Multi-thread sweep, backup and restore
Firebird Conference 2019
Berlin, 17-19 October
Introduction

• Big demand from users to speed up most time consuming regular maintenance operations:
  • Backup
  • Restore
  • Sweep
• Initial implementation based on Firebird 2.5 Classic
  • Firebird 2.5 Super Server is not suitable
• Front ported to the v3 codebase
  • Including Super Server, of course
• Available in HQbird 2020 (for Firebird 2.5 and 3.0)
• Will be included into Firebird 4+
Introduction

• The good parallel implementation should, at least
  • Evenly distribute workload between workers
  • Avoid or minimize possible contentions for shared resources (disk, memory, internal locking)
  • Minimize necessary coordination between workers and task manager
Sweep

• How sweep works
  • Read each table in database
  • Cleanup unneeded record versions
  • Move OIT marker on success
Sweep

• What can be run in parallel?
  • Each parallel worker could handle (read and cleanup) separate table
• What if there is few big tables and many small tables?
Sweep

- What if there is few big tables and many small tables?
- Big table could be handled by few parallel workers
Sweep

- How to divide big table between few workers to minimize contention and coordination?
  - Every worker could handle one data page and then ask for a next (not handled) one
    - Almost fair distribution of workload
    - No contention for the same data pages
    - Some contention for the same pointer page
    - Coordinate with manager very often
Sweep

• How to divide big table between few workers to minimize contention and coordination?
  • Every worker could handle few data pages and then ask for a next (not handled) few pages
    - How much?
How to divide big table between few workers to minimize contention and coordination?

Every worker handle data pages from the same pointer page and then ask for a next (not handled) pointer page

- Workload distribution still fair enough
- No contention for the same data pages
- No contention for the same pointer page
- Coordinate with manager not too often
Sweep

• Implementation details
  • Single attachment can’t be handled by concurrent threads simultaneously
  • Every worker have its own private attachment and transaction
  • Internal pool of worker attachments
    – Per database and per server process
    – Limited by value of new configuration setting $MaxParallelWorkers$
      – Created automatically when required
      – Works in the same server process
      – Closed automatically when last connection to the database is gone
Sweep

• Usage
  • `gfix -sweep -parallel 4 <database>`
    - Run sweep using 4 parallel attachments
      • 1 user attachment and 3 additional worker attachments
  • New DPB tag
    `isc_dpb_parallel_workers`
Sweep

• Usage
  • Auto-sweep also could run in parallel mode
    – New configuration setting *ParallelWorkers*
Test results

- Big database

---

<table>
<thead>
<tr>
<th>Test environment 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Firebird version</strong></td>
</tr>
<tr>
<td>2.5.9 HQBird</td>
</tr>
<tr>
<td><strong>OS</strong></td>
</tr>
<tr>
<td>CentOS 6.7</td>
</tr>
<tr>
<td><strong>Server</strong></td>
</tr>
<tr>
<td>ProLiant DL380 Gen9</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
</tr>
<tr>
<td>2 x Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz</td>
</tr>
<tr>
<td><strong>Cores per socket</strong></td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td><strong>Logical CPU's</strong></td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td><strong>RAM</strong></td>
</tr>
<tr>
<td>96 GB</td>
</tr>
<tr>
<td><strong>HDD</strong></td>
</tr>
<tr>
<td>4xHDD SAS 15k RAID 10</td>
</tr>
<tr>
<td><strong>Database</strong></td>
</tr>
<tr>
<td>510 GB</td>
</tr>
</tbody>
</table>
Sweep

- Test results
- Big database

![Sweep time, Firebird 2.5 SC](chart.png)
Backup

• How backup works
  • Read system tables and store user metadata in backup file
  • Read user tables and store records in backup file
Backup

- What can be run in parallel?
  - Parallel workers could read database independently, but backup file should be written in correct order
    - Serialize workers when backup file is written
Backup

- What can be run in parallel?
  - Parallel workers could read database independently, but backup file should be written in correct order
    - Move all write activity into another dedicated thread
Backup

• What can be run in parallel?
  • Read and store metadata
    - Could be done but
      • It will significantly complicate code
      • Amount of metadata usually much less than size of user data
Backup

• What can be run in parallel?
  • Read and store user data
    – Handle different tables by parallel workers
      • Backup file will contain mix of records from different tables
      • Requires change in backup file structure to allow restore to handle such file
      • “Big table” problem as in sweep case
Backup

• What can be run in parallel?
  • Read and store user data
    – Parallel workers should handle different parts of the same table
    – Requires a way to split table by parts
      • Ideally parts of the equal size
Backup

• How to split table for few parallel workers?
  • Use ranges of primary\unique key values
    - Not every table could have primary\unique key
    - Unknown in advance whole range of key values
    - Uneven distribution of key values
    - How to make ranges for character keys?
    - How to make ranges for composite (multi-segment) keys?
Backup

- How to split table by few parallel workers?
  - Use ranges of data pages
    - `gbak` works “outside” of the engine, it can’t address data pages directly
  - Use ranges of `RDB$DB_KEY` values
    - Engine supports equality comparison only for `RDB$DB_KEY`
    - Application (gbak) have no idea what data page is addressed by given `RDB$DB_KEY` value
    - Need some support from the engine side
Backup

• Use ranges of RDB$DB_KEY values
  • New built-in function MAKE_DBKEY
    - MAKE_DBKEY(relation_id, recnum)
      • Returns dbkey for record recnum
    - MAKE_DBKEY(relation_id, recnum, dpnum)
      • Returns dbkey for recnum at data page dpnum
    - MAKE_DBKEY(relation_id, recnum, dpnum, ppnum)
      • Returns dbkey for recnum at data page dpnum at pointer page ppnum
  • Engine now supports all kind of comparisons with RDB$DB_KEY (%<, %<=, %>, %>=, %=, %!=)
Backup

• How to split table for few parallel workers?
  • Every worker handle records from the data pages from the same pointer page and then ask for a next (not handled) pointer page

```sql
SELECT * FROM TABLE
WHERE RDB$DB_KEY >= MAKE_DBKEY(:relId, 0, 0, :ppNum)
  AND RDB$DB_KEY < MAKE_DBKEY(:relId, 0, 0, :ppNum + 1)
```
Backup

• Backup consistency
  • gbak uses snapshot transaction to read user data in consistent way
  • Every worker uses own attachment and transaction
  • All worker attachments should read the same data despite of other activity in database
  • Need shared database snapshot
Backup

- Shared database snapshot
  - First introduced in Firebird 4 beta
    - Based on new database snapshots architecture using commits order
  - Re-implemented for Firebird 2.5 and Firebird 3 specially to support parallel backup
  - Follows the same interface as of Firebird 4
Backup

• Usage
  • `gbak -b -parallel 4 <database> <backup>`
Backup

- Test results
- Big database

Backup time, Firebird 2.5 SC

<table>
<thead>
<tr>
<th>Workers</th>
<th>Time, min</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>364</td>
</tr>
<tr>
<td>2</td>
<td>215</td>
</tr>
<tr>
<td>4</td>
<td>142</td>
</tr>
<tr>
<td>8</td>
<td>99</td>
</tr>
<tr>
<td>16</td>
<td>83</td>
</tr>
<tr>
<td>24</td>
<td>76</td>
</tr>
<tr>
<td>32</td>
<td>70</td>
</tr>
<tr>
<td>48</td>
<td>59</td>
</tr>
<tr>
<td>64</td>
<td>56</td>
</tr>
</tbody>
</table>
Backup

- Test results
  - Medium database

<table>
<thead>
<tr>
<th>Test environment 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Firebird version</td>
<td>2.5.9 HQBird, 3.0.5 HQBird</td>
</tr>
<tr>
<td>OS</td>
<td>CentOS 6.7</td>
</tr>
<tr>
<td>Server silence</td>
<td>ProLiant DL380 Gen9</td>
</tr>
<tr>
<td>CPU</td>
<td>2 x Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz</td>
</tr>
<tr>
<td>Cores per socket</td>
<td>6</td>
</tr>
<tr>
<td>Logical CPU’s</td>
<td>24</td>
</tr>
<tr>
<td>RAM</td>
<td>32 GB</td>
</tr>
<tr>
<td>HDD</td>
<td>4xHDD SAS 10k RAID 10</td>
</tr>
<tr>
<td>Database</td>
<td>42 GB</td>
</tr>
</tbody>
</table>
Backup

• Test results
  • Medium database

Backup time, Firebird 2.5 SC

<table>
<thead>
<tr>
<th>Workers</th>
<th>Time, sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>961</td>
</tr>
<tr>
<td>2</td>
<td>601</td>
</tr>
<tr>
<td>4</td>
<td>394</td>
</tr>
<tr>
<td>8</td>
<td>330</td>
</tr>
<tr>
<td>12</td>
<td>308</td>
</tr>
</tbody>
</table>
Restore

• How restore works
  • Create new database
  • Read metadata and populate system tables
  • Read data and populate user tables
  • Activate (build) indices
Restore

• What can be run in parallel?
  • Create new database
    – no
  • Read metadata and populate system tables
    – not practical
  • Read data and populate user tables
    – yes
    – probably, requires changes in backup format
    – not now, sorry
  • Activate (build) indices
    – yes, exactly
Restore

• How indices are build at restore
  • Index metadata is created with table metadata
    - Indices are created with `DEFERRED_ACTIVE` flag
  • Indices are activated (build) after all user data is committed
  • Index is actually build at transaction commit
  • Every index is activated in separate transaction
Index build

• Index build steps
  • Read table data
    – Remove unneeded record versions (garbage collect)
    – Put index keys into the sorter
  • Build index b-tree using already sorted data
Index build

- What can be run in parallel?
  - Read table and sort index keys
    - Yes
  - Build index B-tree
    - Non-trivial task: prefix compression of index keys
    - Not now, maybe later
Index build

• What can be run in parallel?
  • Read table and sort data
  • Every worker handle records of data pages from the same pointer page and then ask for a next (not handled) pointer page
  • Every worker have its own attachment, transaction and sorter
• On the “B-tree build” step data from all sorters are merged into common sorted stream
  • By single thread
Index build

- What can be run in parallel?
• Restore with parallel index build

Restore

gbak: opened file … .fbk
gbak: created database ...
gbak: restoring ...
gbak: committing metadata

gbak: restoring index …
gbak: restoring data for table …
gbak: committing metadata

gbak: activating and creating deferred index …
gbak: activating and creating deferred index …
gbak: finishing, closing, and going home

Worker 1
Worker 2
Worker 3
Worker 4
What can be improved next?

- Parallel load of user data into database
  - Backup file format could be changed
- Create few indices simultaneously at one table scan
  - Temporary space usage could be significantly increased
Restore

• Usage
  • `gbak -c -parallel 4 <backup> <database>`
  • Any application
    – DPB tag `isc_dpb_parallel_workers`
      • instruct engine how many parallel workers could be used for some tasks
      • currently index creation and auto-sweep supports such parallel handling
Index build

• Usage
  • Regular *CREATE INDEX* and *ALTER INDEX ACTIVE* statements also could build index with parallel workers
    - Configuration setting *ParallelWorkers*
    - DPB tag *isc_dpb_parallel_workers*
Restore

- Test results
- Big database

![Restore time, Firebird 2.5 SC](chart.png)
Restore

- Test results
  - Medium database

![Restore time, Firebird 2.5 SC](image)

- Workers
- 1: 1373 index, 1559 data, 2932 all
- 2: 930 index, 1556 data, 2486 all
- 4: 689 index, 1569 data, 2258 all
- 8: 631 index, 1570 data, 2201 all
- 12: 640 index, 1561 data, 2201 all

---

Berlin 2019

Firebird 4
Restore

- Test results
  - Medium database

![Restore time, Firebird 3 SS](image)

<table>
<thead>
<tr>
<th>Workers</th>
<th>Index</th>
<th>Data</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1423</td>
<td>1698</td>
<td>3121</td>
</tr>
<tr>
<td>2</td>
<td>1011</td>
<td>1710</td>
<td>2721</td>
</tr>
<tr>
<td>4</td>
<td>827</td>
<td>1701</td>
<td>2528</td>
</tr>
<tr>
<td>8</td>
<td>721</td>
<td>1720</td>
<td>2441</td>
</tr>
<tr>
<td>12</td>
<td>726</td>
<td>1694</td>
<td>2420</td>
</tr>
</tbody>
</table>

Legend:
- Red: index
- Blue: data
- Orange: all
• Test results
  • Medium database

![Restore time, Firebird 3 SC](chart)

<table>
<thead>
<tr>
<th>Workers</th>
<th>Time, sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1619</td>
</tr>
<tr>
<td>2</td>
<td>1611</td>
</tr>
<tr>
<td>4</td>
<td>1625</td>
</tr>
<tr>
<td>8</td>
<td>1610</td>
</tr>
<tr>
<td>12</td>
<td>1610</td>
</tr>
</tbody>
</table>

- **Index**: red
- **Data**: blue
- **All**: red + blue
All together

- Firebird now could run tasks using multiply workers/threads
- Some tasks used parallelism built into engine
  - Sweep
  - Index build, gbak -restore
- Some tasks used parallelism “outside” of the engine
  - gbak -backup
- This list will be enhanced
  - Validation, Statistics
  - Query execution
All together

- firebird.conf, per database settings
  - *MaxParallelWorkers*
    - Set maximum number of parallel workers per Firebird process
  - *ParallelWorkers*
    - Set default number of parallel workers used to run some task
- DPB tag
  - *isc_dpb_parallel_workers*
    - Set number of parallel workers used to run some task by current attachment (overrides *ParallelWorkers* setting)
THANK YOU FOR ATTENTION

Questions?

Firebird official web site

Firebird tracker

hvlad@users.sf.net