FTS scalability improvements and plans

Edward Karavakis
on behalf of the FTS team

Outline

- Scheduler Improvements
- Future work

Scheduler Improvements

Motivation

- ATLAS pushing for a single FTS instance
- Scheduler’s performance is affected when there are many links and many submitted transfers – takes a lot of time to make a decision
- Basic SELECT queries can take tens of seconds to return and everything seems bloated and stuck

Methodology

- FTS3-DEVEL-NEXT cluster (3 VMs) and DBoD (identical to FTS-ATLAS)
- Downloaded the slow queries log file for several days from the DBoD portal
- Examined all the slow queries taking more than 10 seconds to run
  - Passing them from a MySQL query profiler
  - Analysing performance and cost of the query
  - Investigating the use (or not) of an index and if it's the appropriate one
- Adding missing indices and minor optimisations to the schema

Optimised queries

```
SELECT DISTINCT t_file.vo_name, t_file.source_se, t_file.dest_se FROM t_file INNER
JOIN t_job ON t_file.job_id = t_job.job_id WHERE t_file.file_state = ‘SUBMITTED’ AND
(t_file.hashed_id BETWEEN 0 AND 21844) AND t_job.job_type = ‘Y’ ;
```

"query_cost": "908713.52"
"data_read_per_join": "170M"

After adding two indices, one on file_state and one on job_type with:

```
ALTER TABLE `t_file` ADD INDEX `idx_state` (`file_state`) ;
ALTER TABLE `t_job` ADD INDEX `idx_jobtype` (`job_type`) ;
```

"query_cost": "3.24"
"data_read_per_join": "4K"

```
SELECT DISTINCT t_file.vo_name, t_file.source_se, t_file.dest_se FROM t_file INNER
JOIN t_job ON t_file.job_id = t_job.job_id WHERE t_file.file_state = ‘SUBMITTED’  AND
(t_file.hashed_id BETWEEN 45871 AND 52423) AND t_job.job_type = ‘Y’ ;
```

"query_cost": "909043.89"
"data_read_per_join": "938M"

After adding the job_type index:

"query_cost": "3.24"
"data_read_per_join": "4K"

Optimised queries (cont.)

```
SELECT DISTINCT t_file.vo_name, t_file.job_id  FROM t_file  INNER JOIN t_job ON
t_file.job_id = t_job.job_id  WHERE       t_file.file_state = 'SUBMITTED' AND
(t_file.hashed_id >= 13106 AND t_file.hashed_id <= 19658) AND      t_job.job_type =
'H' AND       (t_file.retry_timestamp IS NULL OR t_file.retry_timestamp < '2018-10-23
01:20:26');
```

"query_cost": "812493.93"

"data_read_per_join": "698M"

By using the new `idx_jobtype` index:

"query_cost": "3.24"

"data_read_per_join": "4K"

```
SELECT COUNT(*) FROM t_job  WHERE source_se = 'gsiftp://eosatlassftp.cern.ch' AND
dest_se = 'srm://grid-se.physik.uni-wuppertal.de'     AND job_type IN ('Y', 'H')
AND job_state = 'ACTIVE';
```

"query_cost": "42628.80"
"data_read_per_join": "3M"
using idx_link

Using the jobtype index that we added previously:

"query_cost": "4.81"
"data_read_per_join": "238"

Optimised queries (cont.)

```
SELECT COUNT(*) FROM `t_file` WHERE (`t_file`.`finish_time` >= '2018-11-01 00:00:49' AND
`t_file`.`transfer_host` = 'fts-devel-next001.cern.ch' );
```

"query_cost": "225269.41"
"data_read_per_join": "470M"
time to finish 3.425 seconds
using the idx_finish_time index on the date range.

An index on the transfer_host really makes sense as we only have a handful of hosts. After adding this index:

```
ALTER TABLE `t_file` ADD INDEX `idx_host` (`transfer_host`)    ;
```

"query_cost": "207489.60"
"data_read_per_join": "67M"
time to finish 0.432 seconds

```
delete from t_optimizer_evolution where datetime < (UTC_TIMESTAMP() - interval '6' DAY );
```

Performing a full table scan and locking the table, added an index on datetime:

```
ALTER TABLE `t_optimizer_evolution` ADD INDEX `idx_datetime` ( `datetime`)    ;
```

Schema optimisations

- Denormalise priority in DB *(suggested by Brian Bockelman)*
  - Expensive join between job and file tables
  - Priority replicated also in the file table
  - FTS REST was changed to also store the priority in the file

```
SELECT MAX(priority) FROM t_job, t_file WHERE t_file.job_id =
t_job.job_id AND t_file.vo_name=‘ABC' AND
t_file.source_se=‘AAA' AND t_file.dest_se=‘BBB' AND
t_file.file_state = 'SUBMITTED'
```

```
SELECT MAX(priority) FROM t_file WHERE vo_name=‘ABC' AND
source_se=‘AAA' AND dest_se=‘BBB' AND file_state = 'SUBMITTED';
```

Indexed

Stress testing

- 15 million files with 2000 links & duration of 5 mins
- Mock endpoints without staging
- Production VS new proposed schema (FTS 3.8 vs 3.9)
- Test was performed multiple times – numbers are averages

Results

Time for the scheduler to run

Files	3.8 (in minutes)	3.9 (in minutes)	Diff (in minutes)
1M	0:30	0:22	0:08
1.5M	0:34	0:24	0:10
3M	0:35	0:25	0:10
4.5M	0:46	0:31	0:15
5M	0:48	0:34	0:14
6M	0:49	0:36	0:13
6.7M	0:50	0:37	0:13
7M	1:06	0:41	0:25
7.5M	1:10	0:42	0:28
8M	1:14	0:43	0:31
8.7M	1:16	0:45	0:31
9M	1:18	0:47	0:31
9.5M	1:23	0:49	0:34
10M	1:50	0:50	1:00
15M	7:00	1:10	5:50

Observations

- All queries that were taking more than 10 seconds to run do not appear anymore in the log of slow queries
- Issues that still remain
  - Procedure cleaning historical data -> DB is loaded when it runs. The more files you have, the more it takes to clean
  - Problem with the WebMon breaking and not loading at around 17M submissions

Observations (cont.)

- Don't think we can have one FTS instance per experiment with the 3.9 improvements
  - More number of active transfers per instance, meaning more nodes so more load on the DB
  - More polling from Rucio and/or other clients (maybe moving clients to Messaging?)
  - More staging that was not considered in the tests but also impacts DB performance
- If issue is the diff configs of servers, maybe a central repo to share their configs will be ok?

Future work

Upcoming work for this year

- Measure the time it takes to upgrade DB schema from 3.8 to 3.9
- Repeat the same stress test with staging to check the impact
- Release 3.9 that will include these optimisations

- Further optimisations to be able to have a single instance per experiment
  - DB partitioning and /or separation of tables with only ACTIVE/QUEUED transfers
  - Dedicated FTS rest machines that will only handle the polling

- Implement protection to avoid duplicate submissions of the same file as requested by ATLAS

2020 and beyond

- Improvements currently performed focus on Run3
- Thinking about the future, we need an architectural change in order to handle the increased load of Run4
  - Move out of MySQL?
  - Investigate a queue technology like QuarkDB?
  - Have only one scheduler polling that will send transfers via messaging to different agents?

Questions?