосовать:

+109

Поделиться:

Сохранить:

Комментарии (119)

Похожие публикации

Партнерство Oracle и Microsoft: Oracle Database, WebLogic Server, Oracle Linux и Java в облаке Windows Azure

2

XaocCPS • 26 июня 2013 в 09:56

Настраиваемые кнопки в Яндекс.Почте

44

jogur_t • 22 мая 2013 в 14:45

Представляем данные Lotus Domino в Oracle Database с помощью Oracle Data Cartridge Interface

5

и Domino Java API

ИЗ ПЕСОЧНИЦЫ

art0int • 30 августа 2011 в 16:51

Популярное за сутки

Наташа — библиотека для извлечения структурированной информации из текстов на русском языке

14

alexkuku • вчера в 16:12

Unit-тестирование скриншотами: преодолеваем звуковой барьер. Расшифровка доклада

4

lahmatiy • вчера в 13:05

Люди не хотят чего-то действительно нового — они хотят привычное, но сделанное иначе

25

ПЕРЕВОД

Smileek • вчера в 10:32

Руководство по SEO JavaScript-сайтов. Часть 2. Проблемы, эксперименты и рекомендации

2

ПЕРЕВОД

ru_vds • вчера в 12:04

Как адаптировать игру на Unity под iPhone X к апрелю

0

P1CACHU • вчера в 16:13

Лучшее на Geektimes

Стивен Хокинг, автор «Краткой истории времени», умер на 77 году жизни — 33

HostingManager • вчера в 13:49

Обзор рынка моноколес 2018 — 70

lozga • вчера в 06:58

«Битва за Telegram»: 35 пользователей подали в суд на ФСБ — 40

alizar • вчера в 15:14

Стивен Хокинг и его работа — что дал ученый человечеству? — 8

marks • вчера в 14:46

Sunlike — светодиодный свет нового поколения — 17

AlexeyNadezhin • вчера в 20:32