Что нового в Firebird 5.0. Параллельные возможности

Симонов Денис

Version 1.0 от 04.02.2024

Этот материал был создан при поддержке и спонсорстве компании iBase.ru, которая разрабатывает инструменты Firebird SQL для предприятий и предоставляет сервис технической поддержки для Firebird SQL.

Предисловие

Недавно вышел релиз Firebird 5.0. Это восьмой основной выпуск СУБД Firebird, разработка которого началась в мае 2021 года.

В Firebird 5.0 команда разработчиков сосредоточила свои усилия на повышение производительности СУБД в различных аспектах, таких как:

- параллельное выполнение для распространённых задач: backup, restore, sweep, создание и перестроение индекса;
- улучшение масштабирования в многопользовательской среде;
- ускорение повторной подготовки запросов (кеш компилированных запросов);
- улучшение оптимизатора;
- улучшение алгоритма сжатия записей;
- поиск узких мест с помощью плагина профилирования.

Поскольку объём материала довольно большой, то я разделю описание новых функций на несколько частей:

- улучшение в оптимизаторе запросов;
- новые возможности в языке SQL Firebird 5.0;
- параллелизм и другие функции появившиеся в Firebird 5.0;
- поиск узких места с помощью плагина PSQL профилирования.

В прошлых частях я рассказал о новых возможностях языка SQL и улучшениях оптимизатора. Теперь я расскажу о параллельном выполнении административных задач в Firebird 5.0.

1. Параллельное выполнение задач

Начиная с версии 5.0 Firebird может выполнять некоторые задачи, используя несколько потоков параллельно. Часть этих задач использует параллелизм на уровне ядра Firebird, другие реализованы непосредственно в утилитах. В настоящее время на уровне ядра реализовано параллельное выполнение задач очистки (sweep) и создания индекса. Параллельное выполнение поддерживается как для автоматической, так и для ручной очистки (sweep).

В следующих версиях планируется добавить параллелизм при выполнении SQL запроса.

1.1. Параллельное выполнение задач в ядре Firebird

Для обработки задачи с несколькими потоками движок Firebird запускает дополнительные рабочие потоки и создает внутренние рабочие соединения. По умолчанию параллельное выполнение отключено. Существует два способа включить параллелизм в пользовательском соединении:

- Установить количество параллельных рабочих процессов в DPB, используя тег `isc_dpb_parallel_workers`;
- Установить количество параллельных рабочих процессов по умолчанию с помощью параметра `ParallelWorkers` в `firebird.conf`.

Некоторые утилиты (`gfix`, `gbak`) поставляемые с Firebird имеют ключ командной строки `-parallel` для установки количества параллельных рабочих процессов. Зачастую этот переключатель просто передаёт количество рабочих процессов через тег `isc_dpb_parallel_workers` при соединении с базой данных.

Новый параметр `ParallelWorkers` в `firebird.conf` устанавливает количество параллельных рабочих процессов по умолчанию, которые могут использоваться любым пользовательским соединением, выполняющим распараллеливаемую задачу. Значение по умолчанию равно 1 и означает отсутствие использования дополнительных параллельных рабочих процессов. Значение в DPB имеет более высокий приоритет, чем значение в `firebird.conf`.

Для контроля количества дополнительных рабочих процессов (workers), которые может создать движок, в `firebird.conf` есть две новые настройки:

ParallelWorkers

Устанавливает количество параллельных рабочих процессов по умолчанию, используемых пользовательскими соединениями. Может быть переопределено для соединения путем использованием тега `isc_dpb_parallel_workers` в DPB.

MaxParallelWorkers

Ограничивает максимальное количество одновременно используемых рабочих процессов для данной базы данных и процесса Firebird.

Внутренние рабочие соединения создаются и управляются самим движком Firebird. Движок

поддерживает пулы рабочих соединений для каждой базы данных. Количество потоков в каждом пуле ограничено значением параметра `MaxParallelWorkers`. Пулы создаются каждым процессом Firebird независимо.

В архитектуре SuperServer рабочие соединения реализованы как облегченные системные соединения, а в Classic и SuperClassic они выглядят как обычные пользовательские соединения. Все рабочие соединения автоматически создаются сервером при необходимости. Таким образом, в классических архитектурах нет дополнительных серверных процессов. Дополнительные рабочие присутствуют в таблицах мониторинга. Неработающие рабочие соединения уничтожаются через 60 секунд бездействия. Кроме того, в классических архитектурах рабочие соединения уничтожаются сразу после того, как последнее пользовательское соединение отключается от базы данных.

1.1.1. Практические рекомендации по установке параметров

Параметр `MaxParallelWorkers` позволяет администраторам ограничить максимальное количество одновременно используемых рабочих процессов. Это может полезно для того, чтобы администратор запуская свои задачи был ограничен в "аппетитах". Он может указать слишком большое значение в переключателе `-parallel`, и мешать другим пользователям работать с базой данных, создавая излишнюю нагрузку.

Для архитектуры SuperServer я рекомендую ставить значения параметра `MaxParallelWorkers` равным количеству физических ядер вашего процессора (или всех процессоров) или установить рваным 64 (максимальное значение). Для классической архитектуры `MaxParallelWorkers` надо ставить меньшим равным количеству физических ядер вашего процессора (мы рекомендуем ставить половину от общего количества ядер). В архитектуре Classic ограничение `MaxParallelWorkers` работает для каждого процесса.

Значение параметра `ParallelWorkers` должно быть меньшим или равным значению `MaxParallelWorkers`. У вас может возникнуть соблазн установить `ParallelWorkers = MaxParallelWorkers`, но в этом случае надо учитывать, что вы работаете с базой данных не один. Кроме того, если у вас включен автоматических sweep, то при старте он заберёт почти все ресурсы у других работающих клиентский соединений. Лучше установить `ParallelWorkers` не более половины от `MaxParallelWorkers`, а при необходимости переопределять число рабочих потоков через `isc_dpb_parallel_workers` или переключатель `-parallel` в утилитах.

Все зависит от того, делается это под нагрузкой или нет. Если я восстанавливаю базу данных, то в этот момент с ней точно никто не работает, поэтому я могу запросить максимум параллельных потоков. А вот если вы делаете backup, sweep или создание индекса под рабочей нагрузкой, то вам нужно умерить свои аппетиты.

Далее я покажу как параллелизм влияет на время выполнения построения или перестроения индекса. Влияние параллелизма на автоматический sweep показано не будет, поскольку она стартует автоматически без нашего участия. Влияние параллелизма на ручной sweep будет продемонстрировано при рассмотрении выполнения задач утилитами Firebird.

1.1.2. Параллелизм при создании или перестройке индекса

Параллелизм может быть использован при создании или перестроении индексов.

Сравним скорость при создании индекса для таблицы WORD_DICTIONARY, содержащей 4079052 записей. Для чистоты эксперимента перед новым тестом перезагружаем службу Firebird. Кроме того, для того чтобы таблица была в страничном кеше выполняем

```
SELECT COUNT(*) FROM WORD_DICTIONARY;
```

Как известно построение индекса всегда происходит по COMMIT. В примерах я не даю команду COMMIT, поскольку выполнение происходит в isql, с включенным SET AUTODDL.

Запрос для создания индекса выглядит следующим образом:

```
CREATE INDEX IDX_WORD_DICTIONARY_NAME ON WORD_DICTIONARY (NAME);
```

Статистика выполнения этого запроса с ParallelWorkers = 1 выглядит следующим образом:

```
Current memory = 2577810256
Delta memory = 310720
Max memory = 3930465024
Elapsed time = 6.798 sec
Buffers = 153600
Reads = 11
Writes = 2273
Fetches = 4347093
```

Теперь удалим этот индекс, установим в конфиге ParallelWorkers = 4 и MaxParallelWorkers = 4 и перезапустим сервер. Статистика для выполнения того же запроса выглядит так:

```
Current memory = 2580355968
Delta memory = 2856432
Max memory = 4157427072
Elapsed time = 3.175 sec
Buffers = 153600
Reads = 11
Writes = 2277
Fetches = 4142838
```

Как видите время создания индекса уменьшилось в 2 с небольшим раза.

Тоже самое происходит при перестроении индекса запросом:

```
ALTER INDEX IDX_WORD_DICTIONARY_NAME ACTIVE;
```

1.2. Параллельное выполнение задач утилитами Firebird

Некоторые утилиты (`gfix`, `gbak`) поставляемые с Firebird тоже поддерживает параллельное выполнение задачи. Они используют количество параллельных рабочих процессов установленное в параметре `ParallelWorkers` в `firebird.conf`. Количество параллельных рабочих процессов можно переопределить используя ключ командной строки `-parallel`.

Я рекомендую всегда устанавливать количество параллельных процессов явно через переключатель `-parallel` или `-par`.

Параллелизм в утилитах Firebird поддерживается для следующих задач:

- Создание резервной копии с помощью утилиты `gbak`
- Восстановление из резервной копии с помощью утилиты `gbak`
- Ручной sweep с помощью утилиты `gfix`
- Обновление icu с помощью утилиты `gfix`

1.2.1. Параллелизм при выполнении резервного копирования с помощью утилиты `gbak`

Давайте посмотрим как параллелизм влияет на резервное копирование утилитой `gbak`. Естественно я буду использовать самый быстрый вариант резервного копирования через менеджер сервисов и с отключенной сборкой мусора. Для того чтобы можно было отследить время каждой операции во время резервного копирования добавим переключатель `-stat td`.

Сначала запустим резервное копирования без параллелизма:

```
gbak -b -g -par 1 "c:\fbdata\db.fdb" "d:\fbdata\db.fbk" -se localhost/3055:service_mgr
-user SYSDBA
  -pas masterkey -stat td -v -Y "d:\fbdata\5.0\backup.log"
```

Резервное копирование завершилось за 35.810 секунд.

А теперь попробуем запустить резервное копирование с использованием 4 потоков.

```
gbak -b -g -par 4 "c:\fbdata\db.fdb" "d:\fbdata\db.fbk" -se localhost/3055:service_mgr
-user SYSDBA
  -pas masterkey -stat td -v -Y "d:\fbdata\5.0\backup-4.log"
```

Резервное копирование завершилось за 18.267 секунд.

Как видите при увеличении количества параллельных обработчиков скорость резервного копирования растёт, хотя и не линейно.

На самом деле влияние параллельных потоков на скорость резервного копирования зависит от вашего железа. Оптимальное число параллельных потоков следует подбирать экспериментально.

Любые дополнительные переключатели тоже могут изменить картину. Так например переключатель `-ZIP`, сжимающий резервную копию может свести параллелизм на нет, а может всё ещё давать ускорение копирования. Это зависит от скорости дискового накопителя, производится ли копия на тот же диск где лежит база данных и других факторов. Поэтому необходимо проводить эксперименты именно на вашем железе.

1.2.2. Параллелизм при выполнении восстановления из резервной копировании с помощью утилиты `gbak`

Теперь давайте посмотрим как параллелизм влияет на скорость восстановления из резервной копии. Восстановление из резервной копии состоит из следующих этапов:

- создании базы данных с соответствующей ODS;
- восстановление метаданных из резервной копии;
- восстановлении данных пользовательских таблиц;
- построение индексов.

Параллелизм будет задействован только на двух последних этапах.

Для того чтобы можно было отследить время каждой операции во время восстановления из резервной копии добавим переключатель `-stat td`.

Сначала запустим восстановление из резервной копии без параллелизма:

```
gbak -c -par 1 "d:\fbdata\db.fbk" "c:\fbdata\db.fdb" -se localhost/3055:service_mgr
-user SYSDBA
  -pas masterkey -v -stat td -Y "d:\fbdata\restore.log"
```

Восстановление из резервной копии завершилось за 201.590 секунд. Из них 70.73 секунды ушло на восстановление данных таблиц и 121.142 секунды на построение индексов.

А теперь попробуем запустить восстановление из резервной копии с использованием 4 потоков.

```
gbak -c -par 4 "d:\fbdata\db.fbk" "c:\fbdata\db.fdb" -se localhost/3055:service_mgr
-user SYSDBA
  -pas masterkey -v -stat td -Y "d:\fbdata\restore-4.log"
```

Восстановление из резервной копии завершилось за 116.718 секунд. Из них 26.748 секунды ушло на восстановление данных таблиц и 86.075 секунды на построение индексов.

С помощью 4 параллельных рабочих нам удалось увеличить скорость восстановления почти в два раза. При этом скорость восстановления данных выросла почти в 3 раза, а построение индексов ускорилось в 1.5 раза.

Это объясняется довольно просто, индексы не восстанавливаются параллельно, параллелизм используется только при построении больших индексов. Справочные таблицы обычно небольшие, индексы на них маленькие, а количество таких таблиц может быть большим. Поэтому на вашей базе данных цифры могут быть другими.

Обратите внимание на то, что параметр `MaxParallelWorkers` ограничивает использование параллельных потоков только для ядра Firebird. При восстановлении базы данных утилитой `gbak`, вы можете наблюдать следующую картину: данные в таблицах восстанавливаются быстро (параллелизм заметен), а построение индексов происходит медленно. Дело в том, что индексы всегда строятся ядром Firebird. И если в `MaxParallelWorkers` будет значение меньшее указанное в `-parallel`, то для построения индексов будет использоваться только `MaxParallelWorkers` потоков. Однако данные в таблицы заливает сам `gbak`, используя при этом `-parallel` рабочих потоков.

1.2.3. Параллельный ручной sweep с помощью утилиты `gfix`

sweep (чистка) - это важный процесс обслуживания: Firebird сканирует базу данных, и если в ней присутствуют "мусорные" версии записей удаляет их со страниц данных и из индексов. Основная цель запуска sweep - это подвинуть "вверх" номер Oldest Interesting Transaction (Oldest transaction в gstat -h).

До Firebird 3.0 sweep всегда сканировал все страницы данных. Однако начиная с Firebird 3.0 (ODS 12.0) на страницах данных (DP) и на страницах указателей на страницы данных (PP) есть специальный swept flag, который устанавливается в 1, если sweep уже просмотрел страницу данных и вычистил с неё мусор. При первой модификации записей на этот таблице флаг снова сбрасывается в 0. Начиная с Firebird 3.0 автоматический и ручной sweep пропускает страницы у которых swept флаг равен 1. Поэтому повторный sweep будет проходить намного быстрее, если конечно с момента предыдущего sweep вы не успели поменять записи на всех страницах данных базы данных.

Новые страницы данных всегда создаются с swept flag = 0. При восстановлении базы данных и резервной копии все страницы DP и PP будут с swept flag = 0.

Как правильно тестировать? Холостой sweep после восстановления из резервной копии не дал разницы в однопоточном и многопоточном режиме. Поэтому я сначала провёл на восстановленной БД sweep для того, чтобы следующий sweep не проверял незамусоренные страницы, а потом сделал запрос вроде такого:

```
update bigtable set field=field;
rollback;
exit;
```

Целью этого запроса было создания мусора в базе данных. Теперь можно запускать sweep для тестирования скорости его выполнения.

Сначала запустим sweep без параллелизма:

```
gfix -user SYSDBA -password masterkey -sweep -par 1 inet://localhost:3055/mydb
```

```
DESKTOP-E3INAFT Sun Oct 22 16:24:21 2023
    Sweep is started by SYSDBA
    Database "mydb"
    OIT 694, OAT 695, OST 695, Next 696


DESKTOP-E3INAFT Sun Oct 22 16:24:42 2023
    Sweep is finished
    Database "mydb"
    1 workers, time 20.642 sec
    OIT 696, OAT 695, OST 695, Next 697
```

Теперь снова делаем обновление большой таблицы и rollback, и запустим sweep с 4 параллельным рабочими.

```
gfix -user SYSDBA -password masterkey -sweep -par 4 inet://localhost:3055/mydb
```

```
DESKTOP-E3INAFT Sun Oct 22 16:26:56 2023
    Sweep is started by SYSDBA
    Database "mydb"
    OIT 697, OAT 698, OST 698, Next 699


DESKTOP-E3INAFT Sun Oct 22 16:27:06 2023
    Sweep is finished
    Database "mydb"
    4 workers, time 9.406 sec
    OIT 699, OAT 702, OST 701, Next 703
```

Как видите скорость выполнения sweep выросла в 2 с лишним раза.

1.2.4. Параллельное обновление icu с помощью утилиты `gfix`

Переключатель `-icu` позволяет перестроить индексы в базе данных с использованием нового ICU.

Дело в том, что библиотека ICU используется Firebird для поддержки COLLATION для многобайтных кодировок вроде UTF8. В Windows ICU всегда поставляется в комплекте с Firebird. В Linux же ICU обычно является системной библиотекой. При переносе файла базы данных с одного дистрибутива Linux на другой, ICU установленная в системе может иметь разную версию. Это может привести к тому, что база данных в ОС, где установлена другая версия ICU, окажется бинарно несовместимой для индексов символьных типов данных.

Поскольку перестройка индексов может быть выполнена с использованием параллелизма, то и для `gfix -icu` это тоже поддерживается.

2. Заключение

В этой части мы рассмотрели параллельные возможности инструментов Firebird 5. Если вы хотите узнать больше, как gbak реализует параллельное чтение данных или даже реализовать аналогичный механизм в своем приложении, прочтите подробную статью "Параллельное чтение данных в Firebird".

Мы рассказали ещё не про все новые возможности Firebird 5.0, следите за обновлениями!