ShardingSphere

TPC-C 场景下 ShardingSphere 性能测试实战指南

Apache ShardingSphere PMC & SphereEx 基础设施研发工程师

吴伟杰

September 3, 2022

ShardingSphere

目录

Transaction Processing Performance Council

TPC®

ACTIAN · Alibaba.com · AMD · cisco · DELL Technologies · FUJITSU · Hewlett Packard Enterprise · HITACHI · HUAWEI · IBM · intel · inspur · Lenovo · Microsoft · Nettrix宁畅 · NUTANIX · NVIDIA · ORACLE · Red Hat · TTA · vmware

Home | About the TPC | Benchmarks/Results | Downloads | TPCTC | Contact | Miscellaneous | Search | Member Login

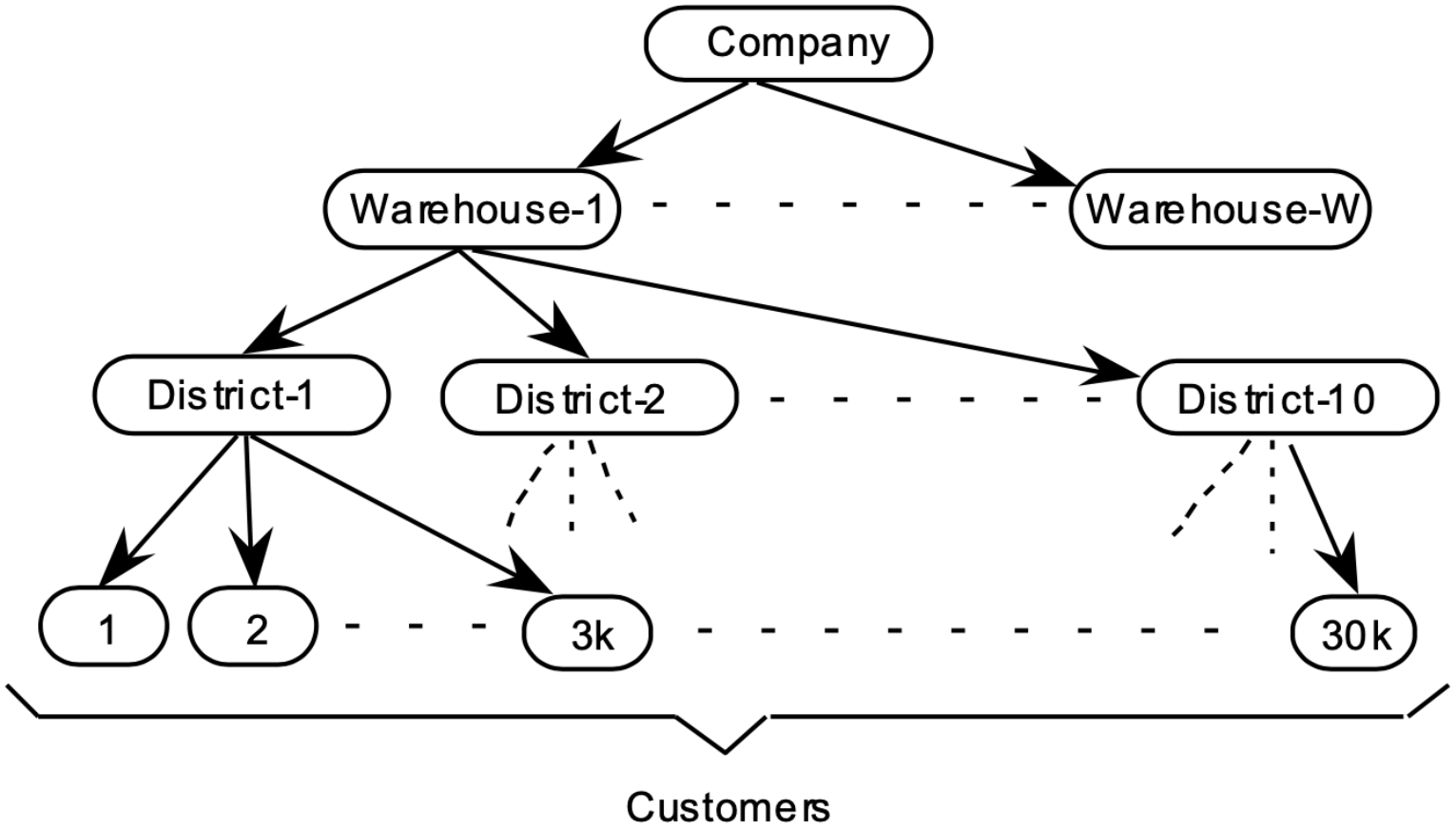
TPC Benchmarks Overview

Year	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022
TPC-A																																			
TPC-App																																			
TPC-B																																			
TPC-C																							10	5	4	2	1			1	3	4	3	2	
TPC-D																																			
TPC-DI																																			
TPC-DS																															1	5	4	4	
TPC-E																							14	11	7	4	5	4	4	2	2	3	1	3	
TPC-H																							10	20	5	9	16	5	10	5	4	9	4	32	
TPC-R																																			
TPC-VMS																																			
TPC-W																																			
TPC Express Benchmark Standards																																			
TPCx-AI																																			
TPCx-BB																													4	3	2	6	2	4	
TPCx-HCI																																		1	
TPCx-HS																												15	8	1	6	14	4	10	
TPCx-IoT																														1	1	5	2	12	
TPCx-V																															1	2		2	
Common Specifications																																			
Pricing																																			
Energy																																			
* ... active benchmark			... obsolete benchmark												Benchmarks published since 2010 as of 12/31/2021								34	36	16	15	22	24	26	13	20	48	20	70	0

TPC-C 与 TPC-E

	TPC-C	TPC-E
业务背景	批发商订单	证券交易所
表数量	9	33
总列数	92	188
事务种类	5	12
读写事务	3	4
只读事务	2	6
定时/条件事务	0	2
硬件成本	与规模、测试程序相关	
测试程序成本	相对较低	相对较高
调优成本	相对较低	相对较高
仿真程度	相对较低	相对较高

TPC-C 业务背景



一家批发商公司

公司建立了 W 个仓库

每个仓库覆盖 10 个区域

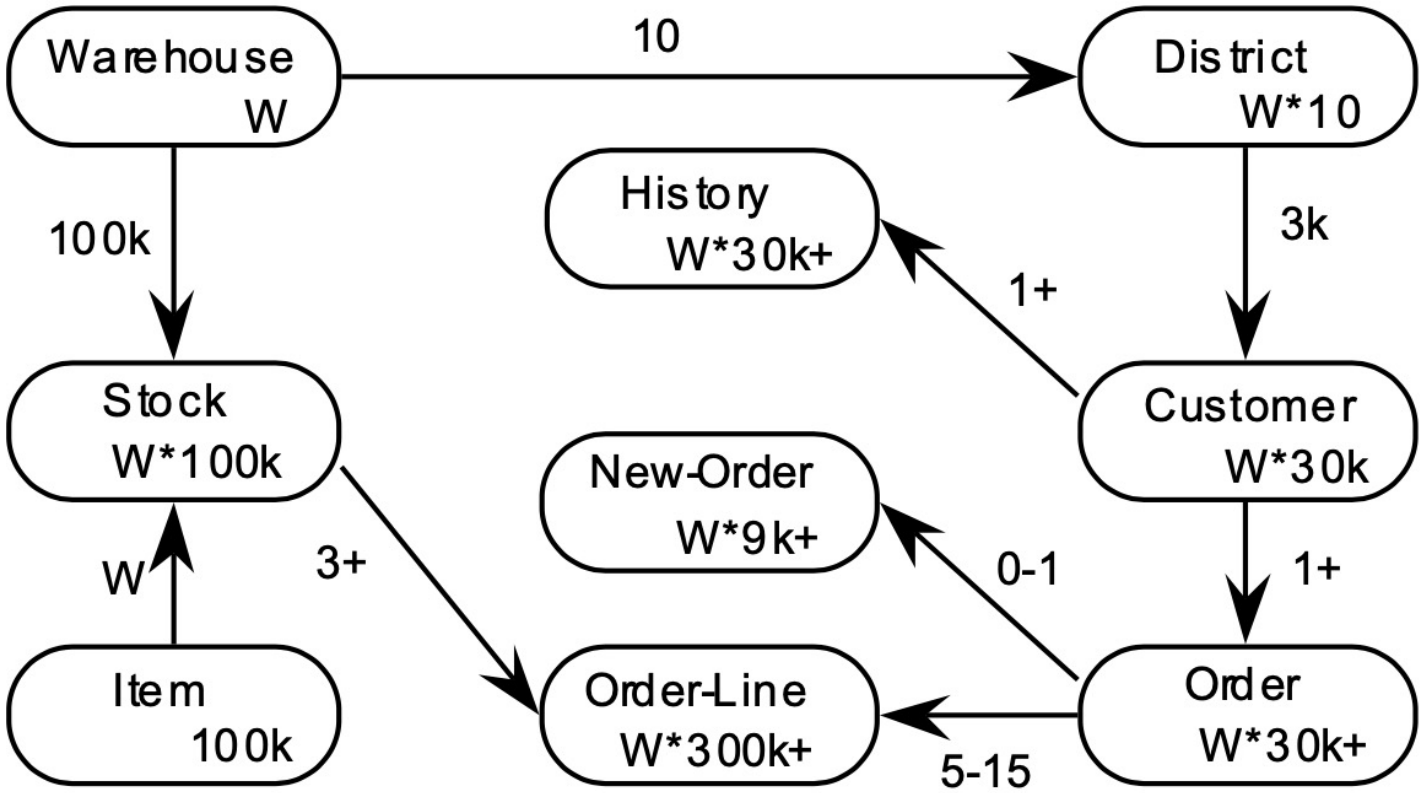
每个区域服务 3000 位顾客

TPC-C 事务种类与指标

	事务类型	负载	执行比例
New Order	读写事务	中	45%
Payment	读写事务	低	43%
Order Status	只读事务	中	4%
Delivery	读写事务	批处理	4%
Stock Level	只读事务	高	4%

tpmC
每分钟 New Order 事务数

TPC-C 数据模型



不同 W 下的初始数据量：

W = 1
初始数据量约 100 MB
Order-Line 初始数据 300,000 行

W = 1000
初始数据量约 100 GB
Order-Line 初始数据 300,000,000 行

W = 8000
初始数据量约 800 GB
Order-Line 初始数据 2,400,000,000 行

测试工具 BenchmarkSQL 5.0

- ✓ 开源
- ✓ 基于 JDBC
- ✓ 基本实现了 TPC-C 压力模拟逻辑
- ✓ 支持采集远程机器资源使用率
- ✓ 支持可视化报告生成

Result Summary

Note that the "simple" driver is not a true TPC-C implementation. This driver only measures the database response time, not the response time of a Sy

Transaction Type	Latency		Count	Percent	Rollback	Errors	Skipped Deliveries
	90th %	Maximum					
NEW_ORDER	0.024s	0.586s	136084	44.778%	0.990%	0	N/A
PAYMENT	0.009s	0.622s	131420	43.243%	N/A	0	N/A
ORDER_STATUS	0.013s	1.730s	12208	4.017%	N/A	0	N/A
STOCK_LEVEL	0.005s	0.437s	12125	3.990%	N/A	0	N/A
DELIVERY	0.000s	0.009s	12070	3.972%	N/A	0	N/A
DELIVERY_BG	0.076s	0.568s	12070	N/A	N/A	0	0

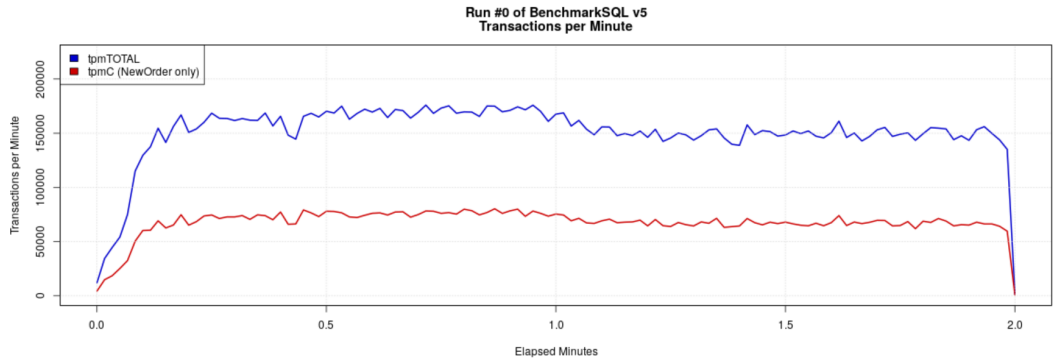
Overall tpmC: 68042.00
Overall tpmTotal: 151953.50

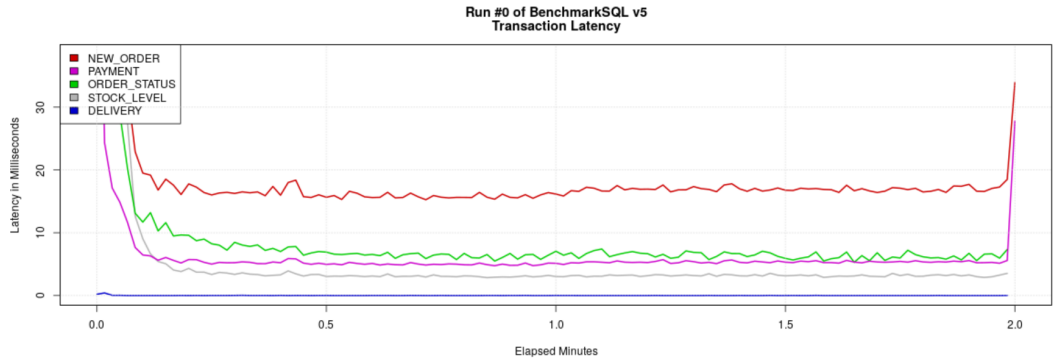
The TPC-C specification has an theoretical maximum of 12.86 NEW_ORDER transactions per minute per warehouse. In reality this value cannot be r
including the database.

The above tpmC of 68042.00 is 16534.312% of that theoretical maximum for a database with 32 warehouses.

Transactions per Minute and Transaction Latency

tpmC is the number of NEW_ORDER Transactions, that where processed per minute. tpmTOTAL is the number of Transactions processed per minute





BenchmarkSQL 零代码改造使用 ShardingSphere-JDBC

```
db=postgres
driver=org.postgresql.Driver
conn=jdbc:postgresql://127.0.0.1:5432/bmsql
```

只须调整数据库驱动、JDBC URL

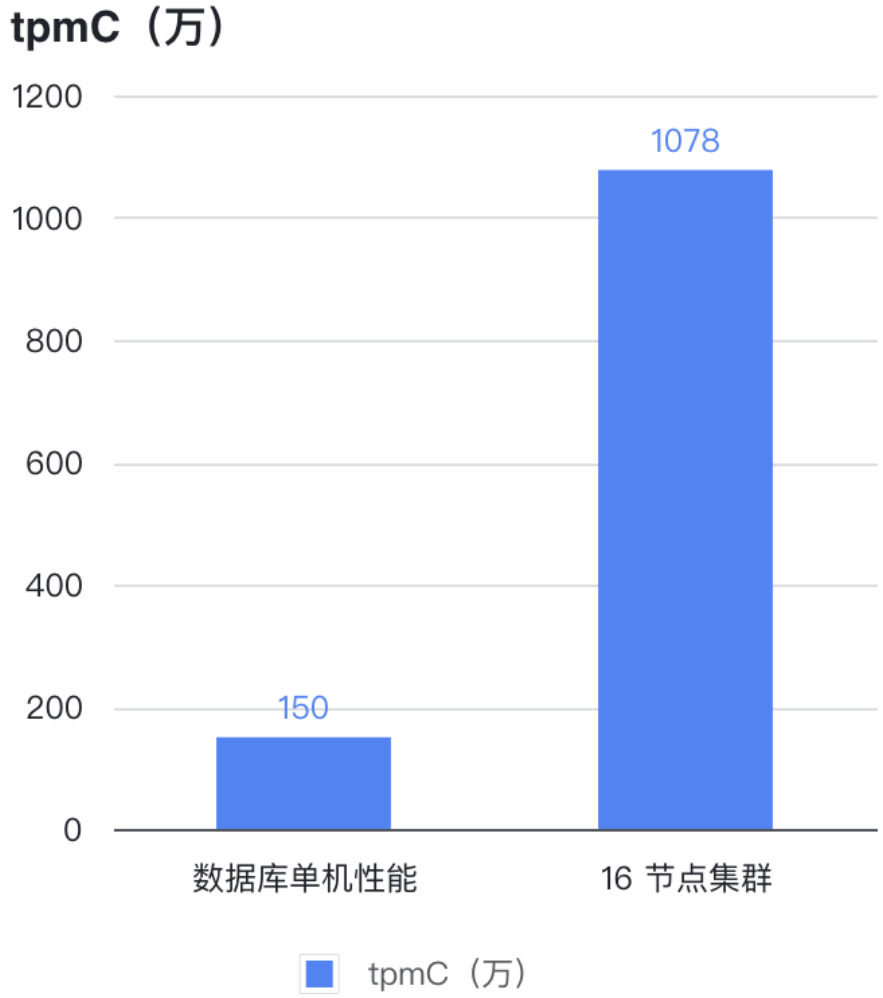
```
db=postgres
driver=org.apache.shardingsphere.driver.ShardingSphereDriver
conn=jdbc:shardingsphere:/home/wuweijie/benchmarksql/run/config-sharding.yaml
```

* ShardingSphere-JDBC Driver 自 ShardingSphere 5.1.2 提供

ShardingSphere

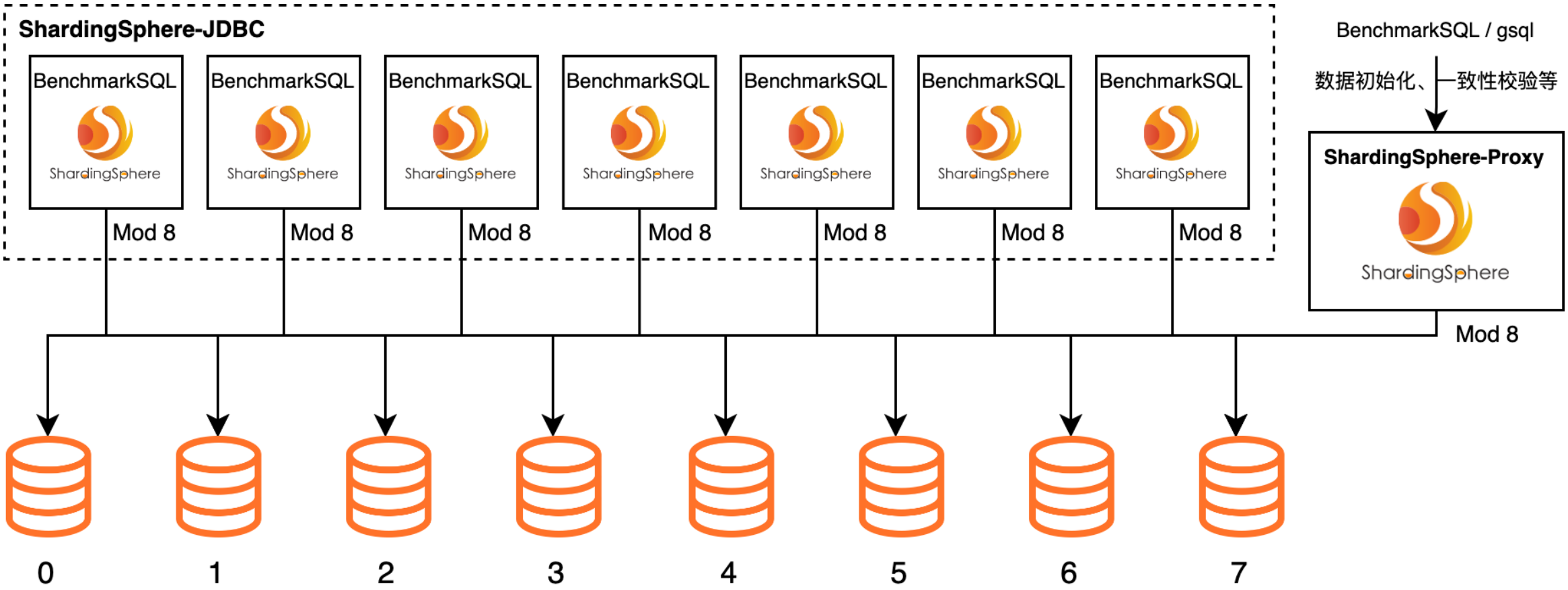
目录

ShardingSphere-JDBC 数据分片 TPC-C 测试案例



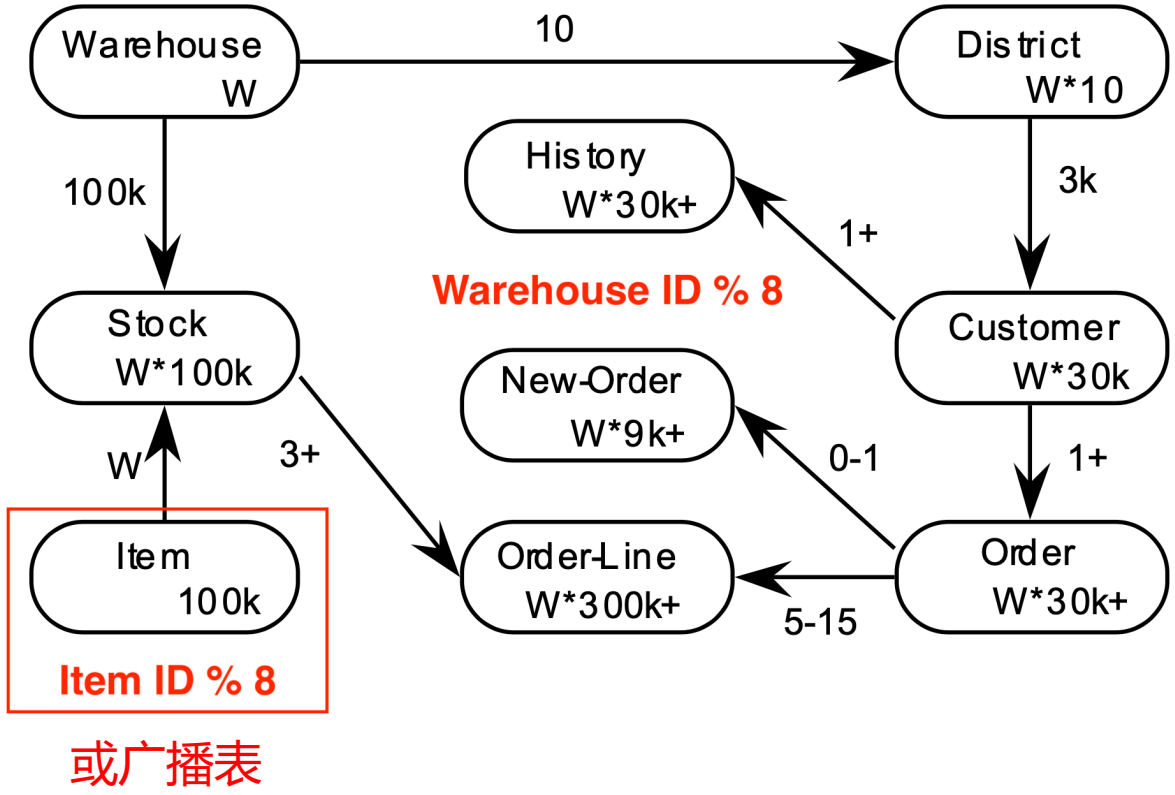
配置	8 x openGauss	7 x ShardingSphere-JDBC	1 x ShardingSphere-Proxy
CPU	128 核 2 路鲲鹏 920	128 核 2 路鲲鹏 920	128 核 2 路鲲鹏 920
内存	768GB	768GB	768GB
系统盘	1TB	1TB	1TB
数据盘	3 x 4TB NVMe SSD	无	无

测试组网


ShardingSphere-JDBC
BenchmarkSQL
ShardingSphere
Mod 8
BenchmarkSQL / gsql
数据初始化、一致性校验等
ShardingSphere-Proxy
0
1
2
3
4
5
6
7

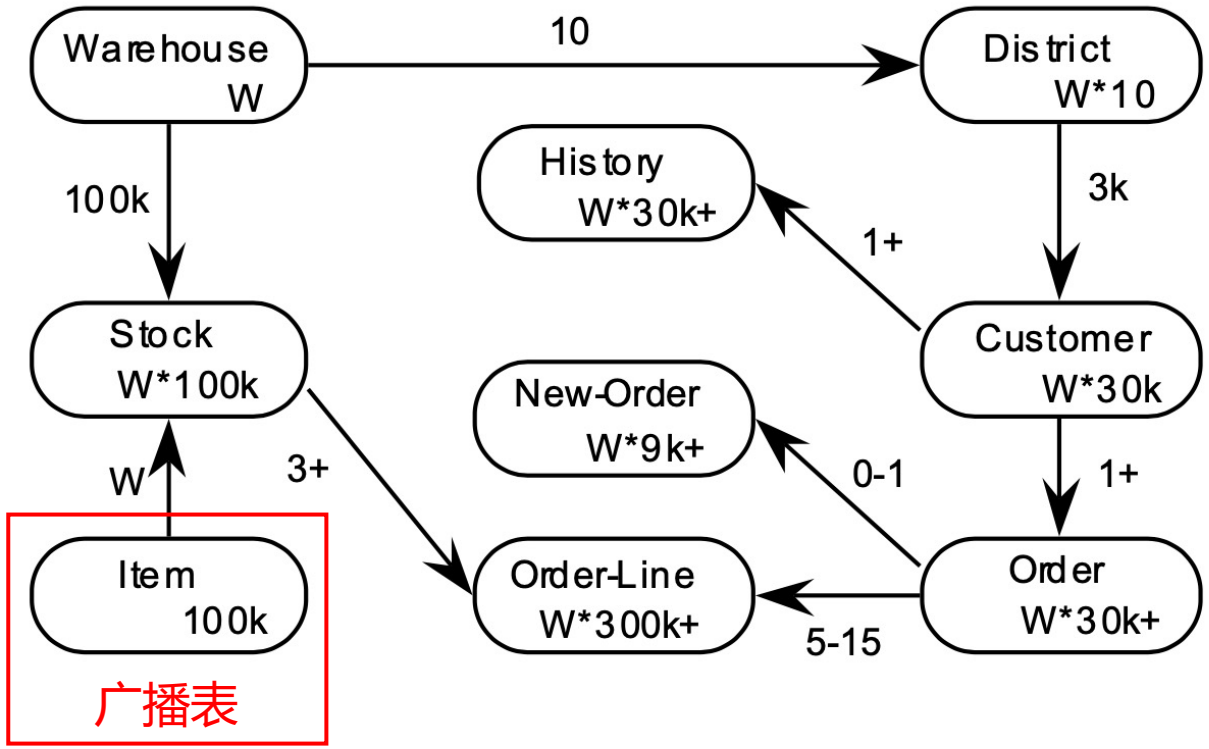
目录

TPC-C 场景下 ShardingSphere 数据分片方案设计



基于 Mod 算法的数据分片方案设计

TPC-C 场景下 ShardingSphere 数据分片方案设计



Warehouse ID

1~1000 -> 数据源 0
1001~2000 -> 数据源 1
...
6001~7000 -> 数据源 6
7001~8000 -> 数据源 7

基于 Range 算法的数据分片方案设计

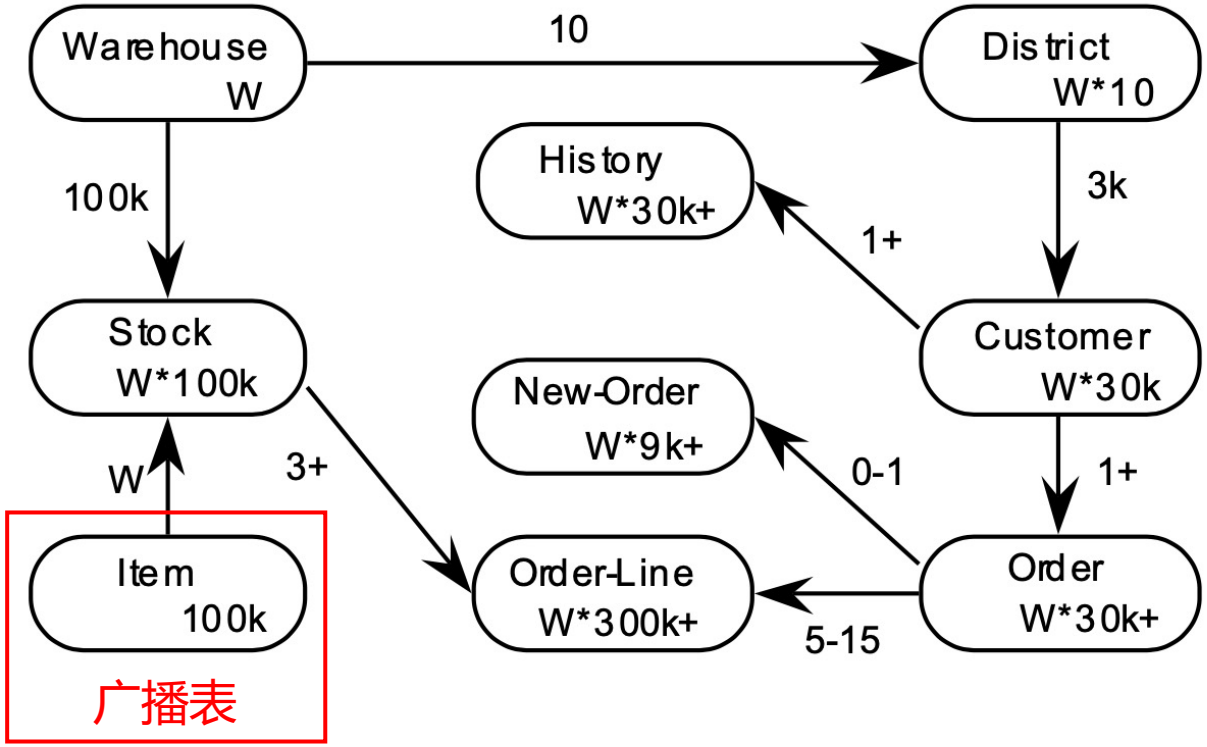
数据分片使用技巧：合理配置绑定表

bindingTables: ['bmsql_stock, bmsql_district, bmsql_order_line']

```
bmsql_sharding=> preview SELECT count(*) AS low_stock FROM (
bmsql_sharding(>     SELECT s_w_id, s_i_id, s_quantity
bmsql_sharding(>         FROM bmsql_stock
bmsql_sharding(>         WHERE s_w_id = 1 AND s_quantity < 100 AND s_i_id IN (
bmsql_sharding(>             SELECT ol_i_id
bmsql_sharding(>                 FROM bmsql_district
bmsql_sharding(>                 JOIN bmsql_order_line ON ol_w_id = d_w_id
bmsql_sharding(>                  AND ol_d_id = d_id
bmsql_sharding(>                  AND ol_o_id >= d_next_o_id - 20
bmsql_sharding(>                  AND ol_o_id < d_next_o_id
bmsql_sharding(>                 WHERE d_w_id = 1 AND d_id = 1
bmsql_sharding(>         )
bmsql_sharding(>     ) AS L;
 data_source_name |                                actual_sql
------------------+-------------------------------------------------------------------
 ds_0             | SELECT count(*) AS low_stock FROM (                               +
                  |     SELECT s_w_id, s_i_id, s_quantity                             +
                  |         FROM bmsql_stock                                          +
                  |         WHERE s_w_id = 1 AND s_quantity < 100 AND s_i_id IN (     +
                  |             SELECT ol_i_id                                        +
                  |                 FROM bmsql_district                               +
                  |                 JOIN bmsql_order_line ON ol_w_id = d_w_id         +
                  |                  AND ol_d_id = d_id                               +
                  |                  AND ol_o_id >= d_next_o_id - 20                  +
                  |                  AND ol_o_id < d_next_o_id                        +
                  |                 WHERE d_w_id = 1 AND d_id = 1                     +
                  |         )                                                         +
                  |     ) AS L
(1 row)
```

```
bmsql_sharding=> preview SELECT count(*) AS low_stock FROM (
bmsql_sharding(>     SELECT s_w_id, s_i_id, s_quantity
bmsql_sharding(>         FROM bmsql_stock
bmsql_sharding(>         WHERE s_w_id = 1 AND s_quantity < 100 AND s_i_id IN (
bmsql_sharding(>             SELECT ol_i_id
bmsql_sharding(>                 FROM bmsql_district
bmsql_sharding(>                 JOIN bmsql_order_line ON ol_w_id = d_w_id
bmsql_sharding(>                  AND ol_d_id = d_id
bmsql_sharding(>                  AND ol_o_id >= d_next_o_id - 20
bmsql_sharding(>                  AND ol_o_id < d_next_o_id
bmsql_sharding(>                 WHERE d_w_id = 1 AND d_id = 1
bmsql_sharding(>         )
bmsql_sharding(>     ) AS L;
 data_source_name |                                actual_sql
------------------+-------------------------------------------------------------------
 ds_3             | SELECT count(*) AS low_stock FROM (                               +
                  |     SELECT s_w_id, s_i_id, s_quantity                             +
                  |         FROM bmsql_stock                                          +
                  |         WHERE s_w_id = 1 AND s_quantity < 100 AND s_i_id IN (     +
                  |             SELECT ol_i_id                                        +
                  |                 FROM bmsql_district                               +
                  |                 JOIN bmsql_order_line ON ol_w_id = d_w_id         +
                  |                  AND ol_d_id = d_id                               +
                  |                  AND ol_o_id >= d_next_o_id - 20                  +
                  |                  AND ol_o_id < d_next_o_id                        +
                  |                 WHERE d_w_id = 1 AND d_id = 1                     +
                  |         )                                                         +
                  |     ) AS L
 ds_2             | SELECT count(*) AS low_stock FROM (                               +
                  |     SELECT s_w_id, s_i_id, s_quantity                             +
                  |         FROM bmsql_stock                                          +
                  |         WHERE s_w_id = 1 AND s_quantity < 100 AND s_i_id IN (     +
                  |             SELECT ol_i_id                                        +
                  |                 FROM bmsql_district                               +
                  |                 JOIN bmsql_order_line ON ol_w_id = d_w_id         +
                  |                  AND ol_d_id = d_id                               +
                  |                  AND ol_o_id >= d_next_o_id - 20                  +
                  |                  AND ol_o_id < d_next_o_id                        +
                  |                 WHERE d_w_id = 1 AND d_id = 1                     +
                  |         )                                                         +
                  |     ) AS L
 ds_1             | SELECT count(*) AS low_stock FROM (                               +
                  |     SELECT s_w_id, s_i_id, s_quantity                             +
                  |         FROM bmsql_stock                                          +
                  |         WHERE s_w_id = 1 AND s_quantity < 100 AND s_i_id IN (     +
                  |             SELECT ol_i_id                                        +
                  |                 FROM bmsql_district                               +
                  |                 JOIN bmsql_order_line ON ol_w_id = d_w_id         +
                  |                  AND ol_d_id = d_id                               +
                  |                  AND ol_o_id >= d_next_o_id - 20                  +
                  |                  AND ol_o_id < d_next_o_id                        +
                  |                 WHERE d_w_id = 1 AND d_id = 1                     +
                  |         )                                                         +
                  |     ) AS L
 ds_0             | SELECT count(*) AS low_stock FROM (                               +
                  |     SELECT s_w_id, s_i_id, s_quantity                             +
                  |         FROM bmsql_stock                                          +
                  |         WHERE s_w_id = 1 AND s_quantity < 100 AND s_i_id IN (     +
                  |             SELECT ol_i_id                                        +
                  |                 FROM bmsql_district                               +
                  |                 JOIN bmsql_order_line ON ol_w_id = d_w_id         +
                  |                  AND ol_d_id = d_id                               +
                  |                  AND ol_o_id >= d_next_o_id - 20                  +
                  |                  AND ol_o_id < d_next_o_id                        +
                  |                 WHERE d_w_id = 1 AND d_id = 1                     +
                  |         )                                                         +
                  |     ) AS L
(4 rows)
```

数据分片使用技巧：合理配置广播表

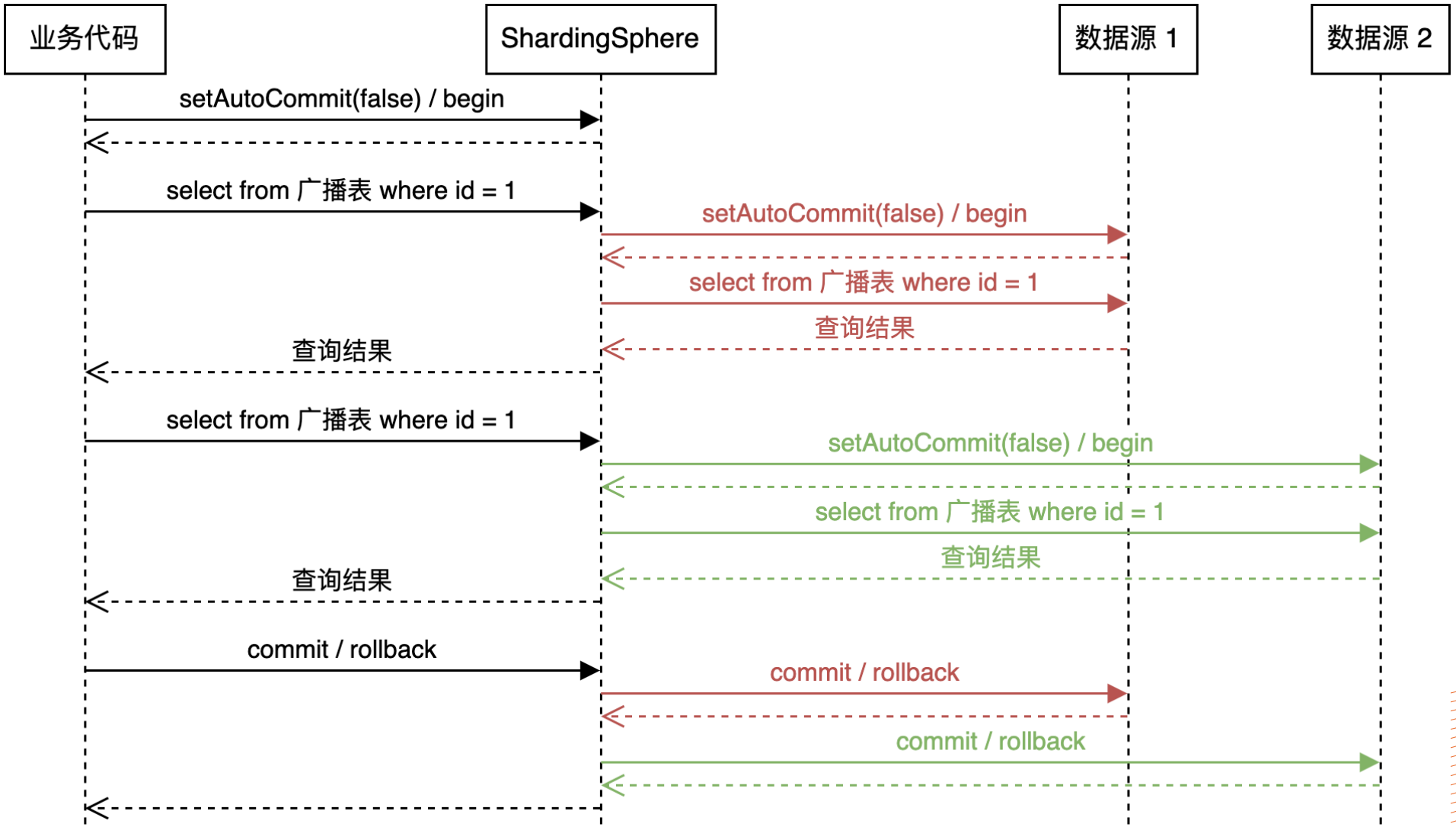


- 数据量可评估
- 读多写少
- 需要与分片表关联查询
- 或事务中需要避免访问多数据源

例如：字典表、商品类目表等

数据分片使用技巧：借助分片表控制广播表路由

单独查询广播表可能存在的问题：在一个事务中访问了多个数据源，增加了网络开销



数据分片使用技巧：借助分片表控制广播表路由

单独查询广播表可能存在的问题：在一个事务中访问了多个数据源，增加了网络开销

```
select i_price, i_name, i_data from item where i_id = 1
```

通过分片表控制广播表路由

```
select i_price, i_name, i_data from item, warehouse where i_id = 1 and w_id = 1
```

```
select i_price, i_name, i_data from item where i_id = 1 and (select true from warehouse where w_id = 1)
```

数据分片使用技巧：必要时实现自定义分片算法

示例：基于同一列分库分表

```yaml
rules:
  - !SHARDING
    tables:
      bmsql_order_line:
        actualDataNodes: ds_${0..3}.bmsql_order_line_${0..9}
        databaseStrategy:
          standard:
            shardingColumn: ol_w_id
            shardingAlgorithmName: database_inline
        tableStrategy:
          standard:
            shardingColumn: ol_w_id
            shardingAlgorithmName: table_inline

    shardingAlgorithms:
      database_inline:
        type: INLINE
        props:
          algorithm-expression: ds_${ol_w_id % 4}
      table_inline:
        type: INLINE
        props:
          algorithm-expression: bmsql_order_line_${((ol_w_id / 4) as int) % 10}
```

以下情况可以考虑实现自定义算法：

- 内置算法无法满足需求
- 性能要求苛刻

```java
@Override
public String doSharding(Collection<String> availableTargetNames,
                         PreciseShardingValue<Comparable<?>> shardingValue) {
    // ...
    long value = getLongValue(shardingValue.getValue()) / divideBeforeMod % shardingCount;
    // ...
}
```

8 核 16 线程 x86 环境， 运行 16 线程 JMH 测试

```
Benchmark                                                    Mode  Cnt          Score         Error  Units
DivideBeforeModeShardingAlgorithmBenchmark.benchDoSharding  thrpt    3  110449240.627 ± 3352731.552  ops/s
InlineShardingAlgorithmBenchmark.benchDoSharding            thrpt    3   17772289.567 ±  223358.561  ops/s
```

ShardingSphere

目录

对 BenchmarkSQL 中的 SQL 进行优化

2.7.4 Transaction Profile

2.7.4.1 The deferred execution of the Delivery transaction delivers one o order = 10) for each one of the 10 districts of the selected warehouse usin transactions. Delivering each order is done in the following steps:

1. Process the order, comprised of:

   1 row selection with data retrieval,

   (1 + items-per-order) row selections with data retrieval and update.

2. Update the customer's balance, comprised of:

   1 row selections with data update.

3. Remove the order from the new-order list, comprised of:

   1 row deletion.

https://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-c_v5.11.0.pdf

查询结果 > 1000 行

```
SELECT no_o_id
    FROM bmsql_new_order
    WHERE no_w_id = 1 AND no_d_id = 1
    ORDER BY no_o_id ASC
```

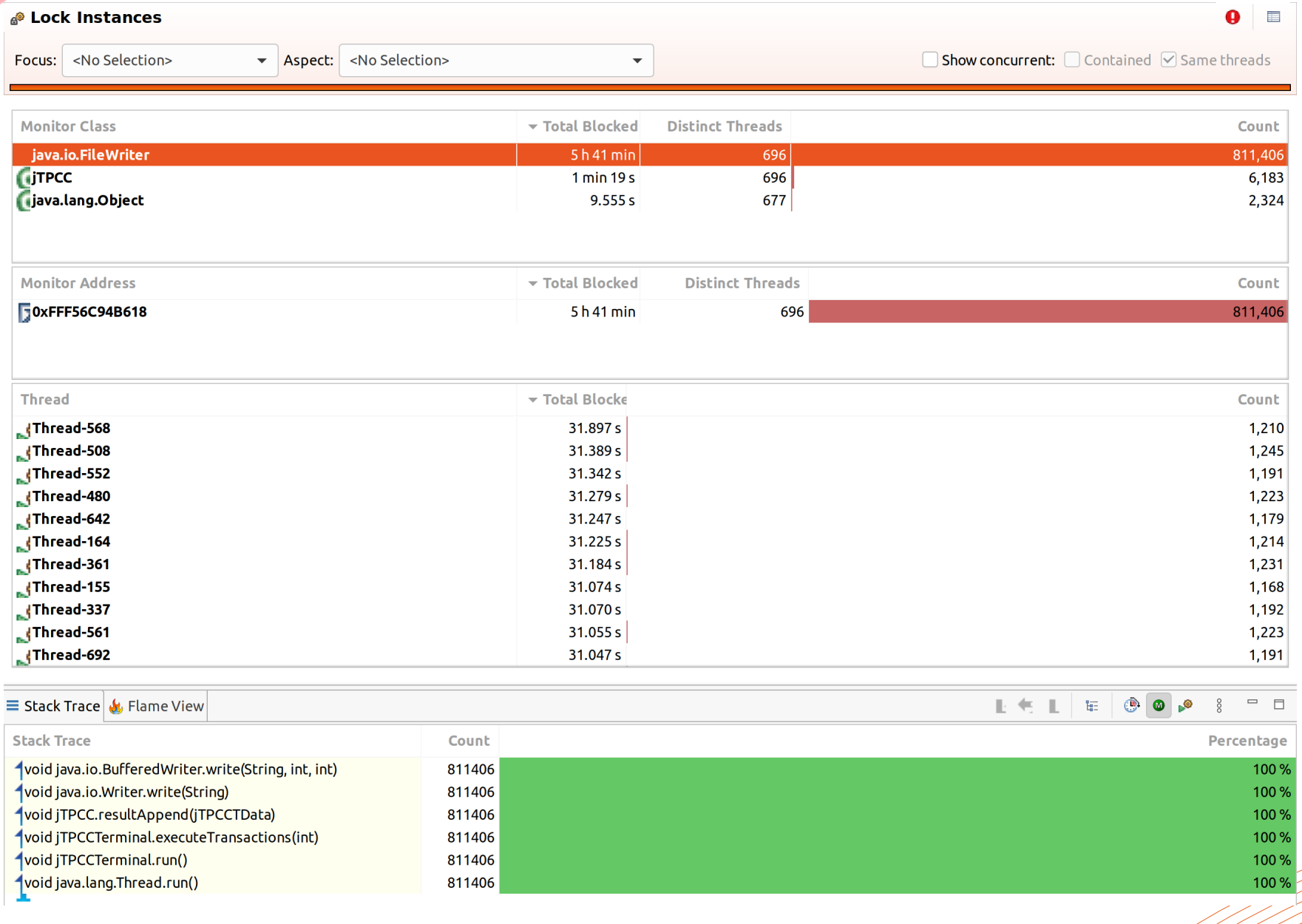
增加 LIMIT 1 或使用 Cursor

查询结果 1 行

```
SELECT no_o_id
    FROM bmsql_new_order
    WHERE no_w_id = 1 AND no_d_id = 1
    ORDER BY no_o_id ASC LIMIT 1
```

tpmC 提升 7%

优化 BenchmarkSQL 多线程文件 I/O 逻辑



相似问题：

* 大量日志输出到 stdout

ShardingSphere

ShardingSphere 5.2.0 新特性预览

数据分片支持 SQL 审计

```
bmsql_sharding=> delete from bmsql_warehouse;
ERROR:  SQL check failed, error message: Not allow DML operation without sharding conditions
bmsql_sharding=>
bmsql_sharding=>
bmsql_sharding=> delete from bmsql_warehouse where w_id in (10, 20);
DELETE 2
```

```java
public interface ShardingAuditAlgorithm extends ShardingSphereAlgorithm {

    /**
     * Sharding audit algorithm SQL check.
     *
     * @param sqlStatementContext SQL statement context
     * @param parameters SQL parameters
     * @param grantee grantee
     * @param database database
     * @return SQL check result
     */
    SQLCheckResult check(SQLStatementContext<?> sqlStatementContext, List<Object> parameters, Grantee grantee, ShardingSphereDatabase database);
}
```

增强 ShardingSphere-Proxy MySQL show processlist 与 kill

同时支持单机模式、集群模式

```
mysql> select sleep(60), w_id from bmsql_warehouse;
```

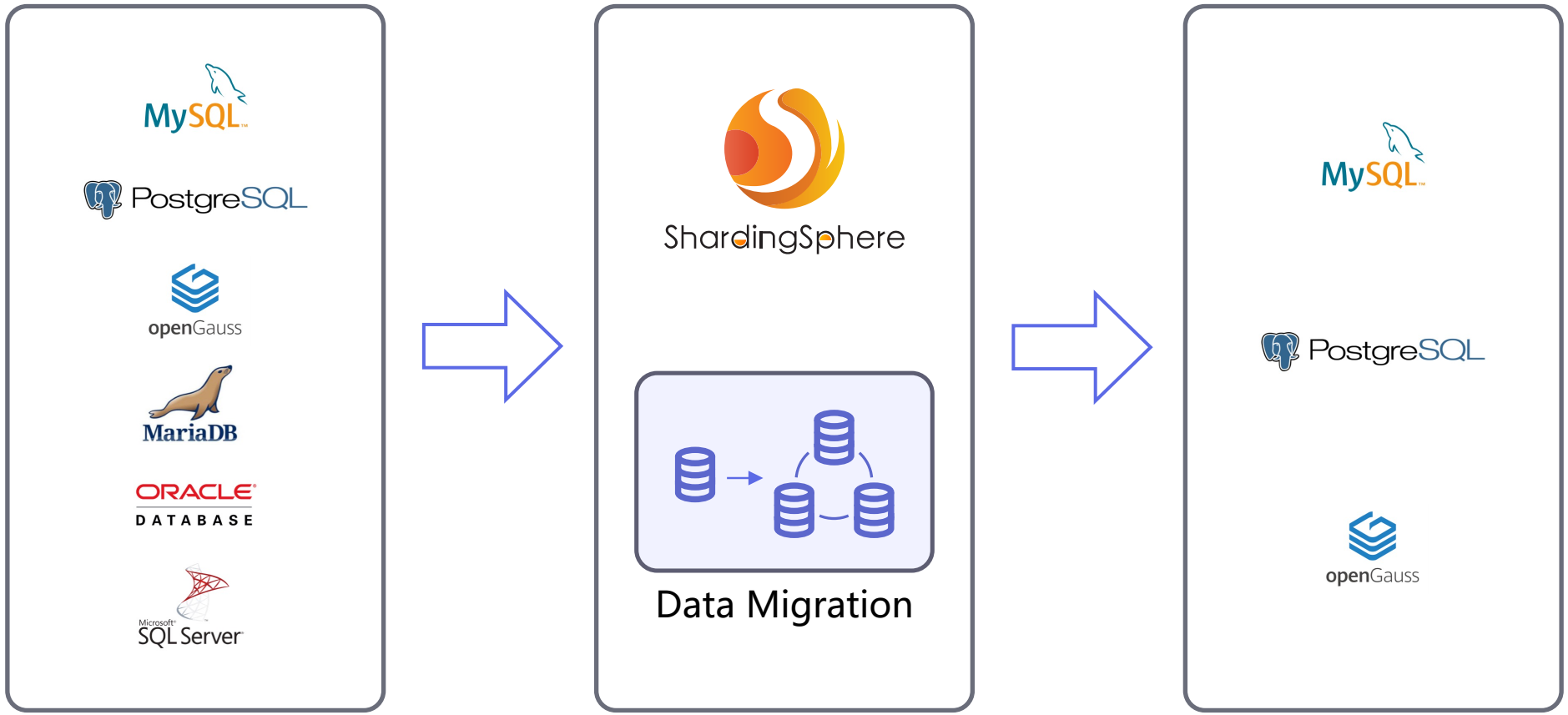
```
mysql> show processlist;
+----------------------------------+------+-----------+----------------+---------+------+---------------+-----------------------------------------+
| Id                               | User | Host      | db             | Command | Time | State         | Info                                    |
+----------------------------------+------+-----------+----------------+---------+------+---------------+-----------------------------------------+
| e5b9054432e5a7cb3d007744b0952a2f | root | 127.0.0.1 | bmsql_sharding | Execute | 16   | Executing 0/4 | select sleep(60), w_id from bmsql_warehouse |
+----------------------------------+------+-----------+----------------+---------+------+---------------+-----------------------------------------+
1 row in set (0.01 sec)

mysql> kill e5b9054432e5a7cb3d007744b0952a2f;
Query OK, 0 rows affected (0.01 sec)

mysql> show processlist;
Empty set (0.00 sec)
```

```
mysql> select sleep(60), w_id from bmsql_warehouse;
ERROR 1815 (HY000): Internal error: Statement cancelled due to client request
```

ShardingSphere-Scaling 初步支持异构数据源迁移



欢迎关注 ShardingSphere

技术干货

加入交流群

Apache ShardingSphere Website：https://shardingsphere.apache.org

Apache ShardingSphere GitHub： https://github.com/apache/shardingsphere

Apache ShardingSphere Slack Channel：https://apacheshardingsphere.slack.com