What’s New in Alibaba’s X-DB SQL Engine

Min Qiu, Alibaba Group
Santa Clara, California | April 23th – 25th, 2018
Agenda

• Introduction to X-DB

• Features in X-DB SQL Engine
  • Query Plan Cache
  • Remote Execution
  • Distributed SQL Processing

• Future Work
Introduction to X-DB
What’s X-DB

• Alibaba home-made distributed database
  • MySQL Compatible syntax/interfaces/protocols
  • High performance at low cost
  • Horizontal scalability with high availability and fault tolerance
  • Data Strong consistency guarantee
  • Globally distributed
  • SW/HW co-designed
X-DB Architecture

Key Components:
- HA(X-Paxos)
- Global Meta Management (GMS)
- Storage Engine (X-Engine)
- Distributed SQL Engine

Features:
- Self-contained
- Decoupling storage and compute allowing separate scaling
- Multiple replicas allowing multiple reads
Features in X-DB SQL Engine
Features in X-DB SQL Engine

• Window Function
• Global Sequence
• Query Plan Cache
• Distributed Query Processing
Query Plan Cache

What and Why?

- Plan is cached to skip compilation efforts in subsequent runs
- Parameter bind is probably necessary
- Good for short-running queries
- Good for query plan insensitive to bind parameters
Query Plan Cache

How?

• Options
  - Full text match (case sensitive)
    `select * from t where c = 1` \(!= select * from t where c = 2`
  - Parameterized SQL template, i.e. Prepared Statement (PS)
    `select * from t where c = ?`

• Our Solution: Extension of existing PS solution
  - Queries from Alibaba online system fall into limited number of patterns
  - No extra work is required to parameterize SQL template
Query Plan Cache

• Problems with MySQL PS
  - *PS objects are saved in client connection, OOM risk*
  - No cache invalidation mechanism
  - *PS only saves parsing time, optimization is still needed*

• Our Enhancement
  - *Cache PS and plan related objects in worker threads*
    • JOIN
    • best_ref
    • QEP_TAB
    • ...
  - *Cache invalidation when capacity limit is hit*
Query Plan Cache

Implementation

• Generate query plan with first-run parameters
  - Good for cases where plan is insensitive to input parameters
  - Performance regression is possible if cached plan is suboptimal for specific parameters

• Parameter substitution
  - New parameter values should be mapped to correct location of relevant data structures

• Decouple execution from optimization

• Cache Management
  - New system variable `plan_cache_size` to control cache size on each worker thread
  - Use LRU to evict if memory limit is hit
Query Plan Cache

Something interesting

- Disable the optimization against const table
- Prevent Impossible Plan from being generated at the first run
  
  ```
  create table t (c1 INT primary key, c2 INT);
  insert into t values (1, 1);
  prepare stmt from 'select * from t where c1 = ?';
  set @a=NULL;
  execute stmt using @a;
  set @a=1;
  execute stmt using @a;
  ```

- Disallow the JOIN_TYPE to be set to ref for between predicate
  
  ```
  create table t (c1 INT primary key, c2 INT);
  insert into t values (1, 1), (2,2);
  prepare stmt from 'select * from t where c1 between ? and ?';
  set @a=1;set @b=1;
  execute stmt using @a, @b;
  set @a=1;set @b=2;
  execute stmt using @a, @b;
  ```
Query Plan Cache

Plan Cache Invalidation

- Cache schema version (i.e. V1) in PS when generating query plan
- Schema version changes (i.e. V2) as DDL is applied
- Raise a flag when a different schema version is detected at runtime
- Automatic re-prepare when invalidated, transparent to the user
Performance Evaluation

Configuration

- Sysbench
  - `sltp_read_only`
  - `select_random_points`
- 100 Tables
- 100K records per table
- 500 Client connections
Distributed Query Processing
Remote Execution

Architecture

• SQL Request Routing/Forwarding
  - Handle incorrect SQL routing

• Partition Location Identification

• Pass back result from remote

• External dependencies
  - RPC service
  - GMS/LMS
Remote Execution

Execution Scheduling

- Promise/Future async paradigm
- CONTEXT
  - Client communication protocol
  - THD context
- Privilege check skipped on node B
Remote Execution

Exception handling

• Only one forwarding is allowed

• Unsupported SQL request is prevent from execution
  - Query which touches data across multiple nodes
  - Transaction across multiple nodes

• Error message/code is overlaid to node initializing the remote execution

• Perform refresh operation if error is caused by out-of-date location cache
Distributed Execution

Architecture

- **Query Coordinator**
  - Accept SQL request
  - Generate distributed query plan
  - Split the execution pipeline(stage)
  - Schedule stage execution

- **Query Worker**
  - Execute pipeline with given control information
  - Redistribute data to next stage

- **Scheduler**
  - Pipeline dependency relation
Distributed Execution

Query Plan on Coordinator

- Generate a single-host query plan first

- Identify the node under which Exchange is inserted if necessary
  - Join
  - Sort
  - Group By

- Record the location where Exchange node is inserted (index of QEP_TAB array)
Distributed Execution

Query Plan on Coordinator

- **mm_tree** is built against query condition for **Partition Pruning**
  - *i.e. only t1p1, t1p3, t2p2, t2p4 left after partition pruning*

- **Partition Location Cache** at LMS tells which hosts hold those partitions
  - *Cache might be out-of-sync which can be detected at execution time*
  - *LMS is forced to refresh from GMS when out-of-sync is found*

- Table access operations are dispatched to corresponding hosts by RPC
Distributed Execution

Modification on optimizer

• Disable "const table" optimization when necessary

• Perform “lock table” operation as late as possible
  - Currently tables are locked in between prepare and optimize

• Avoid diving into storage engine during optimization
  - Row number estimate
  - Record in range estimate

• Use statistics stored in global catalog for cost estimation
Distributed Execution

Terminology

• Pipeline
  - A segment (set of operations) in the query plan tree
  - Receives input from upstream pipeline and generates output for downstream pipeline
  - The unit of scheduling and execution

• Stage
  - The process in which a pipeline execution is performed

• Task
  - A subset of a stage, which deals with a partition of data
  - A stage might be consist of multiple tasks
Distributed Execution

Pipeline Tree Generation

- Exchange node is the pipeline boundary

- A pair of In/Out nodes corresponding to each Exchange
  - Exchange Out node at Producer side
  - Exchange In node at Consumer side

- Pipeline dependency relationship is represented by the edge in the tree

- Pipeline tree is input to scheduler
Distributed Execution

Query Plan on Worker

- Plan on worker must be the same as that on coordinator
  - Pipeline generated on coordinator must match the same segment on worker

- Full SQL statement is compiled only once on worker
  - Plan is cached on worker with globally unique job ID as label
Distributed Execution

Same environment on Coordinator and Workers

- Environment on coordinator to generate query plan must be restored on workers

- Several categories of environment/context information
  - All system variables used by optimizer
  - All session variables used by optimizer
  - Statistics used by optimizer
    - table::file::stats.records
  - Constant values in cost_model objects
Distributed Execution

Two Level Scheduler

• Stage level scheduling
  - Satisfies dependency restriction of pipeline tree
  - Bottom up
  - JOIN::exec() is modified to be reentrant
    • Allow to execute a segment of QEP_TAB array with given start and end indexes.
    • Join_buffer/Temp_table must be ready before the segment starts to run

• Task level scheduling
  - Task is divided into a set of sub-task
  - Schedule sub-task to run in parallel
Distributed Execution

Examples

• Without Data Exchange
  - Simple query without group by/order by/subquery
  - Partition key is prefix of group by list
  - Partition key is prefix of order by list
  - Partition-aware Join
Distributed Execution

Examples

• With Data Exchange
  - *Broadcast*
  - *T1 has 3 partitions on 3 nodes*
  - *T2 has 1 partitions on 1 node*
  - *T2 is small*
Distributed Execution

Examples

• With Data Exchange
  - *Shuffle*
  - *T1 has 4 partitions on 2 nodes*
  - *T2 has 2 partitions on 2 nodes*
Future Work
Future Work

• Distributed Processing Enhancement
  - Support distributed query processing requiring data redistribution
  - Advanced Scheduling Options
    • Parallel scheduling for independent stages
    • Location/Resource based scheduling
  - Load Balancer
  - Admission Control

• Enhancement to optimizer
  - Data Redistribution selection
  - Automatic Parallelism selection

• Enhancement to executor
  - Hash Join
  - Sort Merge Join
About me

Min Qiu

• Staff Engineer at Alibaba Database Kernel Team
• Tech Lead of SQL Engine Team
• Contact: min.qiu@alibaba-inc.com
• LinkedIn: https://www.linkedin.com/in/min-qiu-088b5516/
Thank You!