Apache ShardingSphere: A Holistic and Pluggable Platform for Data Sharding

Ruiyuan Li, Liang Zhang, Juan Pan, Junwen Liu, Peng Wang, Nianjun Sun, Shanmin Wang, Chao Chen, Fuqiang Gu, Songtao Guo

ruiyuan.li@cqu.edu.cn

https://shardingsphere.apache.org/

This Work is Based on Apache ShardingSphere V5.0.0

ICDE 2022, Kuala Lumpur, Malaysia
Background

- Relational Databases are still the Main Forces of OLTP
  - 7/10 of the Top10 list are RDBMSs

- Key Features of RDMBSs
  - Complete ACID Transactions
  - Standard SQL Languages
  - Tests of Time

- Modern Apps’ Requirements
  - Massive Amounts of Data
    - 271.5 billion transactions of JD
  - Highly Concurrent Accesses
    - 583,000 orders/second of Tmall

<table>
<thead>
<tr>
<th>Rank</th>
<th>DBMS</th>
<th>Database Model</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Oracle</td>
<td>Relational, Multi-model</td>
<td>1254.82</td>
</tr>
<tr>
<td>2.</td>
<td>MySQL</td>
<td>Relational, Multi-model</td>
<td>1204.16</td>
</tr>
<tr>
<td>3.</td>
<td>Microsoft SQL Server</td>
<td>Relational, Multi-model</td>
<td>938.46</td>
</tr>
<tr>
<td>4.</td>
<td>PostgreSQL</td>
<td>Relational, Multi-model</td>
<td>614.46</td>
</tr>
<tr>
<td>5.</td>
<td>MongoDB</td>
<td>Document, Multi-model</td>
<td>483.38</td>
</tr>
<tr>
<td>6.</td>
<td>Redis</td>
<td>Key-value, Multi-model</td>
<td>177.61</td>
</tr>
<tr>
<td>7.</td>
<td>Elasticsearch</td>
<td>Search engine, Multi-model</td>
<td>160.83</td>
</tr>
<tr>
<td>8.</td>
<td>IBM Db2</td>
<td>Relational, Multi-model</td>
<td>160.46</td>
</tr>
<tr>
<td>9.</td>
<td>Microsoft Access</td>
<td>Relational</td>
<td>142.78</td>
</tr>
<tr>
<td>10.</td>
<td>SQLite</td>
<td>Relational</td>
<td>132.80</td>
</tr>
</tbody>
</table>

Double-Eleven Shopping Festival, 2020
Existing Solutions

- **Traditional Relational Databases**
  - Designed for Standalone Machines
  - Scalability Issue

- **NoSQL Databases**
  - Apache HBase, Cassandra
  - Lack Completed Transactions
  - Lack Completed SQL Support
  - Usability Issue

- **New Architecture Databases**
  - Spanner, TiDB, CockroachDB, Amazon Aurora
  - Need More Tests of Time
  - Require High Migration Costs
  - Maintenance Issue
Data Sharding Middleware

(a) Before Data Sharding
SELECT * FROM t_user WHERE uid=0;
SELECT * FROM t_user WHERE uid=1;
uid\%2=0

(b) After Data Sharding
SELECT * FROM t_user WHERE uid=0;  \(uid\%2=0\)
SELECT * FROM t_user WHERE uid=1;  \(uid\%2=1\)

How Does Sharing Middleware Works?
- Route the Requests to Different Databases in Different Machines
- Merge the Results from Different Databases

A Modest Solution Based on Existing Relational Databases
- Existing Applications can be Transplanted to Sharding Middlewares without Any Changes
- Important to Some Critical Scenarios, e.g., Financial Systems

Enable Developers to Use Sharded Databases like One Database!
Challenges of Data Sharding

- **Complexity Problem**
  - Many kinds of relational databases
  - Different protocols and dialects
  - Various SQL types
    - selection, aggregation, table join

- **Efficiency Problem**
  - Taking time to forward requests and merge results

- **Transaction Problem**
  - Sharding introduces distributed transactions
  - Many types of distributed transactions

- **Maintenance Problem**
  - Sharding rules are complicated
  - Manual configuration is tedious and error-prone
Our Solution: Apache ShardingSphere

- **Adaptors**
  - ShardingSphere-JDBC
  - ShardingSphere-Proxy

- **SQL Engine**
  - Parser
  - Router
  - Rewriter
  - Executor
  - Merger

- **Features**
  - Shard
  - RW-Split
  - Shadow
  - Encrypt
  - Scale

- **Governor**
  - Config Manage
  - Health Detect

- **Data Sources**
  - MySQL
  - PostgreSQL
  - SQL Server
  - Oracle
  - MariaDB
  - open-Gauss

- **Holistic**
  - 6 RDBMS + other SQL-92 DBMS
  - 3 types of distributed transactions
  - Abundant features

- **Efficient**
  - JDBC and proxy adaptors
  - Intelligent execution strategies

- **Pluggable & Extensible**
  - SPIs and design patterns
  - Free combination of features

- **User-Friendly**
  - Not aware of distributed transax
  - DistSQL + AutoTable for DBAs

Apache’s First Top-level Open-source Platform for Data Sharding!
Data Flow of Apache ShardingSphere

**Scenarios of SS-JDBC**
- Deployed with Java Applications
- Transparent to Developers

**Scenarios of SS-Proxy**
- Deployed as a Single Process
- Act as a Database
- Support Any Language
- Friendly to DBAs

**Governor**
- Deployed Independently
- Monitor Data Sources
- Maintain Configurations

ShardingSphere-JDBC is much more efficient.
### Key Concepts of Data Sharding

#### Types of Data Sharding
- Vertical Table Sharding
- Horizontal Data Source Sharding

#### Sharding Key & Sharding Algorithm
- Logical Table & Actual Table
- Data Node, e.g., DS<sub>0</sub>.t_user_h0

---

<table>
<thead>
<tr>
<th>Table Sharding</th>
<th>Data Source Sharding</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_user_v0</td>
<td>t_user_v1</td>
</tr>
<tr>
<td>t_order_v0</td>
<td>t_order_v1</td>
</tr>
<tr>
<td>t_user_h0</td>
<td>t_order_h0</td>
</tr>
<tr>
<td>t_user_h1</td>
<td>t_order_h1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vertical</th>
<th>Horizontal</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_user_v0</td>
<td>t_user_v1</td>
</tr>
<tr>
<td>t_order_v0</td>
<td>t_order_v1</td>
</tr>
<tr>
<td>t_user_h0</td>
<td>t_order_h0</td>
</tr>
<tr>
<td>t_user_h1</td>
<td>t_order_h1</td>
</tr>
</tbody>
</table>

---

### Example SQL Queries

- SELECT * FROM t_user WHERE uid=0;
- SELECT * FROM t_user WHERE uid=1;
- SELECT * FROM t_user WHERE uid%2=0;
- SELECT * FROM t_user WHERE uid%2=1;

---

- Logical Table
- Sharding Key
- Sharding Algorithm
Distributed Transactions Support

- Three Types of Distributed Transactions for Different Usage Scenarios
  - **XA Transaction**: strict ACID with performance degradation
  - **Local Transaction**: performance boost but no consistency guarantee
  - **BASE Transaction**: performance improvement with eventual consistency

- Challenges of Distributed Transactions with Data Sharding
  - Require Lots of Efforts to Use Distributed Transactions (Steep Learning Curves)

- Support Distributed Transactions in the Same Way of Standalone Transaction
DistSQL for Configuration Management

- **Existing Method for Sharding Rules Configuration**
  - Create actual tables first
  - Write configuration files manually
  - Offline before modification
  - Not friendly to developers and DBAs

- **DistSQL**
  - SQL-like statement
  - Keep online

- **AutoTable**
  - Allow users to not care about which databases store the actual tables

```sql
CREATE|ALTER SHARDING TABLE RULE t_user_h (RESOURCES(ds0, ds1), SHARDING_COLUMN=uid, TYPE(NAME=hash_mod, PROPERTIES("sharding-count"=2)));
```

ShardingSphere would create the actual tables and bind them to the logical tables **intelligently**.
Complete SQL Engine

- SQL Engine is the Kernel of Apache ShardingSphere
- All Features are Pluggable to the System
- Support Almost All SQL Statements for Data Sharding
Experiments

- **Datasets**
  - Sysbench: 20~100 millions data records
  - TPCC: 200 warehouses (120 millions)

- **Comparing Systems**
  - Typical Relational Databases
    - MySQL v5.7.26 (MS), PostgreSQL v10.17 (PG)
  - Other Sharding Middlewares
    - Vitess v12.0.0, Citus v9.0.0
  - New Architecture Databases
    - TiDB v5.2.0, CockroachDB v21.1.11 (CRDB)
  - DBaaS Databases in the Cloud
    - Aurora MySQL v2.07.2 (Aurora_{MS}), Aurora PostgreSQL v4.2 (Aurora_{PG})

- **Metrics**
  - Average Response Time (AvgT)
  - 99\textsuperscript{th}/90\textsuperscript{th} Percentile Response Time (99T/90T)

- **Experimental Settings**
  - For Distributed Experiments
    - 12 Virtual Servers in Huawei Cloud
    - CentOS 7.1 64bit, 32-vCore CPU, 64GB RAM and 1TB disk, Linux Multiqueue Networking Enabled
  - For Cloud or Standalone Experiments
    - 5 Virtual Servers in Amazon Cloud
    - Red Hat Enterprise Linux 8.3 64bit, 8-vCore CPU, 64 GB RAM and 100GB SSD
    - Each Server Runs at most One Data Source with 10 Sharded Tables
Test of Efficiency

- SS-based method always performs best in terms of TPS, 99T/90T and AvgT for both Sysbench and TPCC
- SSJ-based method is about 2~10 times faster than other systems in terms of AvgT
Experiments

- Test of Scalability
  - SSJ-based method always performs the best
  - Bigger data size, TPS decreases slightly
  - More requests, TPS first increases then be stable
  - More data servers, TPS increases linearly

Download the experimental code: [http://ss4icde.urban-computing.com](http://ss4icde.urban-computing.com)
188 Companies Claimed to be Using ShardingSphere!
Use Case: JD Baitiao

- JD Baitiao is an advanced credit payment product of JD.com, one of the biggest E-commerce companies in China.
- 6400 shards, over 100 billion records
- Values:
  - Make the system more scalable
  - Reduce development costs

MySQL Official Case: https://www.mysql.com/customers/view/?id=1461
Thank You!

Github: https://github.com/apache/shardingsphere

Homepage: https://shardingsphere.apache.org/

Top 10 Most Active Projects by Apache Foundation in 2021!

Database Plus      Connect      Enhance      Pluggable

15000+ Stars      8000+ Pull Requests
5000+ Forks       300+ Contributors
5000+ Forks