重庆大学 CHONGQING UNIVERSITY

ShardingSphere

Apache ShardingSphere: A Holistic and Pluggable Platform for Data Sharding

李瑞远, 张亮, 潘娟, 刘钧文, 王棚, 孙念君, 王善民, 陈超, 古富强, 郭松涛

ruiyuan.li@cqu.edu.cn

本工作基于Apache ShardingSphere V5.0.0

Ruiyuan Li, Liang Zhang, Juan Pan, Junwen Liu, Peng Wang, Nianjun Sun, Shanmin Wang, Chao Chen, Fuqiang Gu, Songtao Guo. Apache ShardingSphere: A Holistic and Pluggable Platform for Data Sharding[C]//2022 IEEE 38th International Conference on Data Engineering (ICDE). IEEE, 2022. (**ICDE 2022**, **CCF A**类论文)

自我介绍

基本信息

- 李瑞远，男，重庆大学副教授
- 研究方向：时空数据管理与挖掘，分布式计算，城市计算

工作经历：产研结合

- 2014.07~2018.01　微软亚洲研究院　实习研究员
- 2018.01~2020.08　京东智能城市研究院　实习研究员
- 2020.08~2021.10　京东智能城市研究院　研究员/机构负责人　团队人数：约30人
- 2021.10~至今　重庆大学计算机学院　副教授、硕导

个人标签

- 京东城市时空数据引擎JUST
- 30+篇论文，30+项专利
- 陕西省计算机学会优博
- 2021年度中国优秀专利奖

个人主页
或者搜索
“李瑞远”

微信公众号
（发布论文、
PPT和视频）

背景

❑ 关系型数据库仍是OLTP的主力军

- ❑ 在Top10的榜单中，7个是RDBMS
- ❑ 前4名都是RDBMS

https://db-engines.com/en/ranking, 2022.04

Rank			DBMS	Database Model	Score		
Apr 2022	Mar 2022	Apr 2021			Apr 2022	Mar 2022	Apr 2021
1.	1.	1.	Oracle	Relational, Multi-model	1254.82	+3.50	-20.10
2.	2.	2.	MySQL	Relational, Multi-model	1204.16	+5.93	-16.53
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	938.46	+4.67	-69.51
4.	4.	4.	PostgreSQL	Relational, Multi-model	614.46	-2.47	+60.94
5.	5.	5.	MongoDB	Document, Multi-model	483.38	-2.28	+13.41
6.	6.	↑ 7.	Redis	Key-value, Multi-model	177.61	+0.85	+21.72
7.	↑ 8.	↑ 8.	Elasticsearch	Search engine, Multi-model	160.83	+0.89	+8.66
8.	↓ 7.	↓ 6.	IBM Db2	Relational, Multi-model	160.46	-1.69	+2.68
9.	9.	↑ 10.	Microsoft Access	Relational	142.78	+7.36	+26.06
10.	10.	↓ 9.	SQLite	Relational	132.80	+0.62	+7.74

❑ 关系型数据库的特点

- ❑ 完整ACID事务支持
- ❑ 标准的SQL语言支持
- ❑ 历经时间的考验

❑ 现代网页应用需求

- ❑ 海量数据存储
  - ❑ 京东2715 亿的交易量
- ❑ 高并发量访问
  - ❑ 天猫每秒58.3万的订单峰值

2020双十一购物节

现有解决方案

❑ 传统关系型数据库

面向单机设计，资源有限 → 可扩展问题

❑ NoSQL数据库

- 缺乏完整的事务支持
- 缺乏完善的SQL支持

→ 便捷性问题

❑ 新架构数据库

- 需要更多时间验证
- 需要较高迁移成本

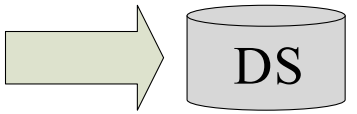
→ 运维问题

数据分片中间件

数据分片中间件的工作原理

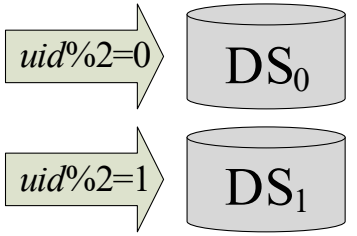
- 将请求路由到不同机器的不同数据库中
- 将不同数据库的结果合并后返回调用端

```
SELECT * FROM t_user WHERE uid=0;
SELECT * FROM t_user WHERE uid=1;
```



(a) Before Data Sharding

```
SELECT * FROM t_uesr WHERE uid=0;
SELECT * FROM t_user WHERE uid=1;
```



(b) After Data Sharding

基于现有关系型数据库，一个更为稳妥的方案

- 现有基于关系型数据库的应用可无缝迁移到分片中间件中
- 对某些关键应用非常重要，例如：金融系统

帮助开发人员以单机版数据库的方式使用分片后的数据库!

数据分片的难点

复杂度问题

- 多种多样的关系型数据库
- 不同的协议和SQL方言
- 不同的SQL语句类型
  - selection, aggregation, table join

事务问题

- 数据分片将引入分布式事务
- 有多种不同的分布式事务
- 不同分布式事务使用方法不同

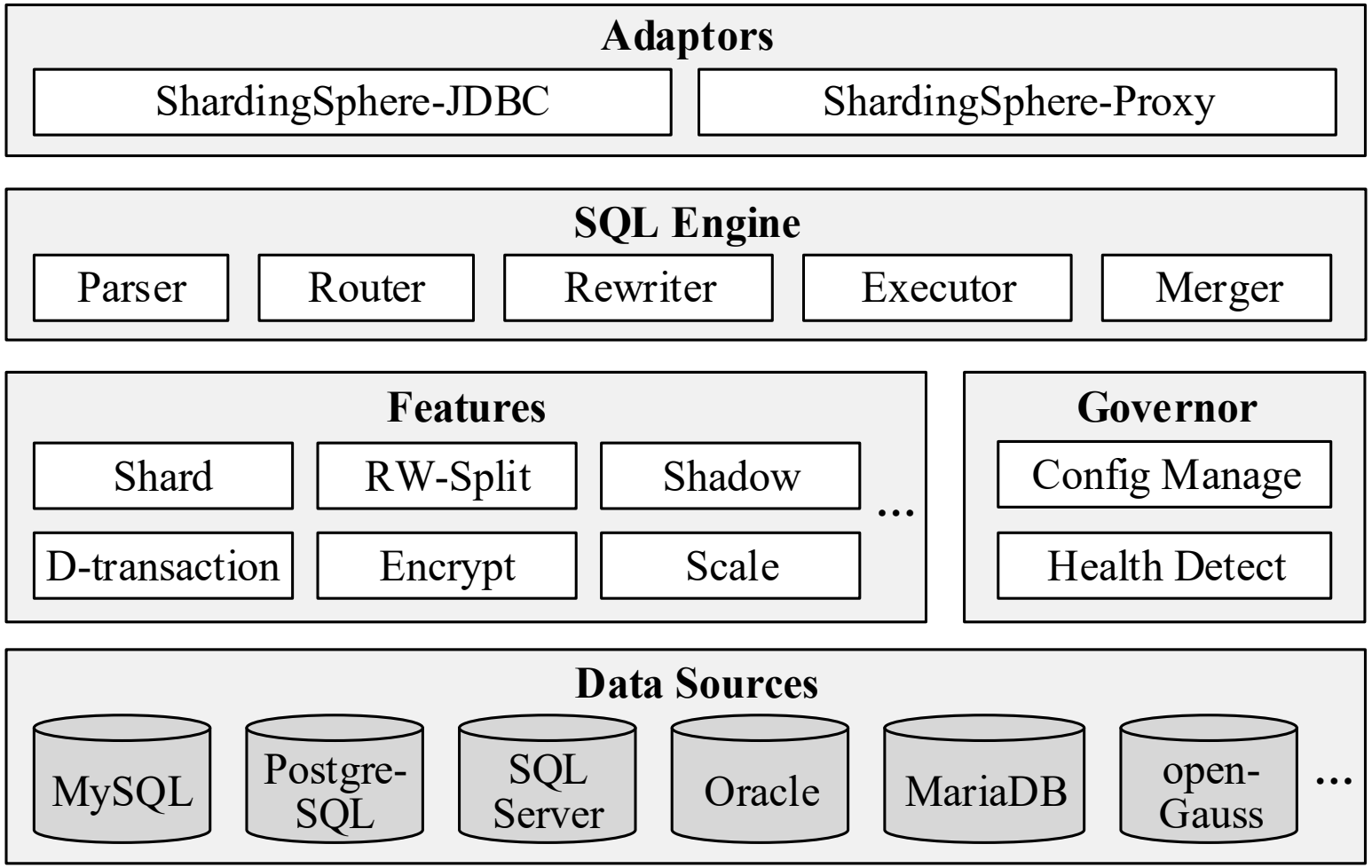
效率问题

- 转发请求、合并结果带来额外开销

运维问题

- 分片规则较为复杂
- 手动配置麻烦易出错

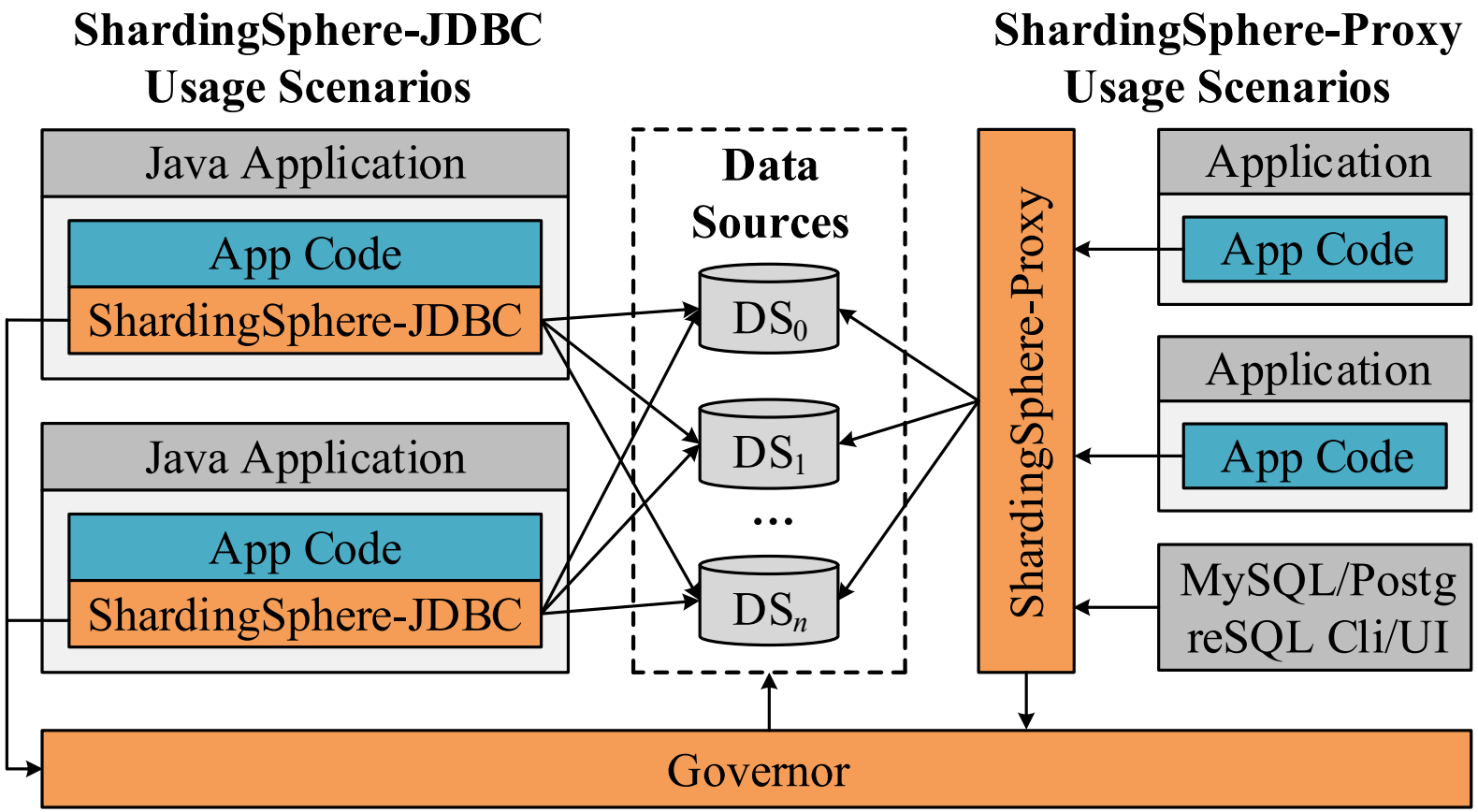
我们的方案: Apache ShardingSphere



Apache基金会首个面向数据分片的顶级开源项目

- 功能完备
  - 6种RDBMS，符合SQL-92标准
  - 3种不同的分布式事务
  - 预置了丰富的功能
- 效率高
  - 两种适配器：JDBC和Proxy
  - 智能执行策略
- 可拔插&可扩展
  - 基于SPIs和设计模式
  - 增加、删除、自由组合功能
- 用户友好
  - 屏蔽分布式事务和分片细节
  - DistSQL + AutoTable自动分片

Apache ShardingSphere的数据流

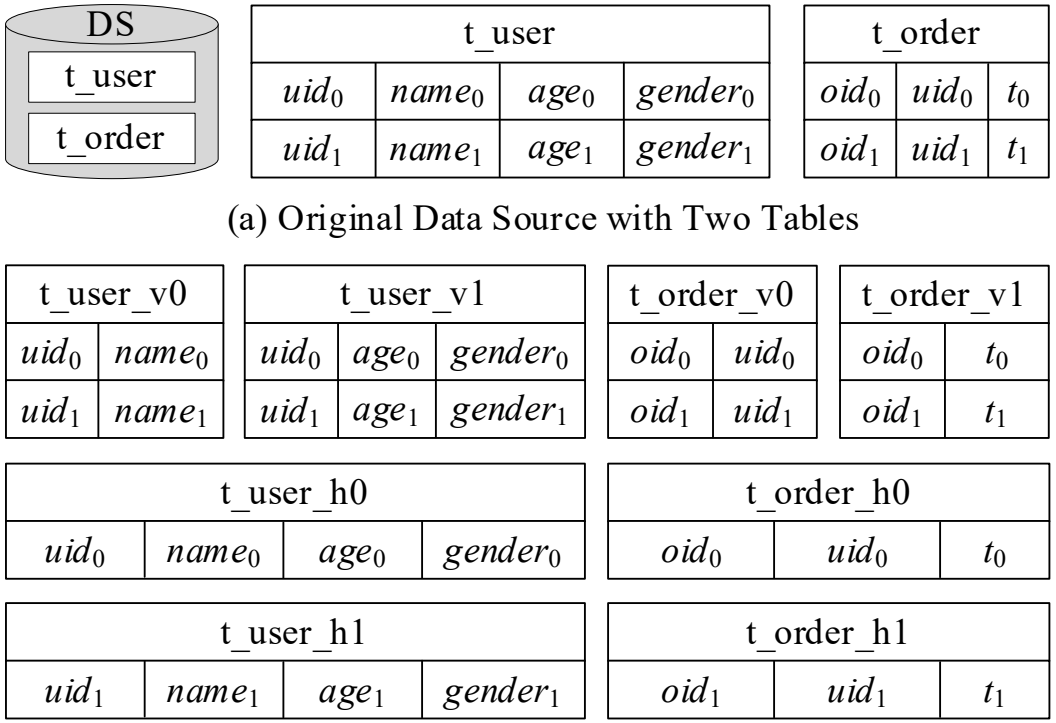


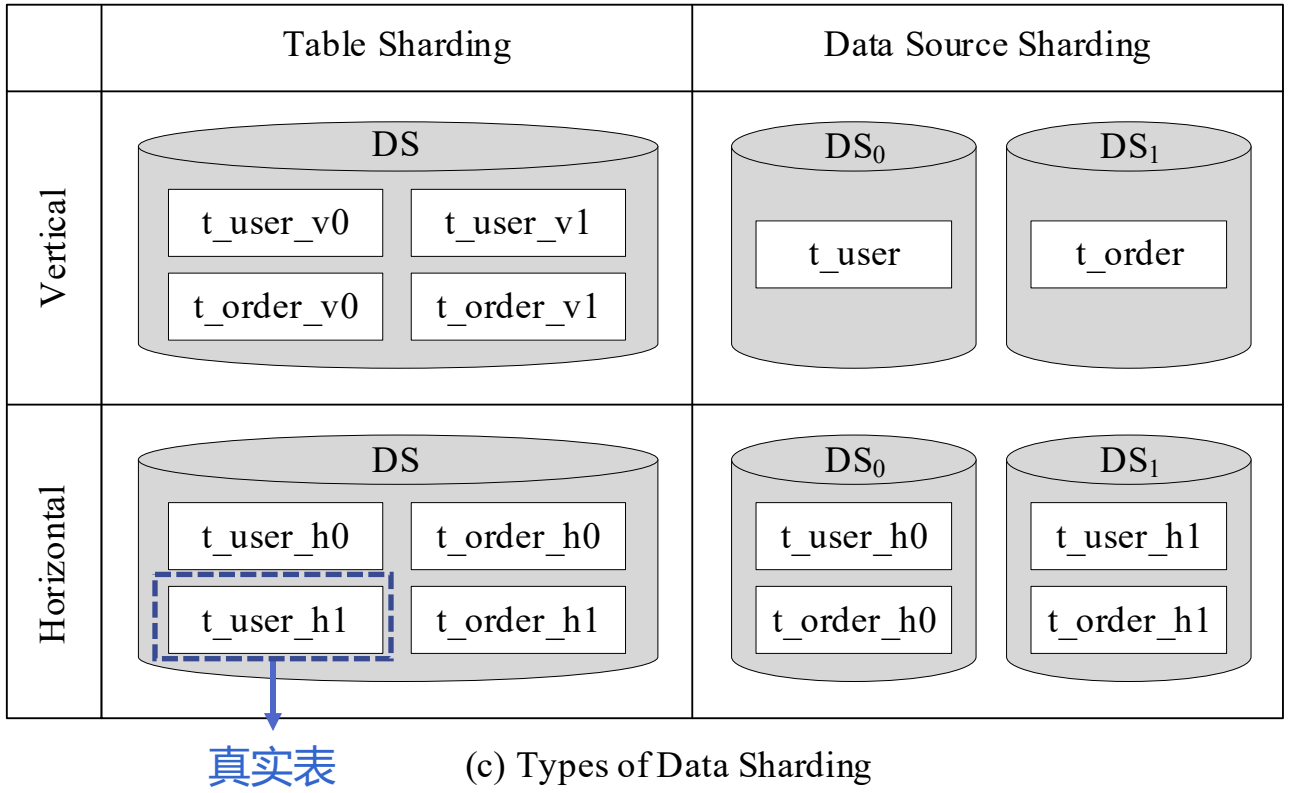
- ShardingSphere-JDBC使用场景
  - 与Java应用程序部署在一起
  - 对开发人员透明
- ShardingSphere-Proxy使用场景
  - 作为单独进程部署
  - 对外表现为关系型数据库
  - 支持任何编程语言
  - 对DBA友好
- 治理模块
  - 作为单独进程部署
  - 监控数据源，存储元数据
  - 维护配置信息

ShardingSphere-JDBC非常高效.

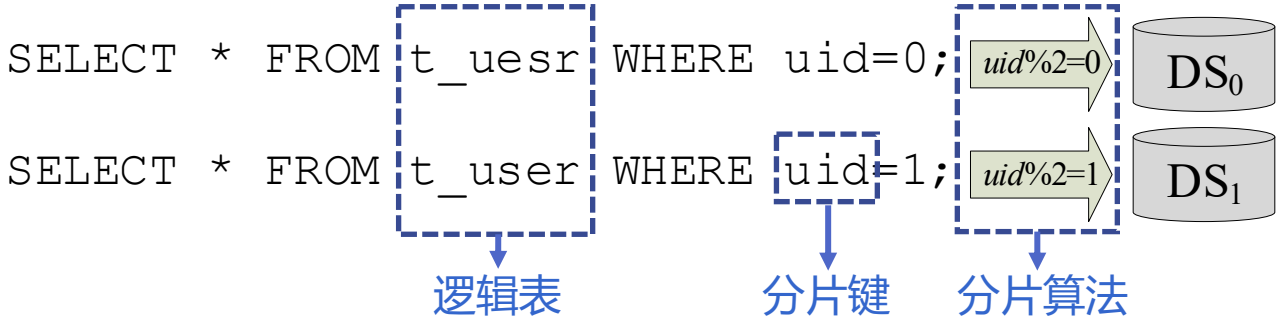
数据分片的关键概念

功能完备





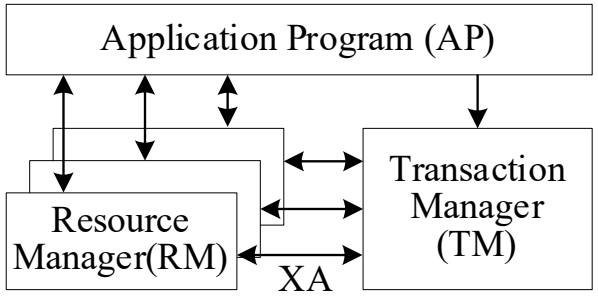
- 数据分片的类型
- 分片键和分片算法
- 逻辑表和真实表
- 数据节点, 例如：DS_0.t_user_h0



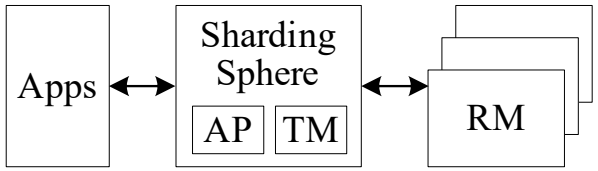
分布式事务

功能完备和用户友好

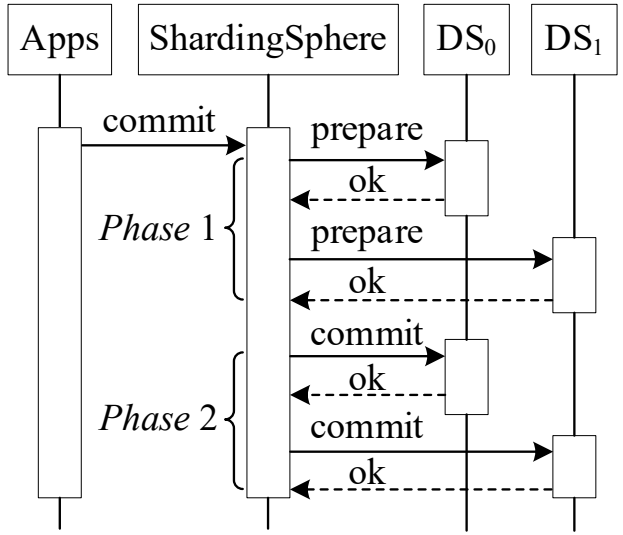
- 针对不同的场景，提供三种类型的分布式事务
  - **XA事务**: 严格的ACID，性能有损耗
  - **本地事务**: 性能极大提升，但不保证一致性
  - **BASE事务**: 性能提升，最终一致性
- 使用数据分片下的分布式使用难点
  - 不同类型分布式事务接口不统一，学习曲线高
- ShardingSphere能让用户像使用单机版事务一样使用分布式事务



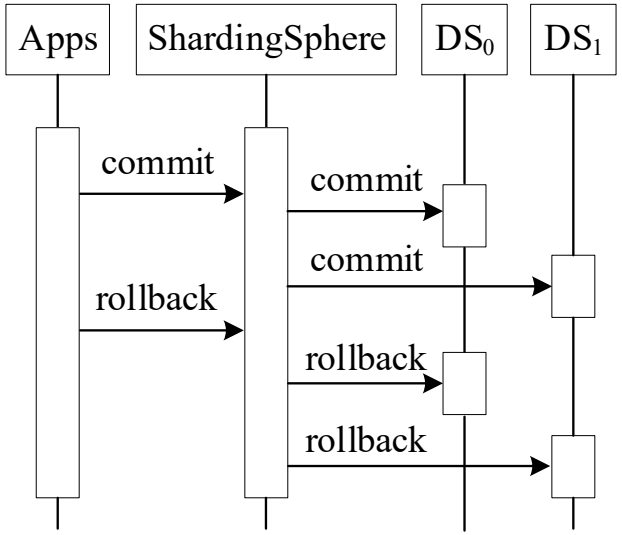
(a) DTP Model



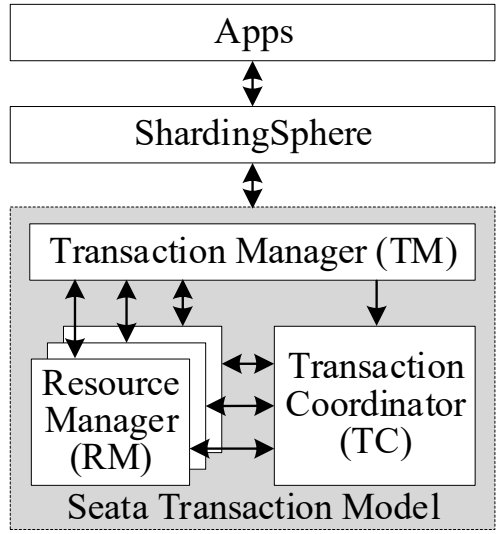
(b) ShardingSphere Acts as AP+TM



(c) XA Transaction



(d) Local Transaction



(e) BASE Transaction Model

DistSQL的配置管理

用户友好

- 现有分片规则的配置
  - 首先创建真实表
  - 然后手动编写配置文件
  - 对开发人员和DBA不友好
  - 生效需要重启应用程序
- DistSQL
  - 类SQL语言
  - 在线生效
- AutoTable
  - 自动分片，让用户不关心真实表存储的位置

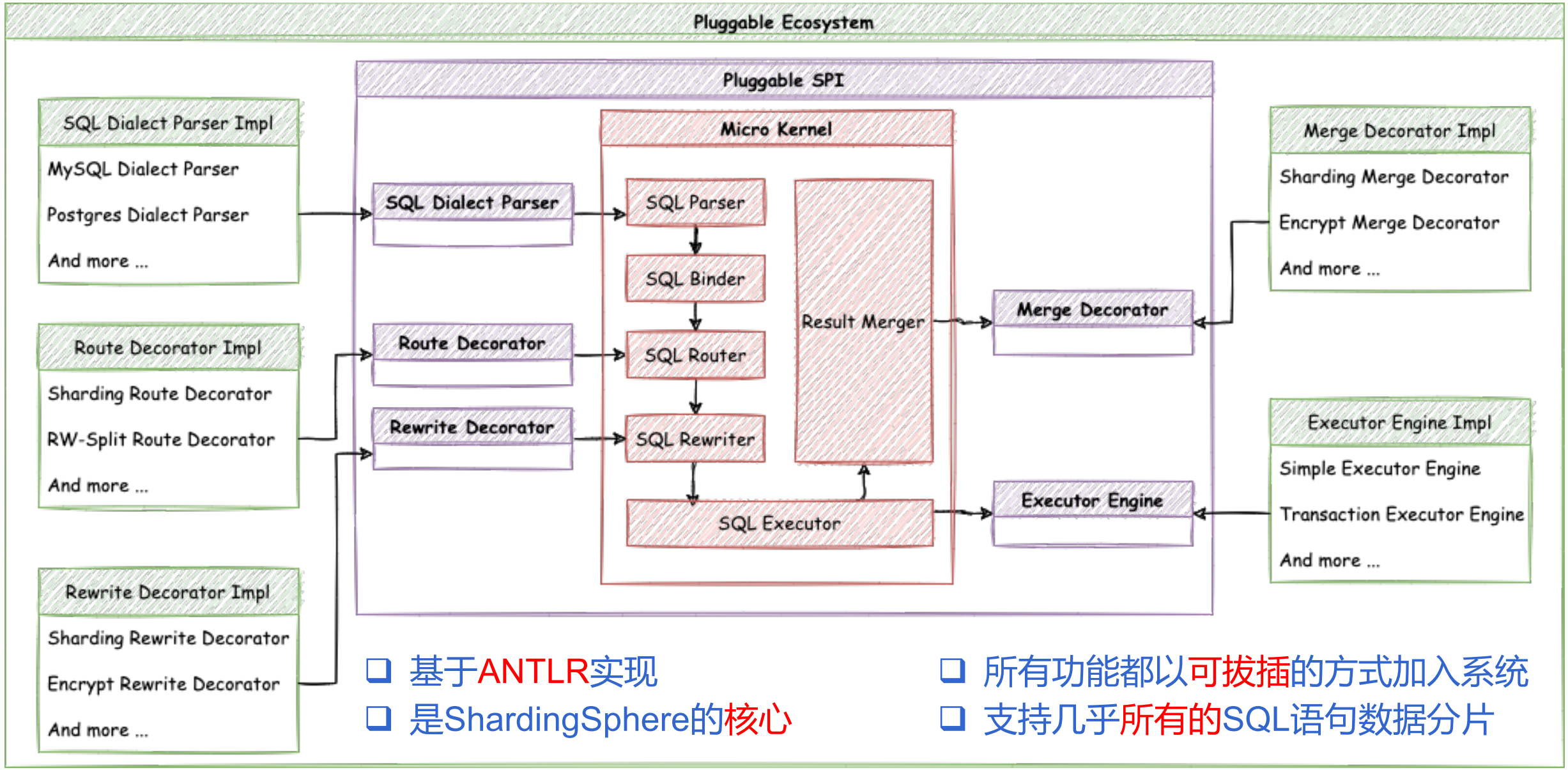
```
rules:
- !SHARDING
  tables:
    t_order:
      actualDataNodes: ds${0..1}.t_order_h${0..1}
      tableStrategy:
        standard:
          shardingColumn: oid
          shardingAlgorithmName: tOrderShardingAlgorithm
  shardingAlgorithms:
    tOrderShardingAlgorithm:
      type: INLINE
      props:
        algorithm-expression: t_order_h${oid % 2}
```

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

Apache ShardingSphere 将自动创建真实表，并自动绑定逻辑表。

完整的SQL引擎

可插拔&可扩展



- 基于ANTLR实现
- 是ShardingSphere的核心
- 所有功能都以可拔插的方式加入系统
- 支持几乎所有的SQL语句数据分片

完整的SQL引擎

- SQL Parser
  - 将SQL语句转化成抽象语法树
  - 预置了多种SQL方言
- SQL Router
  - 将逻辑SQL路由到数据节点中
  - 分类
    - 广播路由：DDL；DCL；TCL；未包含分片键的DQL和DML
    - 分片路由：标准路由和笛卡尔路由

MySQL PostgreSQL ORACLE DATABASE

MariaDB openGauss Microsoft SQL Server ...

```
SELECT * FROM t_user u JOIN t_order o
  ON u.uid = o.uid WHERE uid IN (1,2);
```

标准绑定表路由

```
SELECT * FROM t_user_h0 u JOIN t_order_h0 o
  ON u.uid = o.uid WHERE uid IN (1,2);
SELECT * FROM t_user_h1 u JOIN t_order_h1 o
  ON u.uid = o.uid WHERE uid IN (1,2);
```

笛卡尔路由

```
SELECT * FROM t_user_h0 u JOIN t_order_h0 o
  ON u.uid = o.uid WHERE uid IN (1,2);
SELECT * FROM t_user_h0 u JOIN t_order_h1 o
  ON u.uid = o.uid WHERE uid IN (1,2);
SELECT * FROM t_user_h1 u JOIN t_order_h0 o
  ON u.uid = o.uid WHERE uid IN (1,2);
SELECT * FROM t_user_h1 u JOIN t_order_h1 o
  ON u.uid = o.uid WHERE uid IN (1,2);
```

完整的SQL引擎

SQL Rewriter

- 将逻辑SQL改写成可执行的SQL
- 分类
  - 正确性改写
    - 修改表名、分页、加列、批量插入
  - 优化改写
    - 单节点优化、流式合并优化(见Merger)

加列

```
SELECT oid FROM t_order ORDER BY uid
```

→

```
SELECT oid, uid AS ORDER_BY_DERIVED_0
FROM t_order ORDER BY uid
```

批量插入

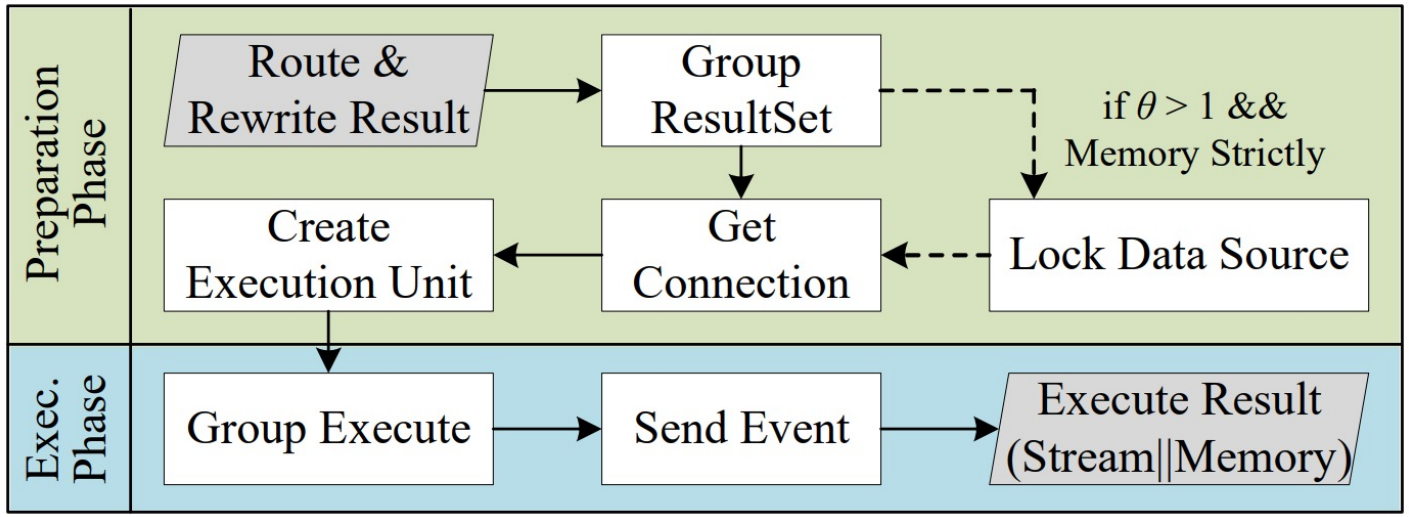
```
INSERT INTO t_order (oid, xxx) VALUES (1, 'xxx'),(2, 'xxx')
```

→

```
INSERT INTO t_order_0 (oid, xxx) VALUES (2, 'xxx')
INSERT INTO t_order_1 (oid, xxx) VALUES (1, 'xxx')
```

SQL Executor

- 将重写后的SQL发到底层存储执行
- 权衡内存消耗和最大连接数
  - 内存限制模式
  - 连接限制模式
- 难点：用户难以决定哪种模式；相同应用不同查询适合模式不同
- ShardingSphere智能选择连接模式
  - $\Theta = 1$，内存限制模式，可用流式合并
  - $\Theta > 1$，连接限制模式，必须内存合并



每个连接执行的SQL数 ← $\theta = \lceil NumOfSQL \div MaxCon \rceil$

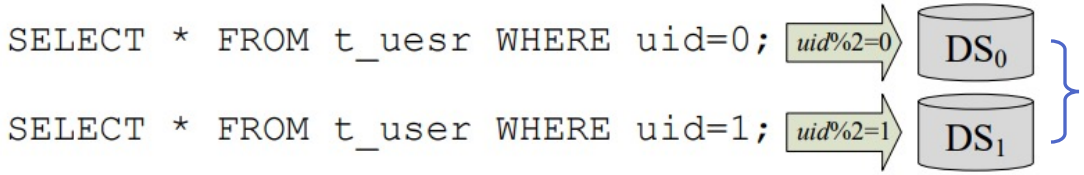
数据库粒度

当前数据库的SQL数　　数据库可用最大连接数/查询

完整的SQL引擎

Result Merger

- 合并多个数据源结果，返回给用户
- 合并方法分类
  - 内存合并
  - 流式合并（优先采用）
- 针对不同的SQL，合并方法不同
  - 简单查询，采用流式合并
  - Order-By查询，多路归并流式合并
  - Group-By查询
    - 包含Group-By和Order-By，且item相同：流式合并
    - 包含Group-By和Order-By，但item不同：内存合并
    - 不包含Order-By：查询重写，自动添加Oder-By，转化成流式合并
  - 聚合查询和分页查询
    - 流式合并或内存合并



t_score0

name	score
Tom	100
Jerry	90
Mary	80

t_score1

name	score
Jerry	95
Tom	85
John	75

t_score2

name	score
John	99
Mary	89
Tom	70

```
SELECT name, SUM(score) FROM
t_score GROUP BY name ORDER BY name
```

(a) Data Records in Data Sources

PriorityQueue (ordered by the first names)

t_score0

name	score
Jerry	90
Mary	80
Tom	100

t_score1

name	score
Jerry	95
John	75
Tom	85

t_score2

name	score
John	99
Mary	89
Tom	70

(b) Get (Jerry, 185)

PriorityQueue (ordered by the first names)

t_score1

name	score
Jerry	95
John	75
Tom	85

t_score2

name	score
John	99
Mary	89
Tom	70

t_score0

name	score
Jerry	90
Mary	80
Tom	100

(c) Get (John, 174)

PriorityQueue (ordered by the first names)

t_score0

name	score
Jerry	90
Mary	80
Tom	100

t_score2

name	score
John	99
Mary	89
Tom	70

t_score1

name	score
Jerry	95
John	75
Tom	85

(c) Get (Mary, 169)

Group-By和Order-By项相同，流式合并示例

实验

数据集

- Sysbench: 2000万~1亿条记录
- TPCC: 200个warehouses (1.2亿条记录)

衡量标准

- 平均响应时间(**AvgT**)
- 99^{th}/90^{th} Percentile Response Time (**99T/90T**)

对比系统

- 传统关系型数据库
  - MySQL v5.7.26 (**MS**), PostgreSQL v10.17 (**PG**)
- 其他数据分片中间件
  - Vitess v12.0.0, Citus v9.0.0
- 新架构数据库
  - TiDB v5.2.0, CockroachDB v21.1.11 (**CRDB**)
- 云数据库
  - Aurora MySQL v2.07.2 (**$Aurora_{MS}$**), Aurora PostgreSQL v4.2 (**$Aurora_{PG}$**)

实验环境设置

- 分布式实验
  - 12台华为云虚拟机
  - CentOS 7.1 64bit, 32-vCore CPU, 64GB RAM and 1TB disk, Linux Multiqueue Networking Enabled
- 云数据库或单机版实验
  - 5台亚马逊云虚拟机
  - Red Hat Enterprise Linux 8.3 64bit, 8-vCore CPU, 64 GB RAM and 100GB SSD
- 每台虚拟机运行之多一个数据库，每个数据库有10个分表

实验

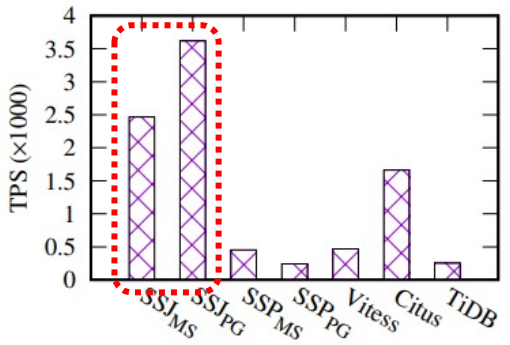
COMPARISON WITH DISTRIBUTED SYSTEMS IN DIFFERENT SCENARIOS (SYSBENCH)

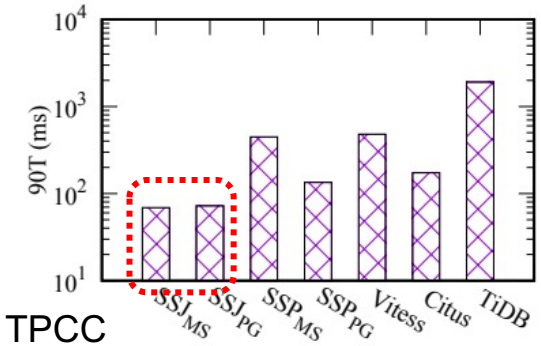
System	Point Select			Read Only			Write Only			Read Write			Update Index			Update Non-index			Delete		
	TPS	99T	AvgT	TPS	99T	AvgT	TPS	99T	AvgT	TPS	99T	AvgT	TPS	99T	AvgT	TPS	99T	AvgT	TPS	99T	AvgT
SSJ_{MS}	**250929**	**1.32**	**0.8**	**50367**	**4.93**	**3.97**	**21163**	**24.58**	**9.45**	**19953**	**32.54**	**10.03**	**50736**	**14.82**	**3.95**	**50632**	**15.06**	**3.95**	**51843**	14.9	**3.86**
SSP_{MS}	185154	2.54	1.08	13165	22.69	15.19	14463	31.37	13.82	7959	48.09	25.12	46207	15.74	4.32	47130	15.09	4.24	49049	**14.64**	4.07
Vitess	155797	4.91	1.28	11806	24.38	16.94	5189	167.44	38.51	3175	189.93	67.87	NA	NA	NA	18638	74.46	10.73	13813	112.67	14.48
CRDB	49225	17.01	4.06	4350	71.83	45.97	2250	404.61	88.86	1611	442.73	124.1	2347	520.62	85.16	23249	38.25	8.6	8380	227.4	23.86
TiDB	141796	7.56	1.41	12140	27.66	16.47	4939	92.42	40.49	3877	101.13	51.58	12171	41.1	16.43	16819	27.17	11.89	27587	28.16	7.25
SSJ_{PG}	**271562**	**1.49**	**0.74**	**171390**	**1.08**	**1.18**	**61015**	**15.04**	**3.28**	**54580**	**14.63**	**3.67**	**100187**	**13.46**	**2**	**156226**	12.78	**1.28**	**174267**	12.15	**1.15**
SSP_{PG}	180357	2.49	1.11	12055	24.09	16.58	25474	19.65	7.85	9121	36.67	21.91	94429	13.94	2.12	138355	**12.23**	1.45	156271	**11.24**	1.28
Citus	51929	12.52	3.85	4288	73.13	46.62	6750	223.34	29.62	3129	277.21	63.89	29584	34.33	6.76	31838	21.11	6.28	37445	17.32	5.34

- 99T, 90T and AvgT are measured in milliseconds, and the best values are marked in **bold**. The same below.
- Vitess does not support updating indexes.

COMPARISON WITH STANDALONE SYSTEMS (SYSBENCH)

System	TPS	99T	AvgT	System	TPS	99T	AvgT
MS	574	1401.61	348.55	PG	1287	337.94	155.27
SSJ_{MS}	**4751**	**152.75**	**42.27**	SSJ_{PG}	**3674**	224.11	**54.61**
SSP_{MS}	380	601.29	555.04	SSP_{PG}	333	816.63	600.23
$Aurora_{MS}$	621	1533.66	289.13	$Aurora_{PG}$	2043	**150.29**	97.89





TPCC

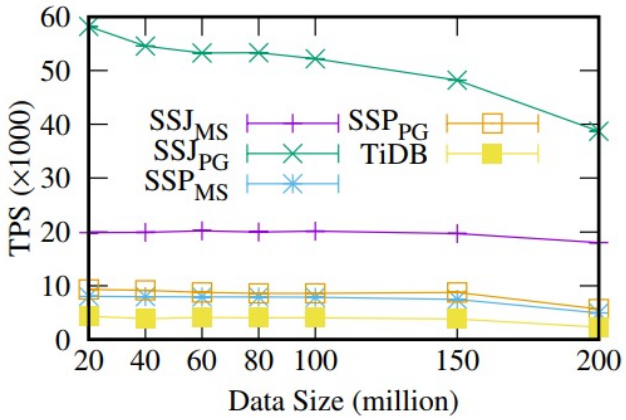
❑ 效率测试

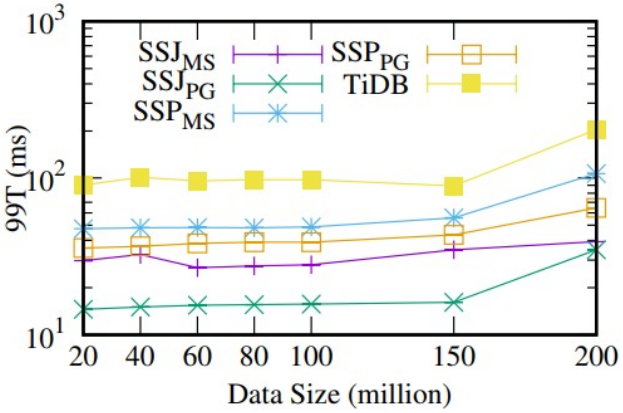
- ❑ 对于两个数据集Sysbench和TPCC，基于SS的方法在TPS、90T/99T、AvgT衡量标准下表现最佳
- ❑ SS-JDBC方法的AvgT比其他系统快2~10倍

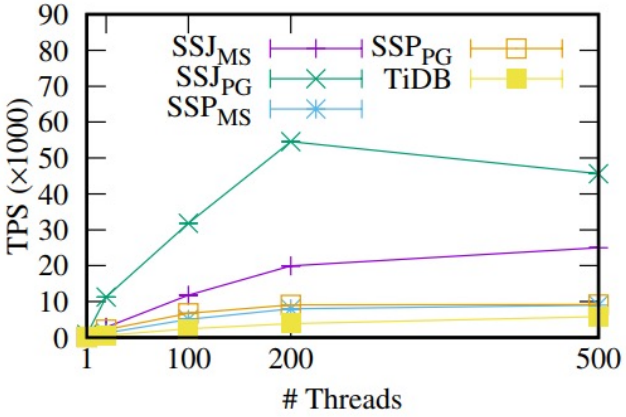
实验

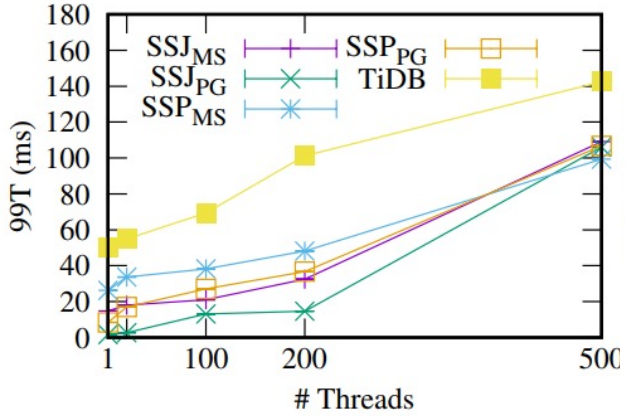
可扩展性测试

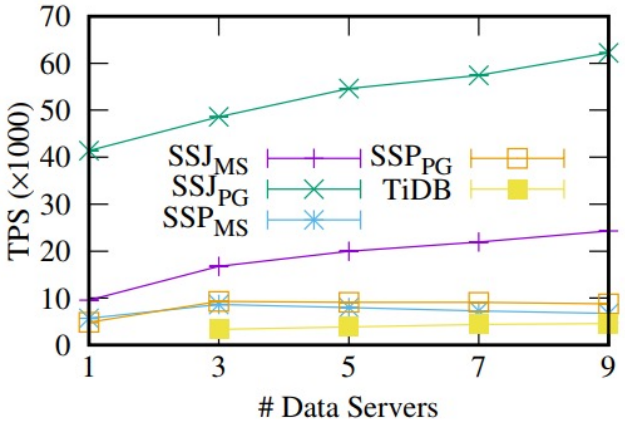
- 基于SS-JDBC的方法表现最佳
- 随着数据量增大，TPS缓慢下降
- 随着请求并发度提高，TPS先升后稳
- 随着服务器数量增加，TPS线性增加

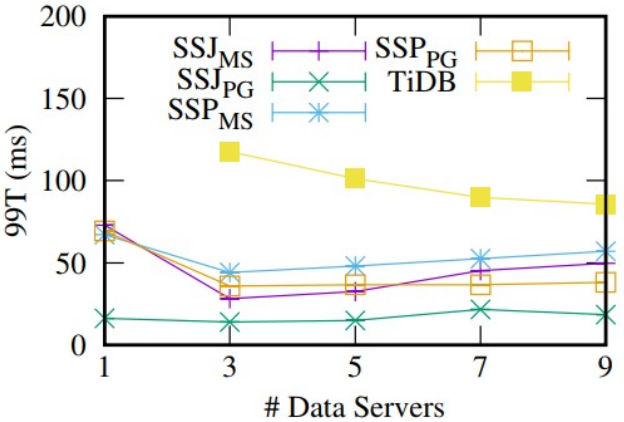












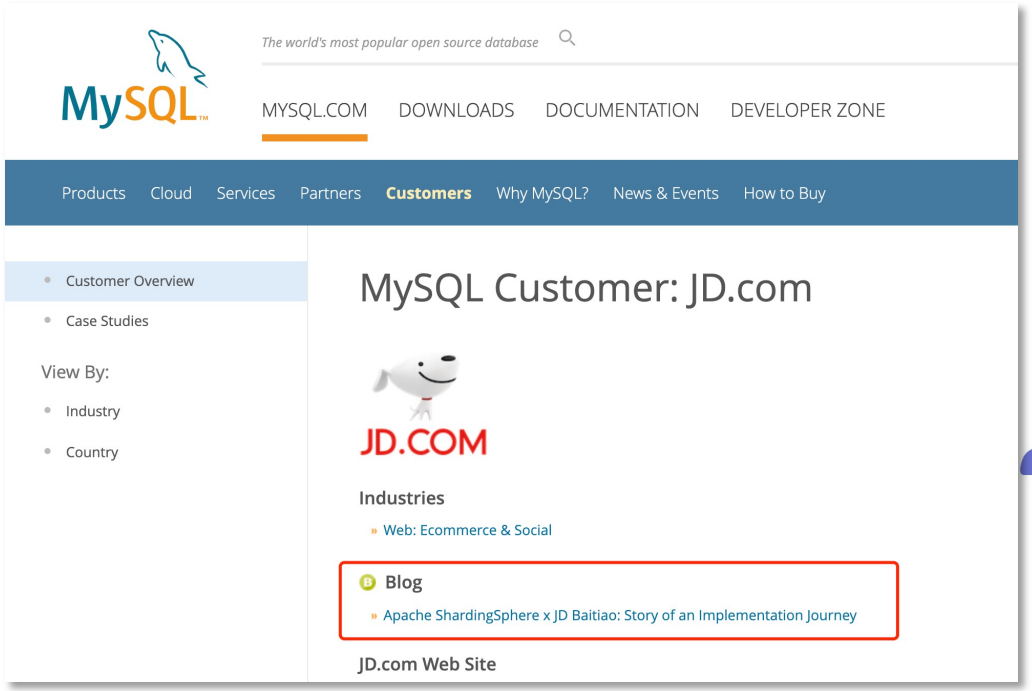
实验代码下载: http://ss4icde.urban-computing.com

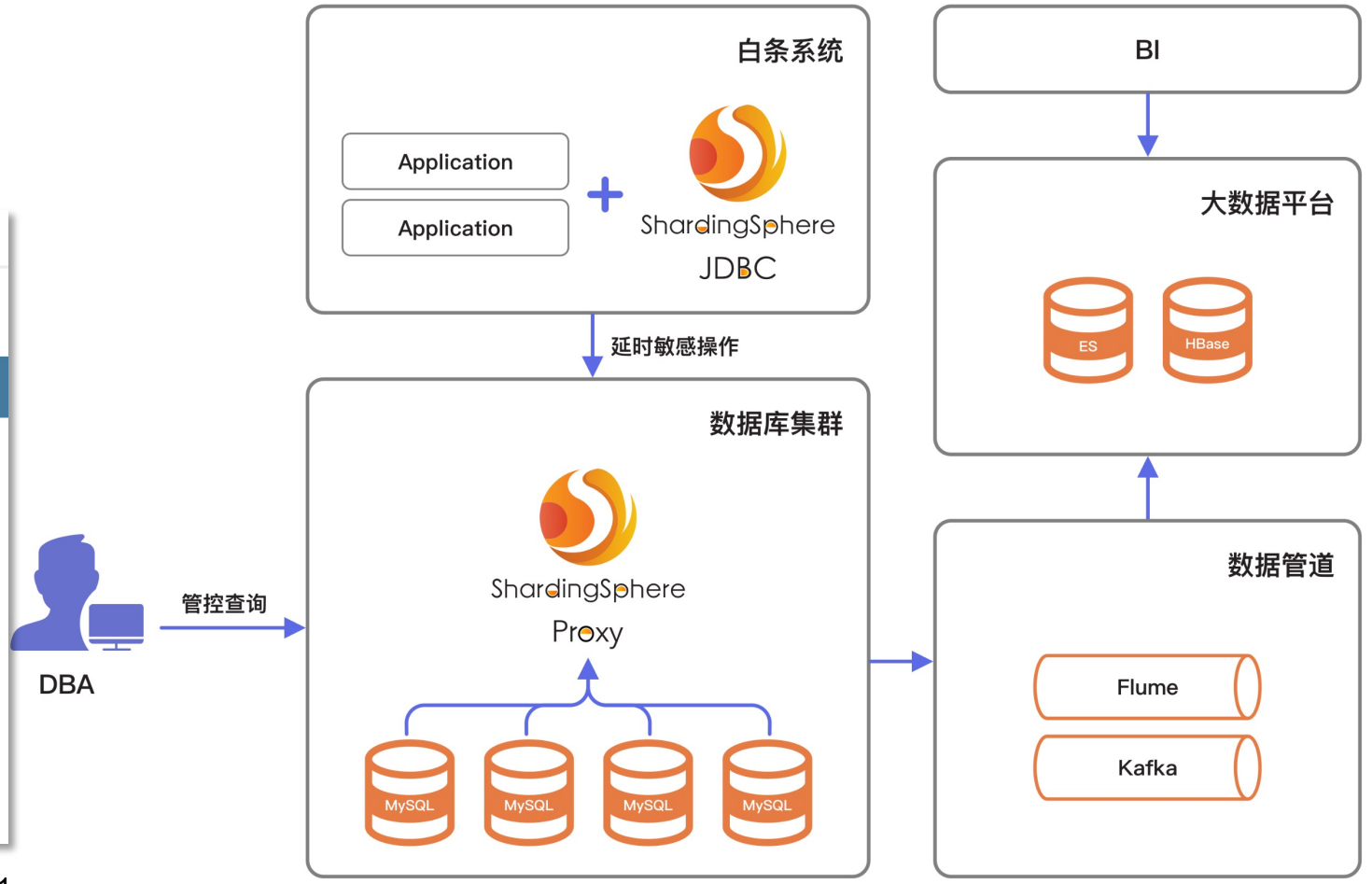
188公司都说在使用ShardingSphere!

...

使用案例: 京东白条

- 京东白条是京东集团旗下的一款信用卡支付产品
- 6400个分片, 超过千亿条数据
- 带来价值:
  - 系统可扩展性更强
  - 降低开发成本





MySQL官方使用案例: https://www.mysql.com/customers/view/?id=1461

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

2021年度Apache年度报告代码提交数量位列前十！

ShardingSphere

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

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

关注公众号，回复ICDE2022_SS_PPT，获取PPT