# Оптимизатор СУБД Firebird: ORDER против SORT

kdv, iBase.ru, 16.08.2018-05.09.2018, 31.05.2025

Статья по мотивам видео
- youtube https://youtu.be/vZUj0j156dg
- rutube https://rutube.ru/video/a9d292c86acb6ce487bc26a6663ef161/

Текст статьи не повторяет видео – часть информации расширена, часть убрана.
В статье – тест планов с элементами ORDER и SORT, как ускорить такие запросы, разница между кэшем SuperServer, Classic и SuperClassic.
Вопросы по статье присылайте на support@ibase.ru.

## Оглавление



## Оптимизация запросов с GROUP BY и ORDER BY

Для оптимизации группировок и сортировок у оптимизатора есть два способа. Один способ – это проход в порядке индекса, если для группируемого или сортируемого столбца он есть, и сортировка в памяти и на диске, если такого индекса нет.

## PLAN table ORDER index

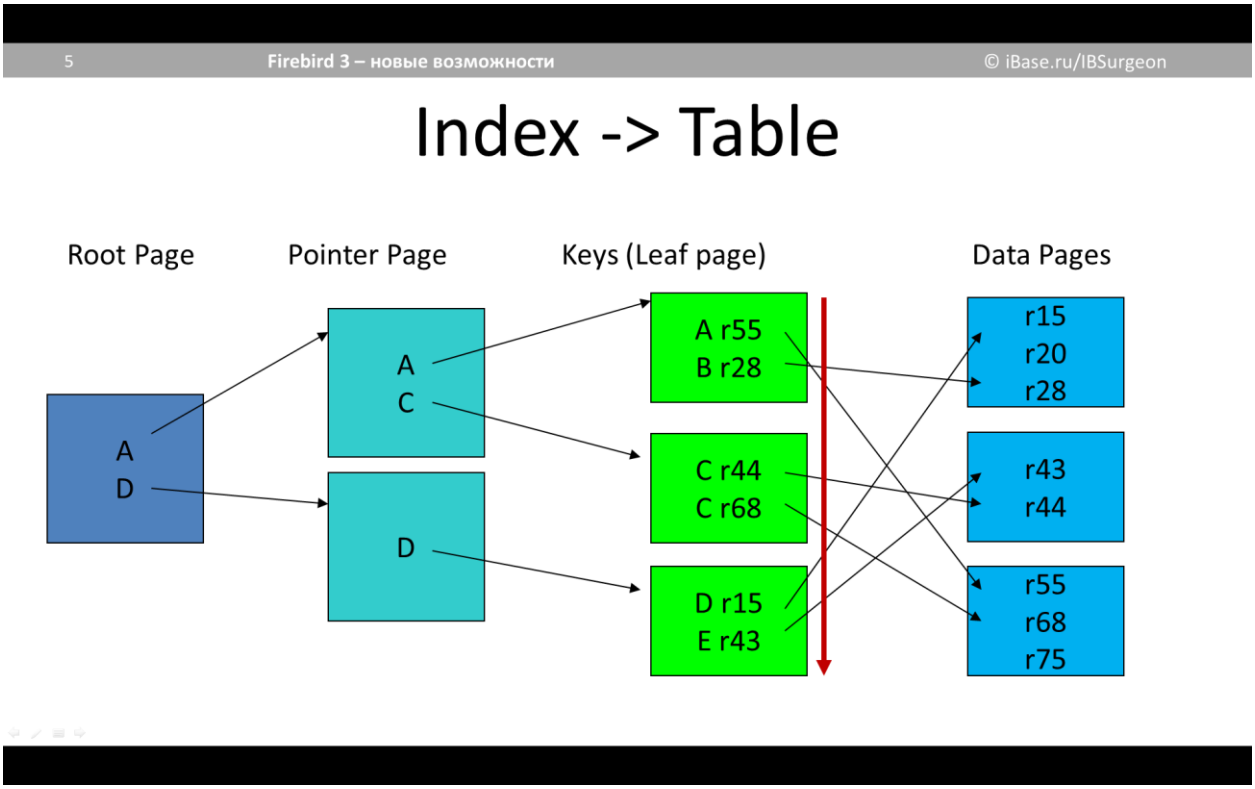


Рис 1.

В индексе значения ключей отсортированы, поэтому их порядок не соответствует расположению записей. Поэтому, при последовательном переборе ключей индекса сервер «прыгает» по страницам данных. По мере чтения страницы данных загружаются в кэш. Но при небольшом размере кэша, как у Classic и SuperClassic (рекомендуемые значения 512-2048 страниц, не более), чтение всё новых страниц данных в кэш приводит к выпадению уже прочитанных страниц из кэша. И чем меньше кэш, тем выше вероятность, что обращение к очередной странице приведет к её повторной загрузке с диска в кэш. Таким образом, запрос, использующий подобный метод доступа, будет генерировать высокий уровень ввода-вывода диск-память.

Запросы с планом, содержащим *table ORDER index*, имеют следующие характеристики:

- Первые записи сервер выдает очень быстро
- При дальнейшем чтении записей происходят прыжки по страницам данных, выпадение страниц из кэша и их повторная загрузка в кэш. Если попытаться получить все записи с сервера, то процесс может занять минуты или даже часы (в зависимости от производительности дисковой подсистемы и количества возвращаемых записей)
- Чем больше несовпадение порядка ключей в индексе и расположения записей на страницах данных, тем хуже производительность при таком способе доступа. В статистике, выдаваемой gstat в Firebird 3 вы можете увидеть, например, такие строки
  Index MINS_CLIENT (0)
  Clustering factor: 3 651 564, ratio: 0.26
  Это означает, что при минимальном кэше Firebird, запрос с планом table ORDER MINS_CLIENT произведет почти 3.5 миллионов чтений страниц данных с диска в кэш (хотя самих страниц данных в конкретном случае всего 120 тысяч).

## PLAN SORT

Альтернативный вариант для сортировки или группировки результата, если нет индекса по столбцу, это PLAN SORT.

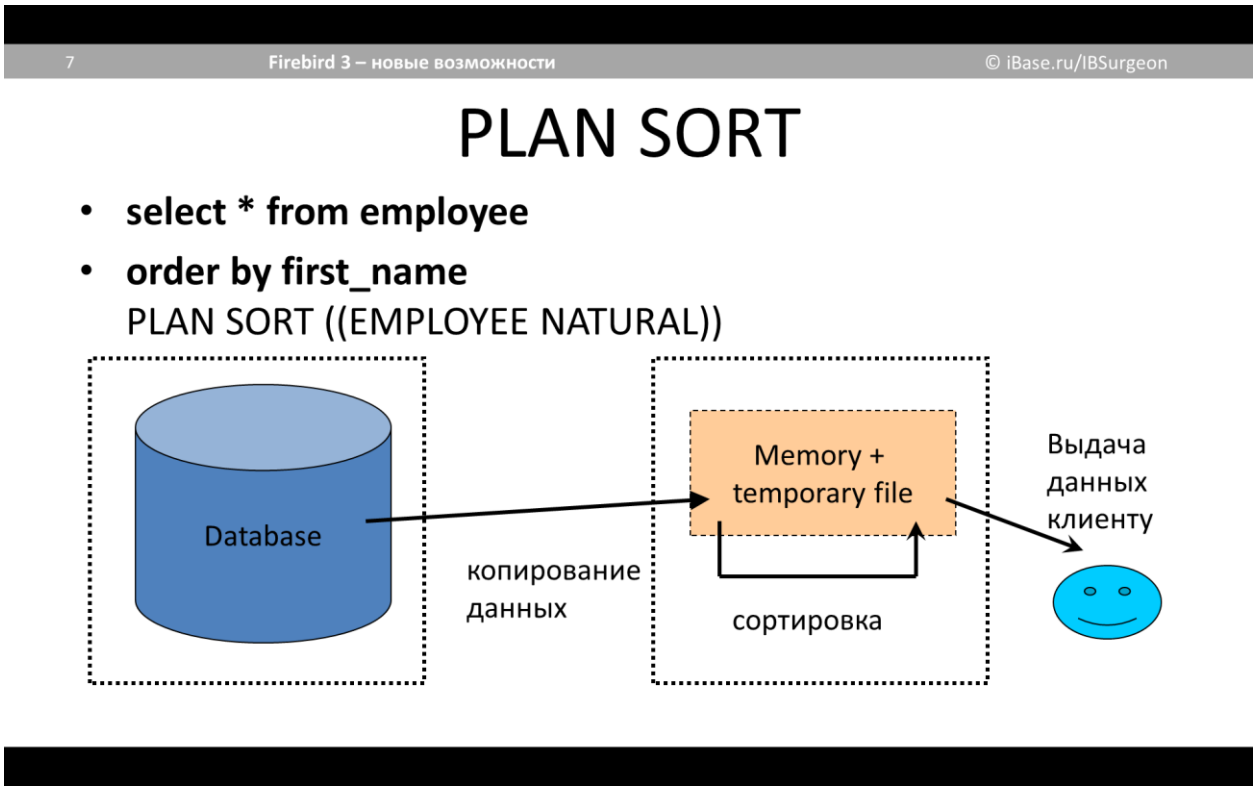


Рис 2.

В упрощенном виде – вначале данные переносятся в специальную область памяти Firebird для временных данных, и если они там не помещаются – то во временный файл. Далее данные сортируются, и по окончании сортировки выдаются пользователю.
В результате пользователь читает данные не из БД, а из временного файла. И ввод-вывод может быть распределен, если база данных и временные файлы находятся на разных физических носителях.
Временный файл с результатами запроса удаляется только когда пользователь прочитает из него последнюю запись. Поскольку временный файл – это временный файл операционной системы (как на Windows, так и на Linux), операционная система сама пытается кэшировать этот файл в доступной памяти.

Запросы с планом, содержащим *SORT*, имеют следующие характеристики:

- Результат выдается с задержкой (пока не отсортируется весь возвращаемый набор записей)
- После сортировки весь результат выдается максимально быстро
- Размер файла сортировки зависит от размера выдаваемой «записи (количества и размера возвращаемых и сортируемых столбцов) и количества записей. На каждый запрос (даже одинаковый) создается отдельный файл сортировки.

## Как оптимизируется ORDER BY

ORDER BY – это упорядочение набора записей по столбцу или набору столбцов. Без него записи всегда выдаются в произвольном порядке. В качестве аргумента ORDER BY можно использовать имя столбца, номер столбца, или вычисляемое выражение.

**Пример 1**

```
select client
from minutes
order by client
```

по столбцу client есть индекс MINS_CLIENT. План запроса будет следующим

```
PLAN (MINUTES ORDER MINS_CLIENT)
```

план запроса в explain plan Firebird 3:

```
Select Expression
    -> Table "MINUTES" Access By ID
        -> Index "MINS_CLIENT" Full Scan
```

То есть, используется выборка записей MINUTES в порядке сортировки ключей (столбца CLIENT) индекса MINS_CLIENT. В Explain plan фраза «full scan» означает, что будут перебраны все ключи индекса, т.к. нет ограничения where по столбцу order by на больше, меньше или равенство. Table access by id означает, что при чтении ключа берется номер записи, и эта запись считывается со страницы данных, на которой она находится (см. Рис. 1).

**Пример 2**

```
select client||'' cln
from minutes
order by 1
```

Индекса по выражению client||'' нет, и здесь это использовано как трюк, чтобы отключить использование индекса оптимизатором, как в примере выше. Для строк это конкатенация с пустой строкой, для числовых столбцов и даты-времени - +0.

```
PLAN SORT (MINUTES NATURAL)
```

Видим, что план изменился, и сервер выберет записи из MINUTES, отсортирует их, и только после этого сможет выдавать клиенту.

*Замечание*: в данном случае отсортированы будут все записи , даже если вы укажете ограничение FIRST или ROWS. Эти ограничения действуют только на выборку записей, а не на количество сортируемых записей.

```
Select Expression
    -> Sort (record length: 68, key length: 24)
        -> Table "MINUTES" Full Scan
```

В расширенном плане видно,  что длина ключа 24 байта. Это размер сортируемого столбца, выровненный на 4 байта (20+4 = 24). Длина записи – 68 байт – это длина ключа плюс номер записи, плюс номер транзакции, плюс выравнивание, и проч. Длина ключа входит в длину записи.

*Замечание*: если столбец client (или выражение) будет иметь длину 1000 байт (например, varchar(250) character_set utf8), то независимо от хранимых в столбце данных (пусть 10-20 символов максимум) для сортировки будет использоваться максимально возможная длина, то есть 1000 байт (чтобы все данные поместились). А это значит, что размер файла сортировки будет примерно равен record length * количество_записей. И чем больше у вас будет длинных столбцов в select, тем больше будет файл сортировки.

## Как оптимизируется GROUP BY

Группировка записей производится выполнением агрегатных функций по уникальным значениям неагрегатных столбцов. GROUP BY в качестве аргумента может иметь имя столбца, номер столбца, и вычисляемое выражение. В общем, всё то же самое, что и для ORDER BY. Разница в том, что ORDER BY это сортировка всего результата, а GROUP BY – применение к результату запроса группировки с агрегатными функциями MIN, MAX, AVG, COUNT и так далее.

**Пример 1**

```
select m.client, count(m.client) cnt
from minutes m
group by m.client
```

Выбираем уникальные значения client, и сколько раз они повторяются.
План ничем не отличается от предыдущего примера с ORDER BY по индексу

```
PLAN (M ORDER MINS_CLIENT)
```

Однако Explain plan Firebird 3 содержит слово Aggregate, которое показывает, что это не просто сортировка.

```
Select Expression
    -> Aggregate
        -> Table "MINUTES" as "M" Access By ID
            -> Index "MINS_CLIENT" Full Scan
```

**Пример 2**

```
select client||'' cln, count(client) cnt
from minutes
group by 1
```

Та же техника «отключения» индекса – превращение столбца в выражение.

```
PLAN SORT (MINUTES NATURAL)
```

План меняется на SORT,

```
Select Expression
    -> Aggregate
        -> Sort (record length: 68, key length: 24)
            -> Table "MINUTES" Full Scan
```

и в расширенном плане те же самые значения размеров ключа и записи.

## Отличается ли скорость ORDER BY от GROUP BY?

Если планы одинаковые, то есть ли различия в скорости?
Дело в том, что тестировать сортировку/группировку миллионов записей по индексу невозможно. Если мы выберем только первые 10 записей (программно, чтением в grid, или указанием в запросе FIRST 10), то дальше проход по индексу будет прекращен, и у вас может возникнуть ложное впечатление, что выборка всех записей производится очень быстро.
Для полноценного теста нужно прочитать все записи, а это значит выбрать на клиента миллионы записей. Они вряд ли поместятся в память клиентского приложения, и кроме того, на пересылку миллионов записей по сети уйдет приличное количество времени.
В отношении SORT, наоборот, записи можно было бы не выбирать на клиента, поскольку основные действия уже произведены – чтение данных и их сортировка. Но тогда это не будет эквивалентом теста table ORDER index, где для полной оценки мы просто обязаны перебрать все записи.
Если же проводить тест на таблицах с малым количеством записей, то группировка или сортировка будет такой быстрой, что разницу понять не удастся.
Поэтому, специально для возможности замены в тесте ORDER BY на GROUP BY, чтобы сократить количество выдаваемых записей, я проверил разницу следующим образом:
с версии Firebird 2.0 в SQL есть производные таблицы (derived tables), то есть, на запрос, сортирующий миллионы записей можно сверху навесить select count. Например:

```
select count(*) from
 (select client
  from minutes
  order by client)
```

То есть,  внешний запрос переберет выдаваемые вложенным запросом записи, независимо от того, каким способом их обработает вложенный запрос. В результате, клиент никак не повлияет на то, сколько записей и как обрабатываются сервером.

**Итог проверки** – при плане table ORDER index GROUP BY и ORDER BY не отличаются ничем. Скорость та же, чтений с диска столько же, и фетчей из кэша – одинаково.
При плане SORT – ORDER BY на 14% быстрее, чем GROUP BY. Так что для дальнейшей части статьи надо помнить, что результаты PLAN SORT несколько быстрее, чем приведены, потому что в тесте использовался GROUP BY.
Почему все приведенные в этой статье результаты не были пересчитаны с использованием внешнего SELECT COUNT? Потому что идея такой проверки появилась не до, а после проведения основного теста, в котором ORDER BY был заменен на GROUP BY.

## Немного специфики

Что, если в запросе указана группировка и сортировка по одному и тому же столбцу? По идее, сервер в этом случае будет делать двойную работу. Но если GROUP BY сортирует результат точно так же, как ORDER BY, то зачем тогда писать ORDER BY?
Увы, ORDER BY писать нужно, потому что в следующих версиях метод обработки GROUP BY может измениться, и сортировать «как-то не так», как это нужно для ORDER BY.

**Пример 1**

```
select m.client, count(m.client) cnt
from minutes m
group by m.client
order by m.client
```

В стандартном плане видно, что используется только одна сортировка и для GROUP BY, и для ORDER BY, только непонятно, для какой из частей

```
PLAN (M ORDER MINS_CLIENT)
```

В расширенном плане уже ясно, что сортировка используется для GROUP BY, а для ORDER BY она «пропускается»

```
Select Expression
    -> Aggregate
        -> Table "MINUTES" as "M" Access By ID
            -> Index "MINS_CLIENT" Full Scan
```

**Пример 2**

Сортируем результат по столбцу, отличному от столбца группировки

```
select m.client, count(m.client) cnt
from minutes m
group by m.client
order by 2
```

Здесь сразу видно, что после прохода по индексу возникает сортировка на диске

```
PLAN SORT (M ORDER MINS_CLIENT)
Select Expression
    -> Sort (record length: 52, key length: 8)
        -> Aggregate
            -> Table "MINUTES" as "M" Access By ID
                -> Index "MINS_CLIENT" Full Scan
```

Замечу, что даже если группируются миллионы записей, но на выходе группировки получается гораздо меньше, то сортировка сгруппированных записей не вызывает проблем, и скорее всего

произойдет в памяти. Если же группировка выдаст большое число записей, то сортировке тоже придется потрудиться, и запрос будет выполняться достаточно долго.

**Пример 3**

Теперь сгруппируем и отсортируем по выражению

```
select m.client||'', count(m.client) cnt
from minutes m
group by 1
order by 1
```

Две сортировки!

```
PLAN SORT (SORT (M NATURAL))

Select Expression
    -> Sort (record length: 76, key length: 24)
        -> Aggregate
            -> Sort (record length: 68, key length: 24)
                -> Table "MINUTES" as "M" Full Scan
```

Получается как-то не очень хорошо, и непонятно, почему так. Дело в том, что если вместо m.client||'' будет указан просто неиндексированный столбец, то все будет нормально, то есть сортировка будет только одна. Но вот для группировки плюс сортировки по одному выражению есть проблема с двойной сортировкой, и она зарегистрирована в трекере http://tracker.firebirdsql.org/browse/CORE-5814

Точно так же, как и в Примере 2, производительность здесь может быть «спасена» только если группировка вернет относительно небольшое количество записей.

## Размер записи сортировки

Размер записи сортировки зависит от количества и типов столбцов, выбираемых запросом. Вот один и тот же запрос, с разным количеством выбираемых столбцов

```
select * from employee
order by first_name
PLAN SORT (EMPLOYEE NATURAL)
Select Expression
    -> Sort (record length: 142, key length: 24)
        -> Table "EMPLOYEE" Full Scan

select emp_no, last_name, first_name, salary from employee
order by first_name
Select Expression
    -> Sort (record length: 86, key length: 24)
        -> Table "EMPLOYEE" Full Scan
```

В первом случае указана звездочка, т.е. выбирать все столбцы. В результате размер записи сортировки равен 142 байта, и тут еще повезло, что среди столбцов таблицы EMPLOYEE нет длинных строковых столбцов в сотни или тысячи байт.
Во втором случае выбираются конкретные столбцы, и размер записи сортировки почти в 2 раза меньше, 86 байт.
Следовательно, при одинаковом количестве записей для второго запроса размер файла сортировки будет в 2 раза меньше.

## Сравнение ORDER и SORT

Исходная таблица содержит 14 миллионов записей. Группируются или сортируются все записи, по индексу и без индекса.

**PLAN (TABLE ORDER A)**
**Execute time = 1m 12s**
**Buffers = 2 048**
**Reads = 3 627 028**
**Fetches = 32 224 797**

**PLAN SORT ((A NATURAL))**
**Execute time = 35 s**
**Buffers = 2 048**
**Reads = 119 915**
**Fetches 14 767 524**

Сортировка в этом примере 2 раза быстрее, чем проход в порядке индекса.
Ускорить сортировку можно увеличением параметра TempCacheLimit в firebird.conf, и размещением временных файлов на SSD или ram-диске.
Но перед настройками нужно посмотреть на размер сортируемых файлов (имена файлов - fb_...tmp, находятся во временных папках определенных в системе, или куда указывает параметр TempDirectories в firebird.conf). Тестовый запрос должен содержать именно ORDER BY, а не GROUP BY, потому что GROUP BY при тех же размерах файла сортировки может выдать так мало записей, что они будут сразу буферизированы клиентским приложением, и файл сортировки исчезнет с диска.
У тестируемого запроса размер файла сортировки – 1 гигабайт. По умолчанию TempCacheLimit 8 мегабайт для Classic (раздельная память сортировок), и 64 мегабайта для SuperClassic и SuperServer (общая память сортировок). Для проверки можем увеличить этот параметр до 2х гигабайт, но с 32битным Firebird этого делать нельзя, т.к. у 32битных процессов это максимум памяти, которую они могут адресовать и получить от операционной системы.

Меняем firebird.conf, рестартуем Firebird и повторяем запрос. Файл сортировки в temp так и не появился, Firebird-у хватило выделенной памяти. Но время осталось тем же самым – 35 секунд. Отсюда вывод – на конкретном оборудовании у операционной системы достаточно памяти для хранения временного файла сортировки в своем кэше (т.е. в RAM). А сам файл создается «для проформы», на тот случай, если файловый кэш вдруг придется освободить для других задач.
На вашем сервере, конечно, может быть по другому.

Вывод – можно не настраивать firebird.conf, если у операционной системы достаточно памяти, она и так сама закэширует временные файлы сортировки, если они вообще у вас есть.

### Почему запрос с планом ORDER в 2 раза медленнее?

Обратите внимание на Reads. При плане SORT просто вся таблица была прочитана один раз (120 тысяч страниц), выгружена в файл сортировки, отсортирована и выдана клиенту. А при плане

ORDER чтений с диска – 3.6 миллиона страниц. В 30 раз больше, чем сам размер таблицы. Это как раз происходит из-за «прыжков» по страницам данных, о которых было рассказано в начале статьи.
Ускорить запрос с таким планом можно только за счет увеличения размера кэша Firebird (параметр DefaultDbCachePages в firebird.conf). Но для Classic и SuperClassic этого делать нельзя, потому что кэш страниц приватный для каждого соединения, и если кэш увеличивать, то потребляемый размер памяти умножается на количество коннектов. Для SuperServer кэш страниц можно увеличить без проблем, т.к. он общий для всех соединений.

## Classic против SuperClassic против SuperServer

Теперь, зная всю специфику, можно проверить разницу сортировки и прохода по индексу на разных архитектурах, и может ли увеличение кэша страниц повлиять на скорость запросов.
В Firebird 3 SuperServer отлично распараллеливается по ядрам процессора, так что вполне можно избавиться от Classic и сильно увеличить кэш.

Поскольку с запросом с PLAN SORT уже всё понятно – кэш Firebird не влияет, и везде всё будет одинаково, проверяем запрос с планом table ORDER index, где 3.6 миллиона чтений страниц с диска.
У всех архитектур кэш явно задан равным 2048 страниц.

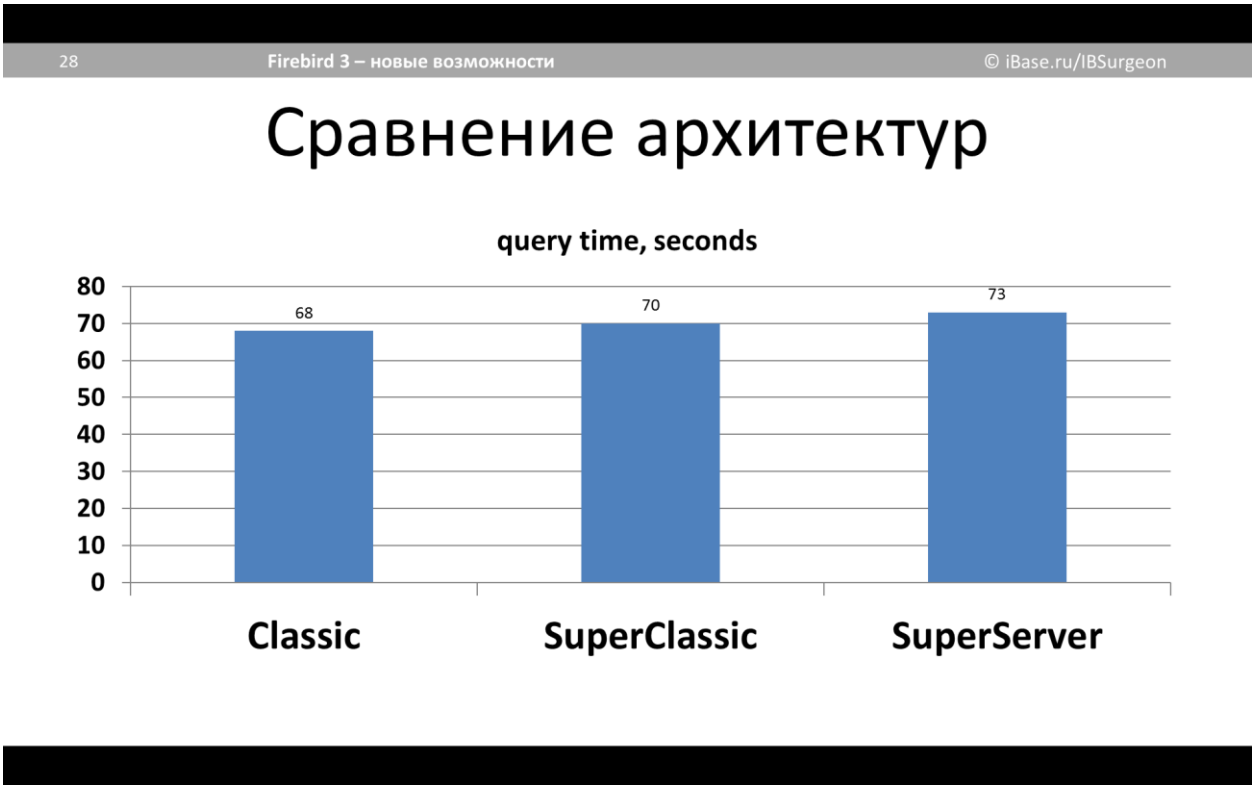


Рис. 3

Видно, что Classic быстрее всех (68 секунд), SuperClassic чуть медленнее (70 секунд), и SuperServer еще медленнее (73 секунды). 5 секунд разницы – это примерно 7.1%, так что больших отличий по производительности здесь нет. Кроме того, запрос выполнялся в монопольном режиме, с одним коннектом.

Но зачем оставлять у SuperServer кэш размером 2048 страниц? Из предыдущих данных видно, что все 14 миллионов записей тестовой таблицы помещаются на 119 тысячах страниц. Зачем их читать по 30 раз каждую из-за маленького кэша?
Поэтому, следующий тест – SuperServer с кэшем в 150 тысяч страниц (не забудьте увеличить параметр FileSystemCacheThreshold больше значения DefaultDbCachePages, иначе Firebird отключит файловый кэш операционной системы для базы данных).

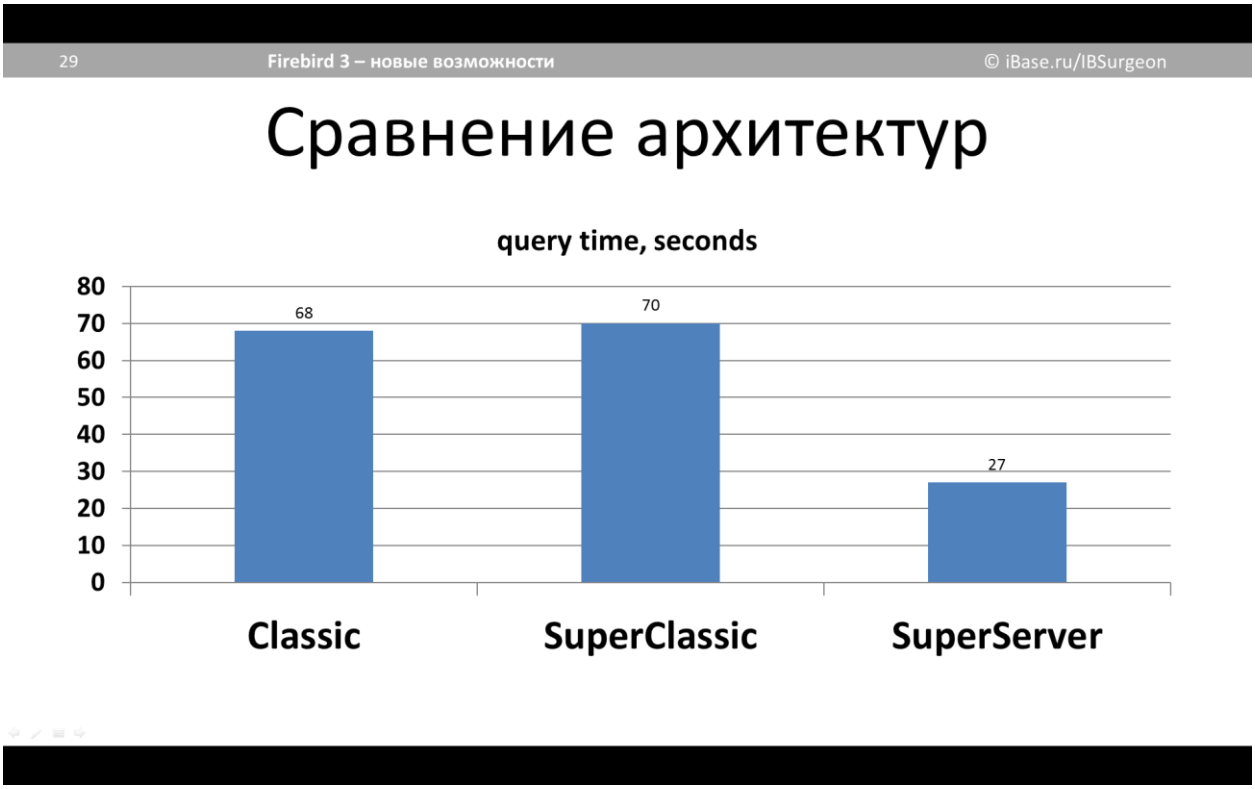


Рис. 4.

С таким кэшем запрос на SuperServer стал выполняться в 2 раза быстрее – 27 секунд вместо прежних 73 секунды. Это результат того, что страницы не считываются повторно с диска в кэш, и туда поместились не только страницы данных, но и страницы индекса (в сумме – 124 тысячи чтений страниц с диска).
Обратите внимание, что 27 секунд – быстрее 35 секунд с планом SORT на любой архитектуре.
Еще одна польза от большого кэша суперсервера, что все коннекты пользуются этим кэшем. Если другие коннекты выполняют такой же запрос, или обращаются к тем же таблицам, что уже хотя бы частично в кэше, то это однозначно даст прирост производительности по сравнению с Classic и SuperClassic. Конечно, другие коннекты могут начать читать другие таблицы, и вытеснить эту из кэша, но ничего не мешает увеличить кэш Firebird еще больше – 150 тысяч страниц по 16 килобайт, это 2.3 гигабайта RAM.

Проверим только SuperServer с разным кэшем и повторным выполнением запроса.

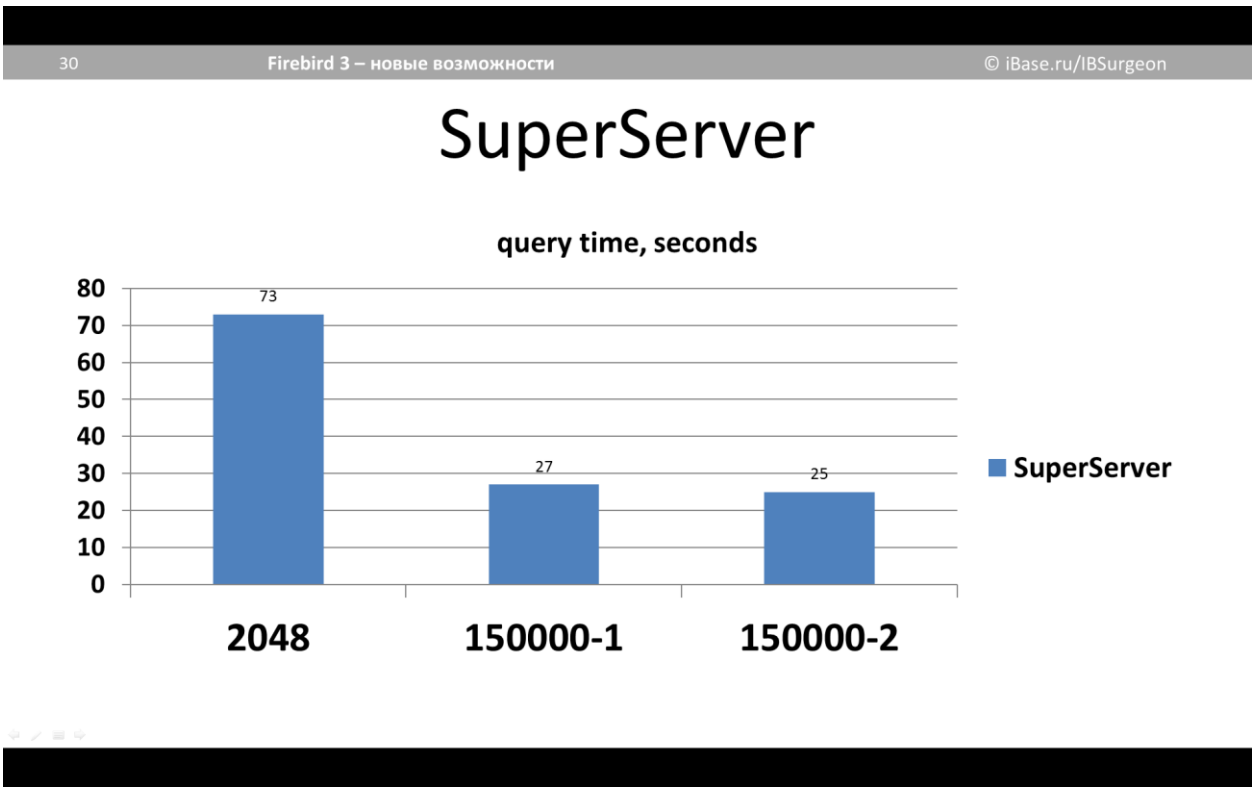


Рис. 5.

Повторное выполнение запроса дает Page reads равным нулю, т.к. все нужные страницы уже в кэше. И в этом случае запрос выполнился на 2 секунды быстрее – 25 секунд против 27 секунд.

Впрочем, если для Classic и SuperClassic выставить такой же кэш, скорость выполнения запросов улучшится абсолютно так же, до 27-25 секунд. Но мы этого сделать не можем, потому что каждый пользовательский коннект будет потреблять по 2.3 гигабайт памяти.

Поэтому дальше речь будет идти только про SuperServer.

## Последовательное увеличение кэша

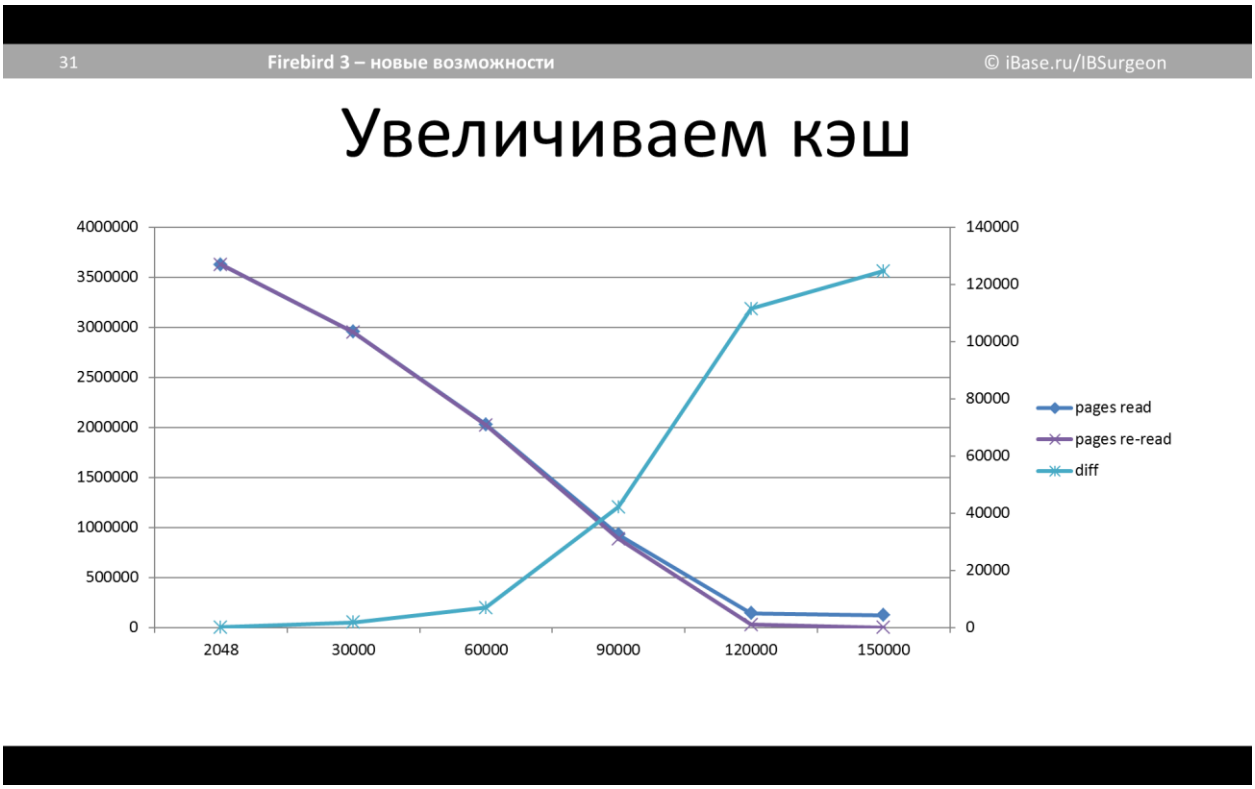


Рис. 6.

Здесь приведены данные по последовательному увеличению размера кэша Firebird – 2048 страниц, 30 тысяч, 60, 90, 120 и 150 тысяч.
Тест выполнялся сразу после рестарта Firebird, замерян page reads, затем повторное выполнение запроса с новым page reads, чтобы оценить эффективность заполнения (и вытеснения) конкретного объема кэша.
На левой шкале – количество чтений с диска в кэш. График голубого цвета со шкалой справа – отличие pages read от pages re-read. В самом конце графика отличие 124 тысячи, то есть при первом выполнении прочитано 124 тысячи страниц, а при повторе – 0 чтений, все страницы в кэше. То есть, чем больше разница, тем лучше.
Видно, что перелом по перечитываниям для данного запроса наступает на уровне кэша примерно в 90 тысяч страниц.

## Скорость при увеличении кэша




Рис 6.

Ничего необычного. Раз количество чтений с диска уменьшается при увеличении кэша, то и скорость выполнения запроса увеличивается, с тех самых 73 секунды до 25-27 секунд.

## Объединенный график



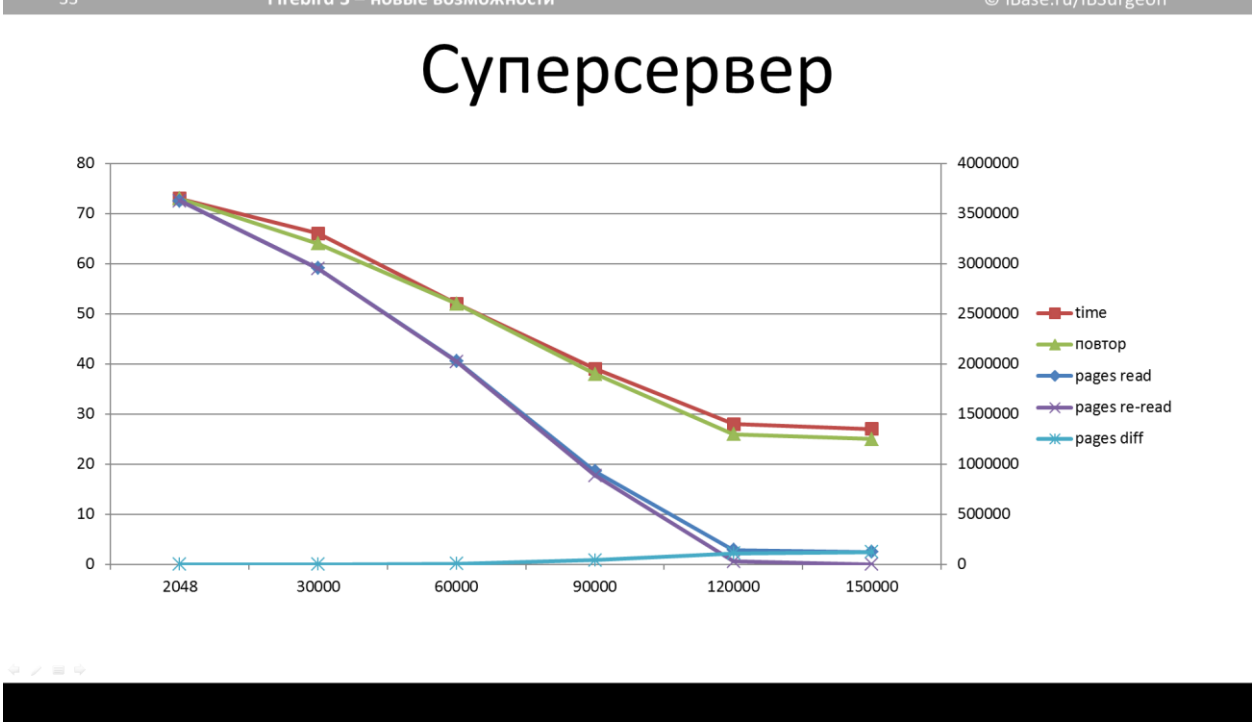


Рис. 7.

Здесь сведены предыдущие графики. Левая шкала – время выполнения запроса (красный и зеленый графики). Правая шкала – чтений страниц (все синие графики).

Здесь может возникнуть вопрос. Хорошо, мы увеличиваем кэш, запрос ускоряется. Но почему такая небольшая разница между первым и повторным выполнением запроса. При кэше в 90 тысяч страниц разницы практически нет, она появляется только с кэшем в 120 тысяч страниц. И даже при кэше в 150 тысяч страниц – разница всего 2 секунды (27 и 25 секунд).
При повторном выполнении запроса кэш Firebird заполнен, чтений с диска не происходит. И выходит, что кэш Firebird дает для повторного чтения только 2 секунды?
Для ответа нужно провести тот же тест без кэша операционной системы.

## Влияние кэша операционной системы

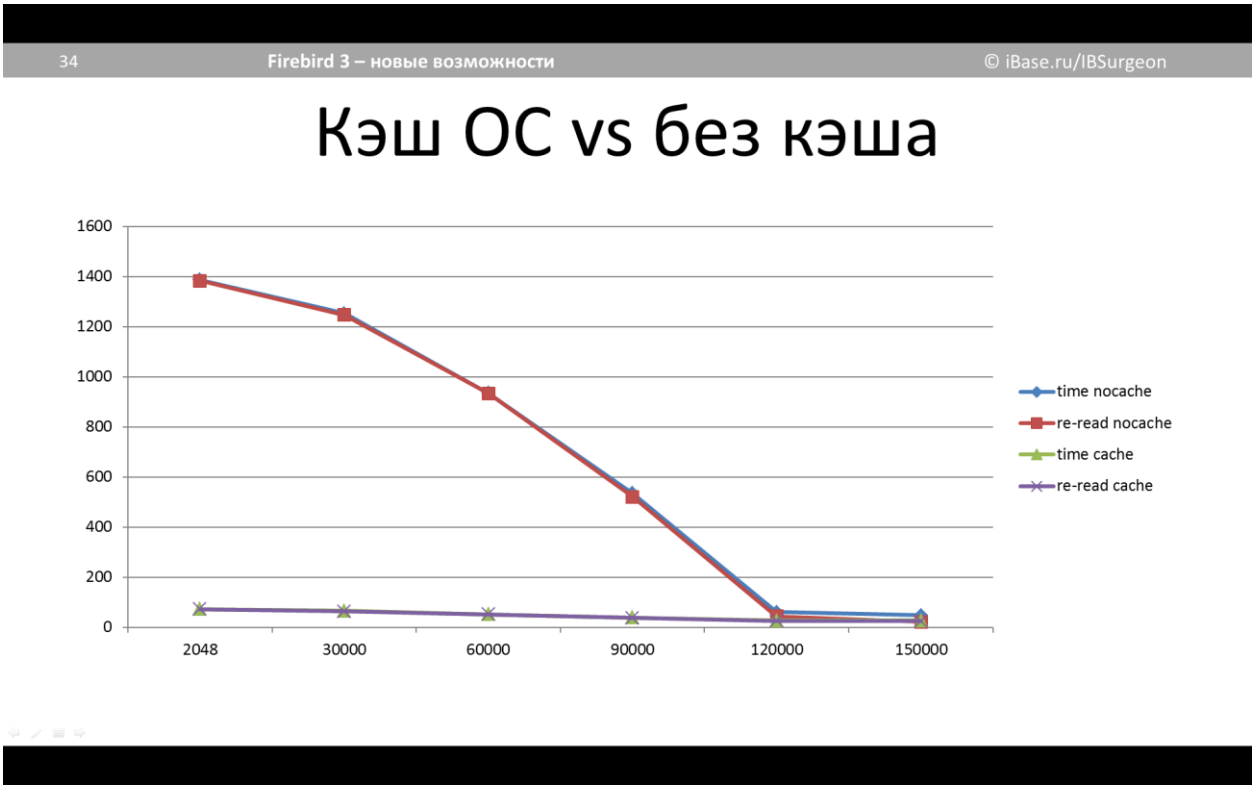


Рис. 8

На этом графике к времени с кэшем (внизу) добавлены графики с выключенным кэшем ОС. Тест повторен с параметром FileSystemCacheThreshold равным нулю, в результате чего кэш операционной системы не используется.

Производительность при отключенном кэше ОС и малом кэше Firebird падает катастрофически. Сравните 73 секунды с кэшем ОС, и 1387 секунд без кэша ОС (шкала слева). Это двадцать три минуты против одной минуты, и если быть точным, то в 19 раз медленнее. Получается, при малом кэше у Classic и SuperClassic файловый кэш операционной системы помогает весьма сильно.

Однако, для SuperServer с увеличением кэша производительность сильно повышается, так что кэш Firebird вполне работает и дает существенный прирост производительности.
Увеличим правую часть графика, чтобы было лучше видно результаты.

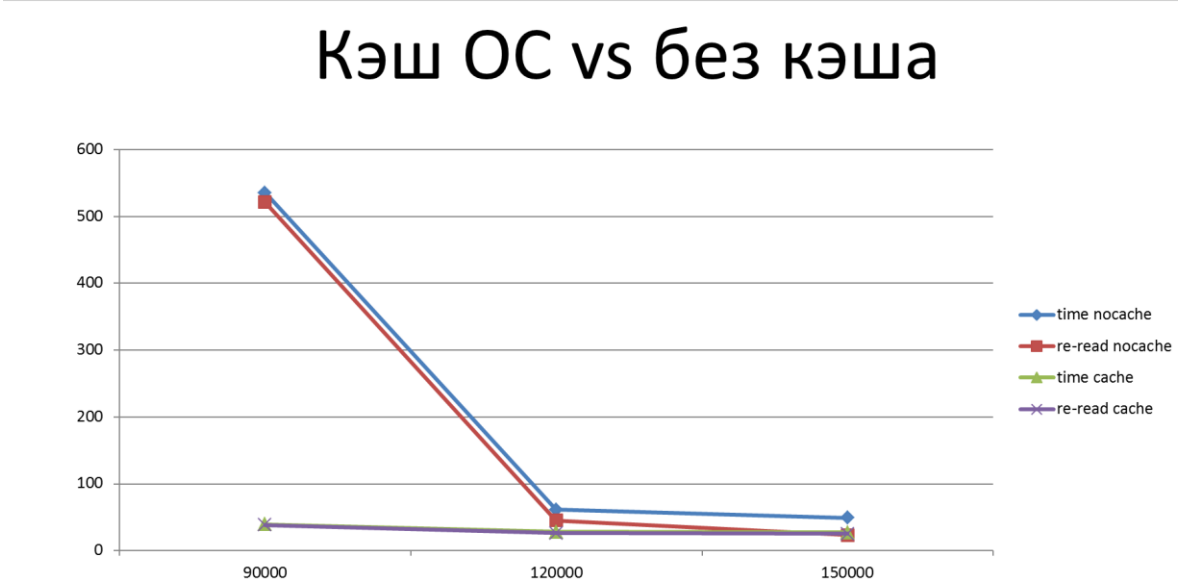


Рис. 9

Оставлены результаты с кэшем в 90, 120 и 150 тысяч страниц. Здесь видно, что при выключенном кэше ОС и кэше Firebird 90 тысяч страниц запрос уже не в 19, а в 7 раз медленнее, чем с кэшем ОС. Волшебных цифр, примерно равных скорости с кэшем операционной системы, Firebird достигает при своем кэше в 120 тысяч страниц.
Еще раз увеличим часть графика.



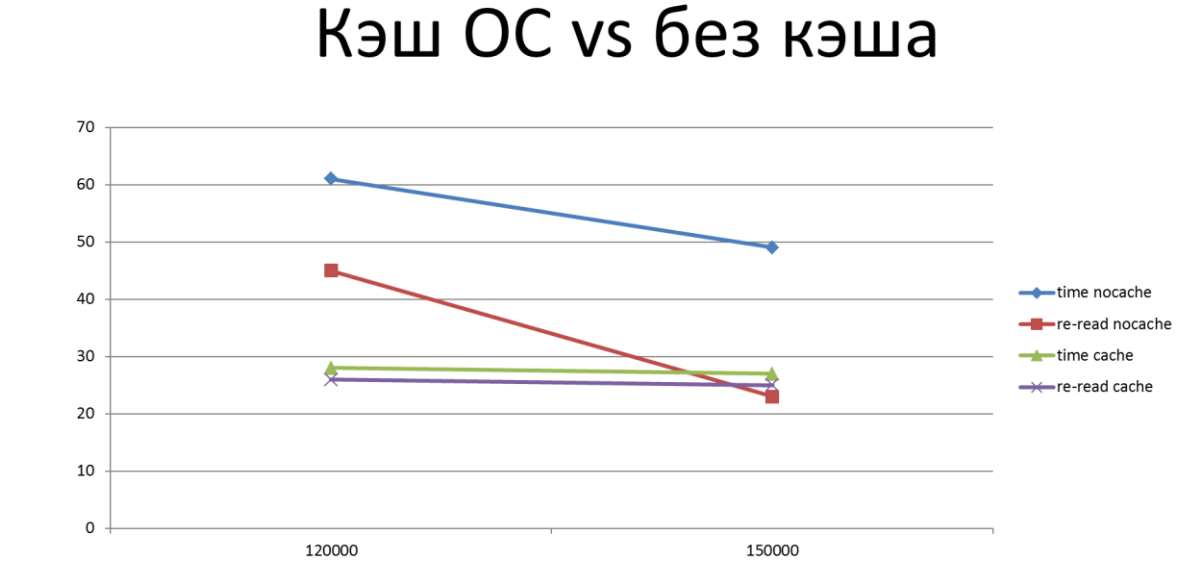


Рис. 10

Видно, что повторное выполнение запроса при отсутствии кэша ОС имеет гораздо больший эффект – практически двукратный при кэше в 150 тысяч страниц. И скорость повторного выполнения запроса даже выше, чем с включенным кэшем ОС – 23 секунды против 25 секунд. Видимо, сэкономленные 2 секунды – это оверхед на обращения к файловому кэшу операционной системы.

## Итоги

- SuperServer выгоднее Classic и SuperClassic, поскольку позволяет указать больший кэш
- Чем больше кэш – тем лучше
- Кэш файловой системы существенно помогает
- Если БД можно «поместить» в кэш Firebird, то лучше это сделать

Понятно, что распараллеливаемый SuperServer Firebird 3 выгоднее Classic и SuperClassic – у них нельзя указать большой кэш, а большой кэш явно увеличивает производительность.
Раз уж вы используете SuperServer, то он должен быть 64разрядным, и кэш желательно указать побольше. При этом нужно учитывать количество оперативной памяти на сервере. Например, если на сервере 32 гигабайта ОЗУ, то под кэш можно выделить 15-16 гигабайт, а остальное пусть использует операционная система под файловый кэш (который пойдет и на кэширование файла базы данных, и на сортировки).
Если же ваша база данных поместится в такой кэш Firebird целиком, то можно отключить файловый кэш параметром FileSystemCacheThreshold – он сработает только в отношении баз данных, а временные файлы будут точно так же продолжать кэшироваться в памяти операционной системой. Но лучше так делать только если размер базы данных не превышает примерно 40% размера оперативной памяти сервера.
В общем, путём правильного конфигурирования вы сможете извлечь из вашего сервера и Firebird максимум производительности.

И, финальные результаты по планам и кэшу, как напоминание

**PLAN (TABLE ORDER A)**
**Execute time = 1m 12s**
**Buffers = 2 048**
**Reads = 3 627 028**
**Fetches = 32 224 797**

**PLAN (TABLE ORDER A)**
**Execute time = 27s 518ms**
**Buffers = 150 000**
**Reads = 124 663**
**Fetches = 32 224 797**

**PLAN SORT ((A NATURAL))**
**Execute time = 35 s**
**Buffers = 2 048**
**Reads = 119 915**
**Fetches 14 767 524**

## Дополнительные материалы

- Оригинальное видео - https://youtu.be/vZUj0j156dg
- Руководство по аппаратному обеспечению для Firebird http://www.ibase.ru/files/firebird/Firebird_Hardware_Guide_2015_rus.pdf
- Эффективное использование памяти СУБД Firebird - https://youtu.be/E1EzmSYPRgc
- Архитектуры Firebird - https://youtu.be/ewrNLNDOtiE
- Оптимизатор Firebird 3 и планы запросов - https://youtu.be/0KITHwMNDtw

- Руководство по языку SQL СУБД Firebird 2.5 - http://www.ibase.ru/files/firebird/Firebird_2_5_Language_Reference_RUS.pdf
- Руководство по языку SQL СУБД Firebird 3.0 - http://www.ibase.ru/files/firebird/Firebird_3_0_Language_Reference_RUS.pdf
- Все статьи на iBase.ru – http://www.ibase.ru
- Все видео по Firebird и InterBase - https://www.youtube.com/user/ibdeveloper/
- Официальный сайт Firebird – https://www.firebirdsql.org