DTCC

第十一届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2020

架构革新 自主可控

IT168.com ChinaUnix.net ITPUB

北京国际会议中心 | 2020/09/21-09/23

银联分布式存储引擎LAMOST研发实践

中国银联云计算中心 裴晨光

目录

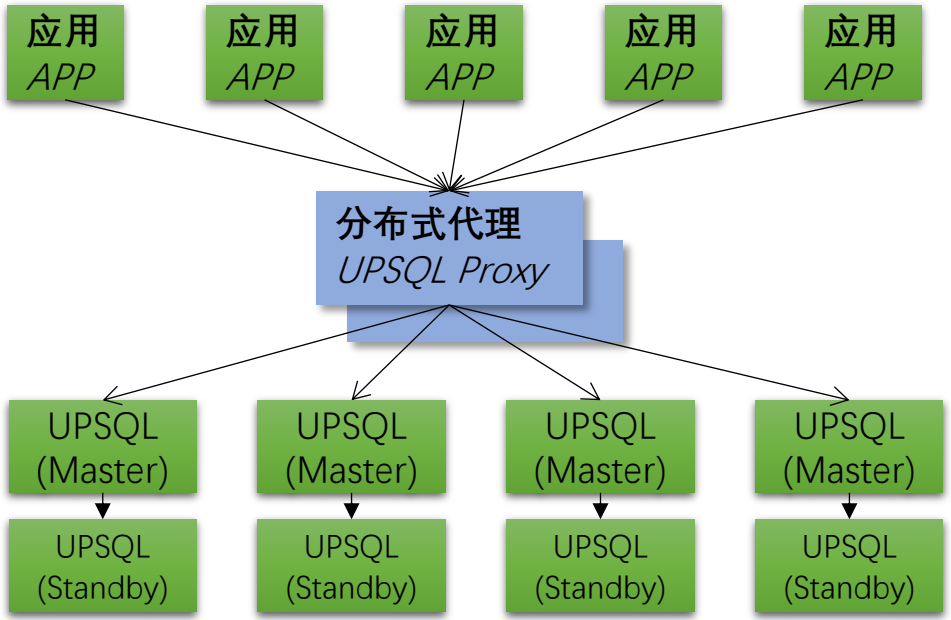
- 根据现状，参考业界标准，可以将分布式数据库分为3条路线，目前共4类产品：

技术路线	含义	特征	产品	混合方案	产品
新架构	基于新架构创建的新型分布式数据库。	设计之初就考虑了分布式特征。具备较高的硬件要求。	OceanBase TiDB	混合存储引擎和中间件方案，在兼容性、稳定性、扩展性上进一步发展。	PolarDB-X UPDRDB
存储引擎	在传统数据库之后，研发分布式存储引擎，以具备分布式数据库关键特性。	与传统数据库具备极好的兼容性，但性能损耗较大。	SequoiaDB-MySQL(巨杉) MariaDB Spider Aurora (by Amazon) PolarDB (by Alibaba) CynosDB (by Tencent)		
透明中间件	在传统数据库之上构建中间件。	具备高性能、低成本和稳定性特征。 但与传统数据库的兼容性最差。	Vitess(from Youtube) Sharding-Sphere UPSQL Proxy TDSQL GoldenDB		

上代产品：UPSQL Proxy架构

自研分布式数据库代理，实现数据库节点高可用、读写分离、数据拆分等。

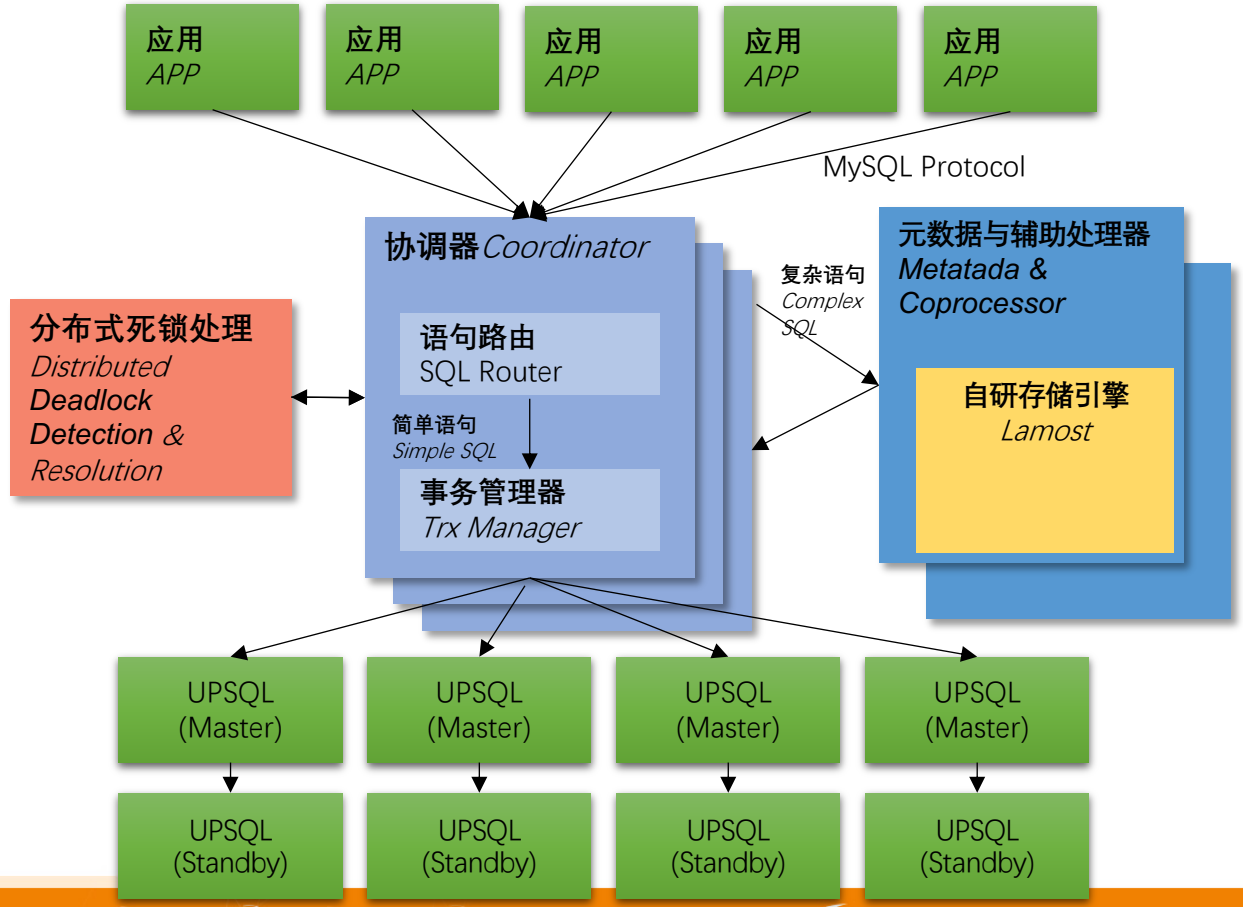
- **其他核心功能**
  - 支持分布式事务
  - 数据库连接池
  - 流式通信处理
- **不足**
  - 分库需要用户配置
  - 缺少在线扩缩容能力
  - 不支持复杂查询
  - 不支持单点ddl



新生代：UPDRDB

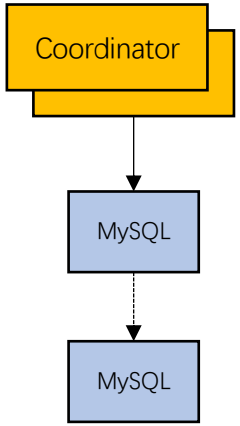
对UPSQL Proxy进行架构升级，采用了独特的语句分类路由与分布式事务管理机制。

- **创新架构优势**
  - 简单语句高性能（优于上代分布式代理产品）
  - 复杂语句支持更为完备
- **其他关键功能**
  - 单点DDL
  - 自动扩缩容
  - 免分库配置

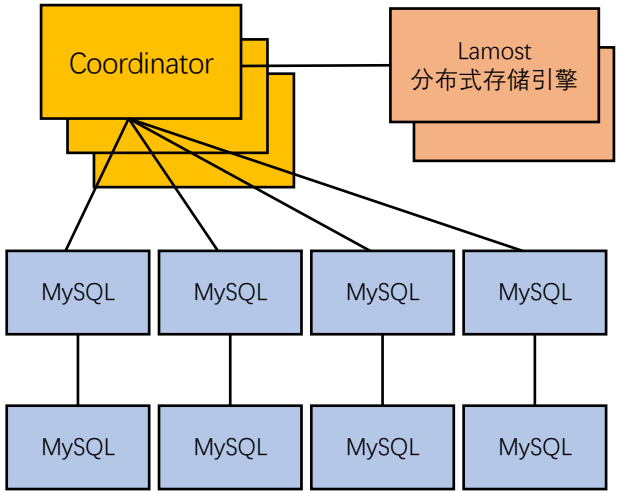


目录

不分片



分片



用户模式

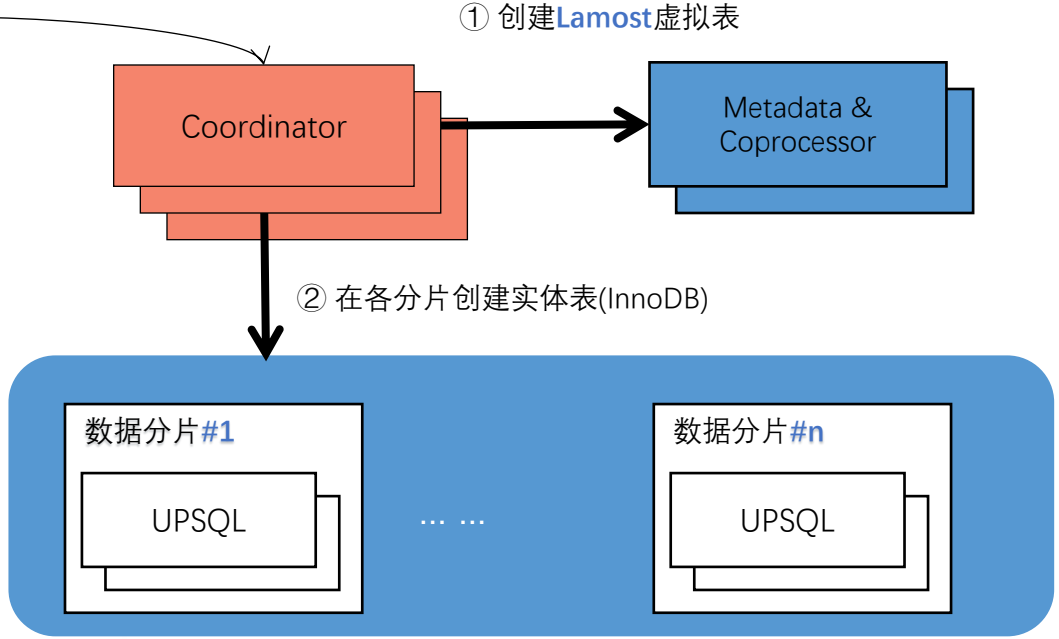
- 读写主库
- 读写分离
- 只读
  - 根据权重分流读，禁止写
- 读根据权重分流
- 仅访问主库节点

SQL语句路由规则

- 简单语句（由协调器处理）
  - 简单DML，一般为插入、更新、删除、无聚合操作的单表查询等
  - 事务管理：begin、commit
- 复杂语句（由辅助处理器处理）
  - 复杂DML，一般为多表操作、有特殊函数的单表查询、视图、UDF等
  - DDL语句
  - show语句

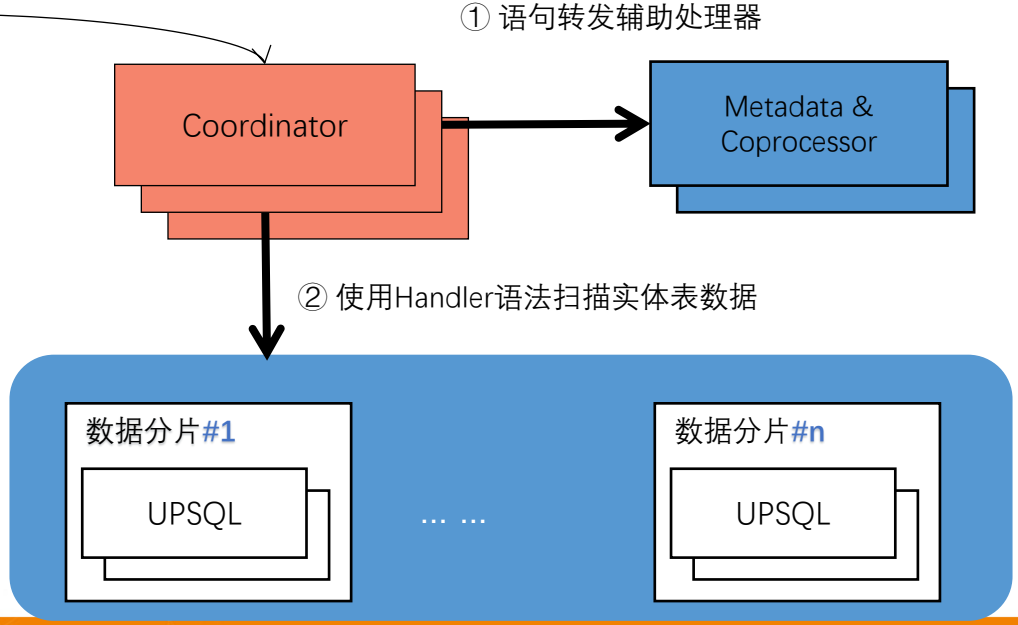
- 通过自研存储引擎Lamost，实现分布式DDL:
  - 协调器(Coordinator)将DDL语句转发给辅助处理器(Coprocessor)
  - 如果操作的表为Lamost引擎，Lamost引擎会将相应操作转换为对各分片的实体表DDL操作

```
create table `st3` (
  `seq` bigint(20) not null default '0',
  `date` char(8) not null default '',
  `time` char(6) not null default '',
  `flag` char(8) not null default '',
  primary key (`seq`,`date`)
) engine=Lamost default charset=utf8
comment= "[seq]" ;
```



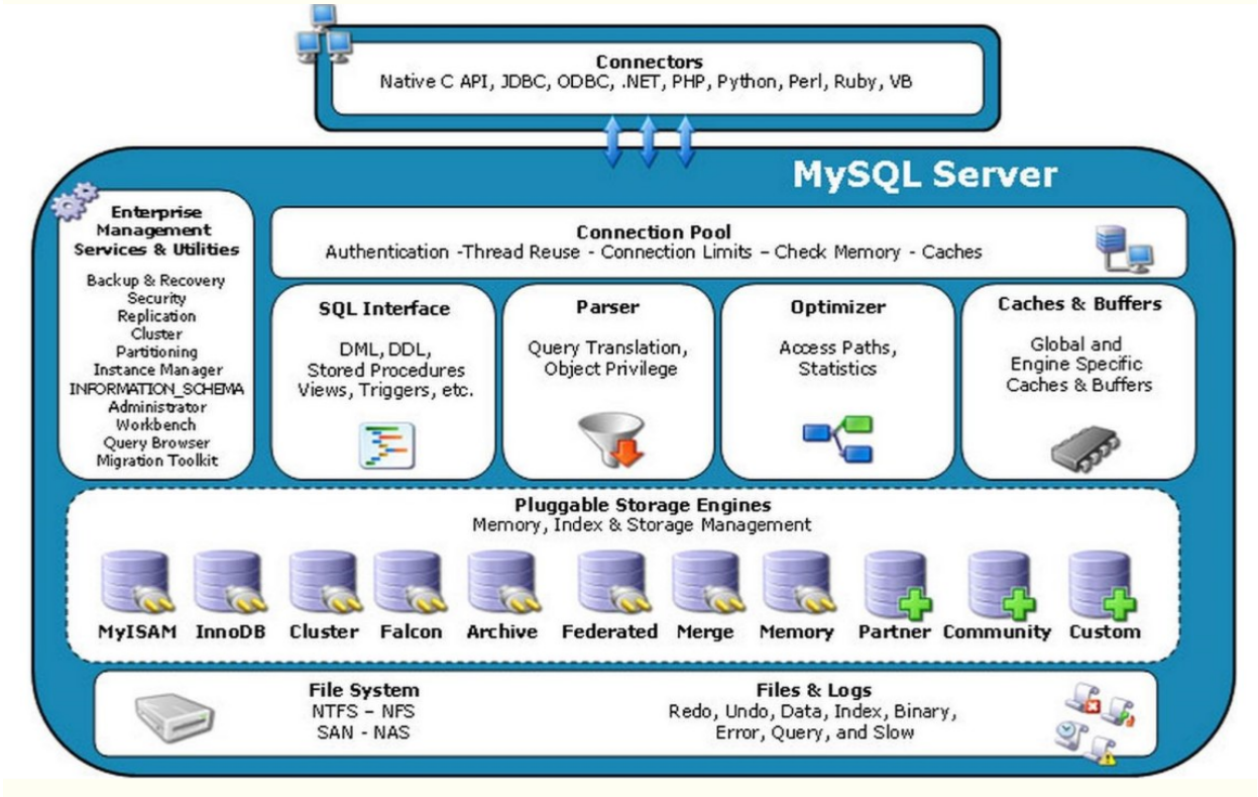
- UPDRDB能支持跨分片join、子查询、视图、UDF等复杂语句，其一般处理流程为：
  - 协调器(Coordinator)将复杂语句转发给辅助处理器(Coprocessor)
  - 如果操作的表为Lamost引擎，Lamost引擎会将相应操作转换为Handler语句，从实体表获取数据
  - 辅助处理器根据获取到的数据进行语句处理，并将最终结果返回协调器

```
select product, `sum`
   from
   ( select product, sum(profit) as 'sum'
      from t1
         group by product with rollup
   ) as tmp
   where product is null;
```



目录

MYSQL体系结构图



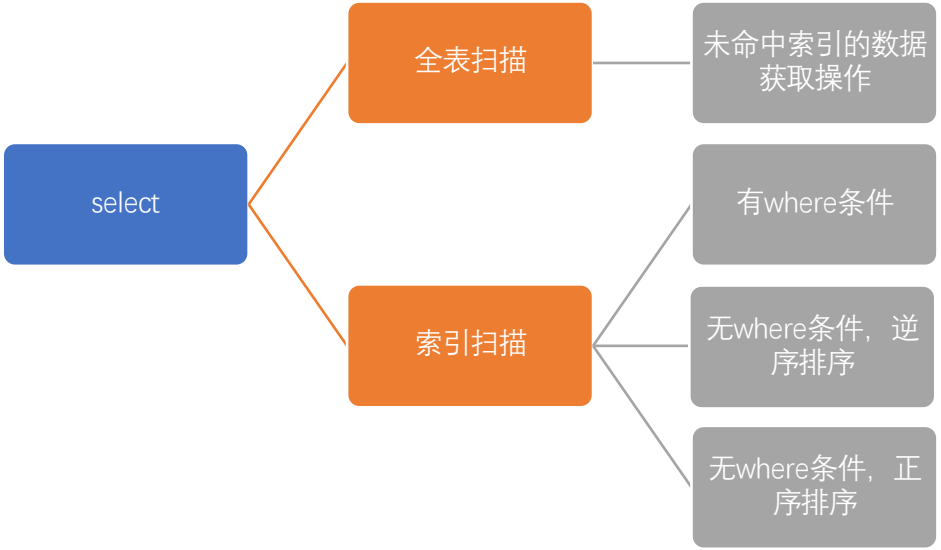
Lamost 存储引擎

注：左图引自网络

- 存储引擎接口是什么？
  - MySQL定义了一系列存储引擎API（ Handler类），以支持插件式存储引擎架构
  - Handler类提供诸如查询数据、写记录、删除记录等基础操作方法
- 存储引擎要做什么？
  - 每一个存储引擎通过继承handler类，实现上述方法，在方法里对底层数据进行操作

读数据接口

- 读数据接口是所有语句类型中变化最多的，涉及面最广，处理逻辑最复杂的
- 不仅常见的select语句是命中这块逻辑，某些增删改场景也会涉及，举例如下：
  - Update语句会先调用读接口，命中到需要的数据，然后对该行数据进行加锁后删除并插入新数据
  - Replace语句会先调用insert接口，尝试插入，如果行记录冲突后，调用查询接口命中数据并加锁，最后执行删除并插入新数据



■ 全表扫描

1. •初始化全表扫描
   int rnd_init (bool scan);
2. •从表中读取下一行
   int rnd_next (byte* buf);
3. •结束全表扫描
   •int rnd_end();
4. •精确获取某行数据
   int rnd_pos(uchar *buf, uchar *pos);

典型SQL举例：

```
1. select * from tbl limit n;
2. select * from tbl where none_index >0 ;
3. select * from tbl order by none_index;
```

实际接口调用截图：

```
ha_lamost::open
ha_lamost::reset
ha_lamost::extra(operation=HA_EXTRA_NO_READCHECK))
ha_lamost::extra (operation=HA_EXTRA_IS_ATTACHED_CHILDREN)
ha_lamost::extra (operation=HA_EXTRA_ADD_CHILDREN_LIST)
ha_lamost::store_lock (lock_type=TL_READ)
ha_lamost::external_lock
ha_lamost::info
ha_lamost::scan_time
ha_lamost::rnd_init
ha_lamost::extra (operation=HA_EXTRA_CACHE)
ha_lamost::rnd_next(多次，直到满足n 或者数据已取尽)
...
ha_lamost::rnd_end
ha_lamost::external_lock
ha_lamost::extra (operation=HA_EXTRA_NO_CACHE)
ha_lamost::extra (operation=HA_EXTRA_DETACH_CHILDREN)
ha_lamost::extra (operation=HA_EXTRA_DETACH_CHILDREN)
ha_lamost::reset
```

无where条件的索引扫描（非range）

- 正序

1. 告知存储引擎选定的索引，遍历方向 •ha_lamost::index_init（keynr=0, sorted=true）
2. 读取索引第一条数据 •ha_lamost::index_first()
3. 循环读取索引下一条数据 •ha_lamost::index_next()

语句举例：

```
select * from tbl order by p_key limit n;
```

- 逆序

1. 告知存储引擎选定的索引，遍历方向 •ha_lamost::index_init（keynr=0, sorted=true）
2. 读取索引最后一条数据 •ha_lamost::index_last()
3. 循环读取索引上一条数据 •ha_lamost::index_prev()

语句举例：

```
select * from tbl order by p_key desc limit n;
```

■ 有where条件的索引扫描（range）

1. 初始化
Ha_lamost::index_init

2. 根据给定的索引进行范围查询
Ha_lamost::read_range_first

3.循环获取索引位置的下一条数据
Ha_lamost::read_range_next

语句举例：
有where语句，且字段为索引字段：

```
select * from tbl where index_key >0 limit 10;
```

实际接口调用：

```
1. ha_lamost::open
2. ha_lamost::reset
3. ha_lamost::extra(operation=HA_EXTRA_NO_READCHECK))
4. ha_lamost::extra ( operation=HA_EXTRA_IS_ATTACHED_CHILDREN)
5. ha_lamost::extra ( operation=HA_EXTRA_ADD_CHILDREN_LIST)
6. ha_lamost::store_lock ( lock_type=TL_READ )
7. ha_lamost::external_lock
8. ha_lamost::info
9. ha_lamost::scan_time
10. ha_lamost::index_init
11. ha_lamost::read_range_first (start_key=0x7fff4c02fc68, end_key=0x0,
    eq_range_arg=false, sorted=false)
12. ha_lamost::read_range_next
13. ha_lamost::index_end
14. ha_lamost::external_lock
15. ha_xproxy::extra(operation=HA_EXTRA_NO_CACHE)
16. ha_xproxy::extra(operation=HA_EXTRA_DETACH_CHILDREN)
17. ha_xproxy::extra(operation=HA_EXTRA_DETACH_CHILDREN)
18. ha_xproxy::reset
```

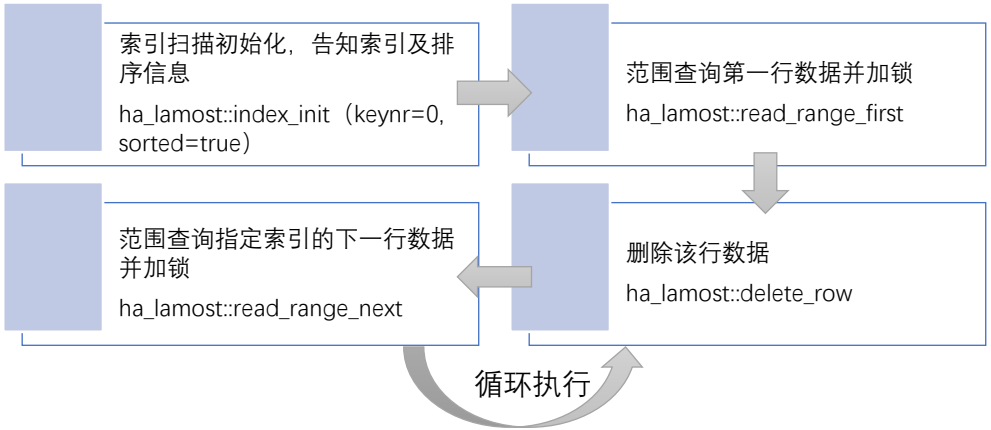
- 方法：
  - 数据查询接口去数据节点获取数据时使用的是mysql的handler语法。
- UPDRDB新增Handler语法：

```
1.  HANDLER tbl_name OPEN [ [AS] alias]
2.  HANDLER tbl_name READ index_name { = | <= | >= | < | > } (value1,value2,...) [ WHERE where_condition ] [LIMIT ... ]
3.  HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST } [ WHERE where_condition ] [LIMIT ... ]
4.  HANDLER tbl_name READ { FIRST | NEXT } [ WHERE where_condition ] [LIMIT ... ]
5.  HANDLER tbl_name READ index_name { = | <= | >= | < | > } (value1,value2,...) { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ] （新增语法）
6.  HANDLER tbl_name CLOSE
```

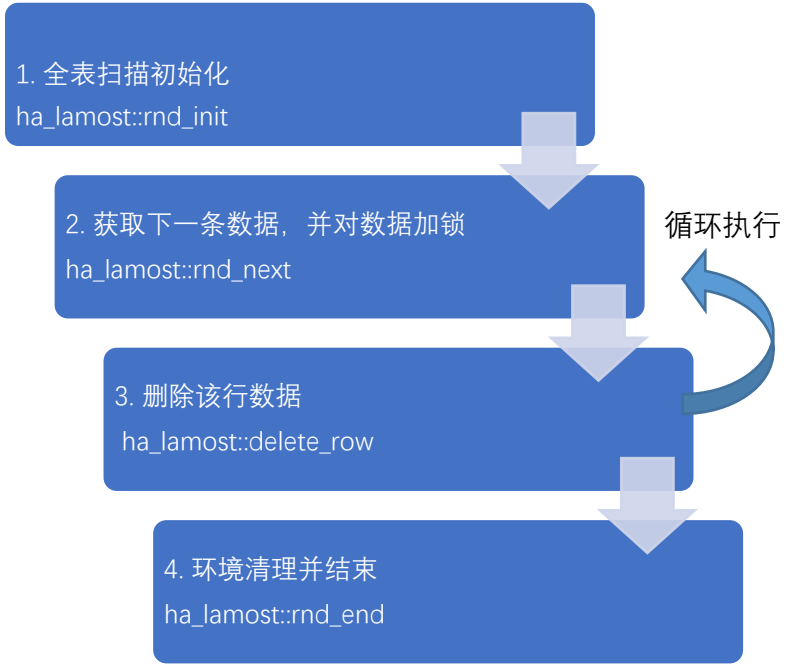
- 多表关联慢么？
  - 语句举例：
    - select * from tbl_1 inner join tbl_2 on tbl_1.p1_key=tbl_2.p2_key；（假设tbl_1两百万数据，tbl_2两百行数据）
- 存储引擎调用逻辑：
  1. 准备工作（打开表等）
  2. 对tbl_2进行全表扫描，每次查询获得tbl_2中的一条数据后，去tbl_1中进行查询，检测是否命中该行数据，如果命中数据则返回该条数据到server层
  3. 循环执行步骤2，直至tbl_2中所有数据遍历完成
  4. 收尾工作（关闭表等）
- 发生灾难：
  - 如果tbl_2也是两百万行数据
  - 如果p1_key不是索引，或者代价模型计算有误，未命中该索引，则会全表扫描

DELETE接口

- 命中索引
  - 语句举例:
    - delete from tbl where index_col>0 limit n;

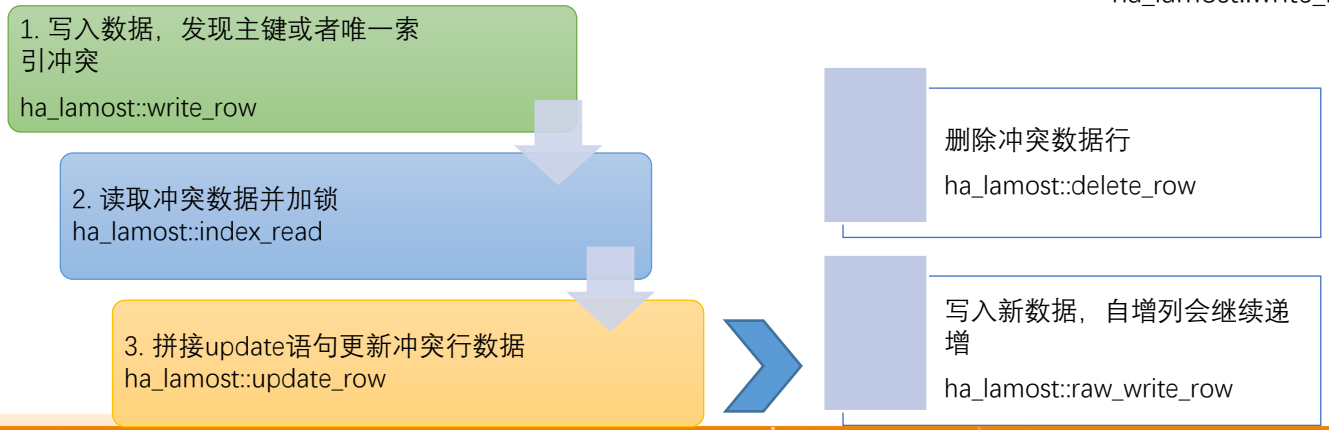


- 未命中索引
  - 语句举例:
    - delete from tbl where none_index >0 limit n;



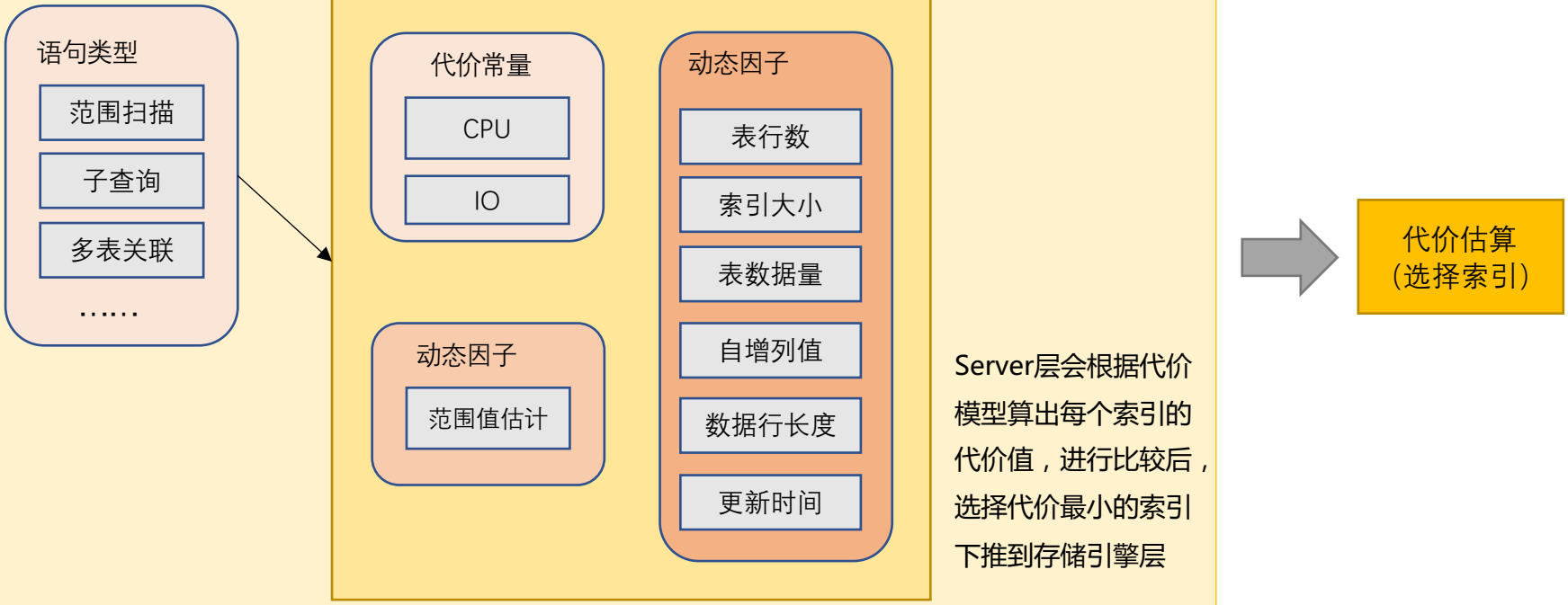
REPLACE接口

- 数据有冲突
- 语句举例:
  - replace into tbl (p_key,u_key,shardcol) values (4, 33,'2002');

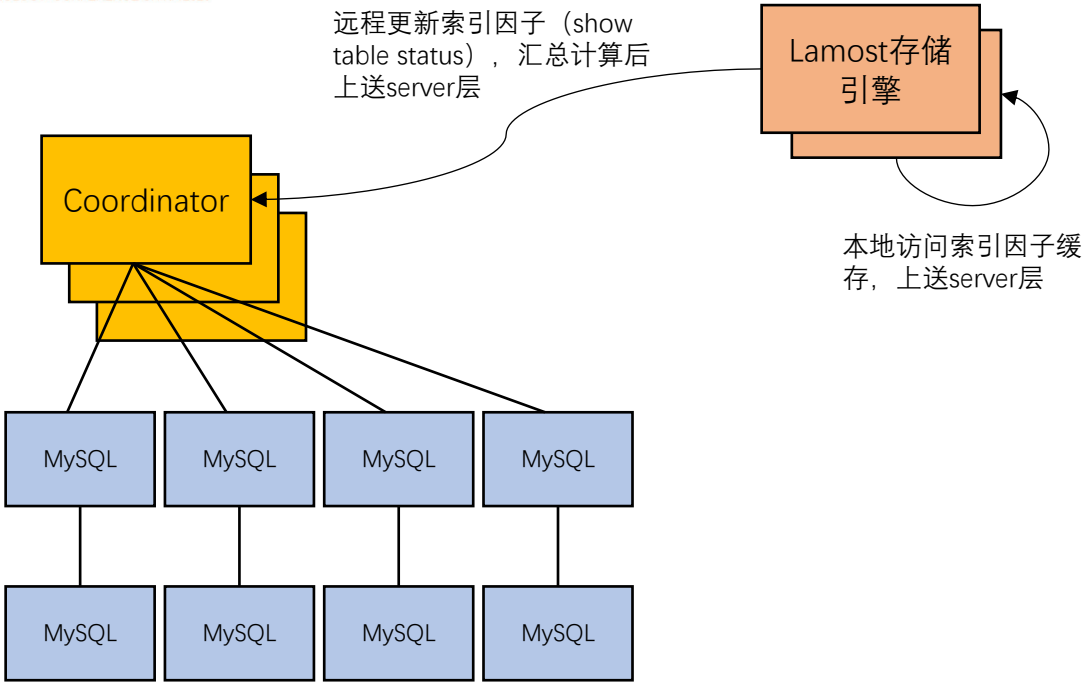


- 数据无冲突
- 语句举例:
  - replace into tbl (p_key,u_key,shardcol) values (4, 33,'2002');
- 逻辑：调用写数据接口，与insert逻辑相同
  - ha_lamost::write_row

代价模型



Server层会根据代价模型算出每个索引的代价值，进行比较后，选择代价最小的索引下推到存储引擎层



索引因子上送策略：

- 未触发更新条件则访问本地缓存，该场景命中绝大多数情况
- 索引因子需要更新，则下发自定义命令

索引更新触发条件：

- 超过指定的超时时间（参数可配）
- 部分DDL语句执行之后

实现效果：

- 绝大部分语句索引选择能与单机mysql保持一致
- 基本不影响复杂语句执行性能

目录

• SQL黑白名单

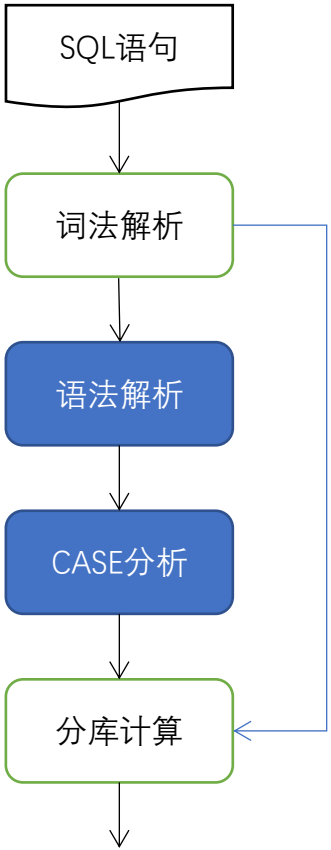
- 防止未授权的SQL语句在数据库中执行，解决高危SQL执行的风险。
- 根据语句类型或语句摘要，由用户进行标识和分类：
  - 灰：一般为新增语句或类型
  - 白：可放行语句
  - **黑**：禁止执行
- 在用户维度配置三种策略：
  - 白名单
  - 灰名单可执行
  - 灰名单执行并报警

• SQL执行统计

- 统计一定时间周期内（如5秒）内，相同摘要语句：
  - 执行次数
  - 最短、最长、平均耗时
  - 报错码及次数
  - 返回行数
- 平台或应用可以基于上述数据，进一步感知系统运行状态

相似性SQL专利

- SQL解析分为词法解析和语法解析，整个解析过程十分占用性能，约整体性能的50%左右
- 受语句分类路由机制的启发，我们发现并实现了一种相似性SQL复用机制，并获得了美国专利授权。
- 该机制能让相似语句规避最为耗时的语法解析与CASE分析等阶段，使得SQL解析与分布式执行计划构建的整体性能提升至少1倍以上。



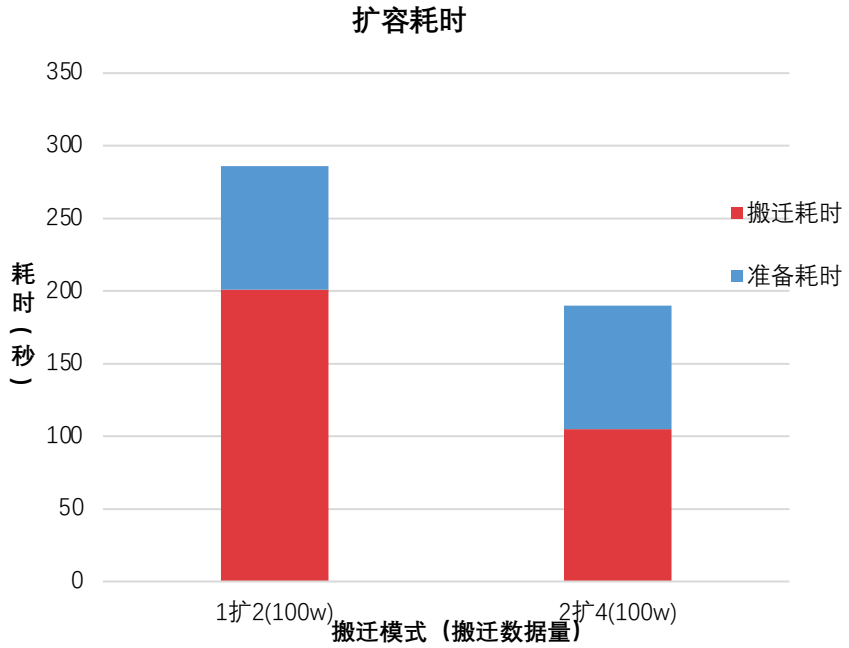
自动化扩缩容

基本原理

1. 将数据分为1024个slot，在辅助处理器上维护slot和数据节点之间的映射关系。
2. 在扩缩容时，调整slot与数据节点之间的映射关系，并使用分布式事务进行数据搬迁。
3. 在数据搬迁过程中，使用2套映射关系的并集进行数据查询。

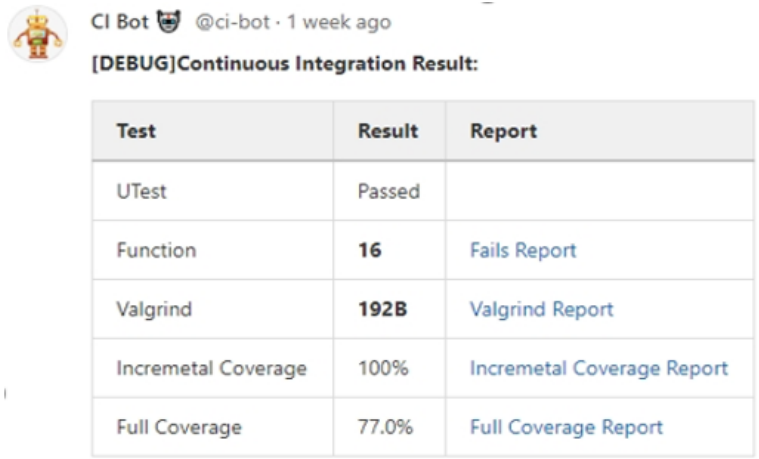
能力

1. 支持在线扩缩容
2. 支持任意分片数量的扩容和缩容
3. 支持指定搬迁数据比例



代码质量保证

- 得益与UPDRDB对复杂语句的全面支持，我们引入了67个MySQL测试案例集，总案例集数量达到172+。
- 自动化全量执行耗时大于48小时，所以又研发了并行执行工具，构建CI，提升研发效能。
- 以CI为抓手规范研发流程，共执行CI328次，全量代码覆盖率从最初61%提升到80.7%。



UPDRDB的CI机器人

目录

UPDRDB近期技术状态总结

- 支持视图
- 支持UDF
- 支持触发器
- 支持存储过程
- 支持子查询
- 支持表关联
- 支持动态扩缩容
- 支持单点DDL
- 支持分库免配置

- 不支持分布式全局索引
- 不支持Range的拆分方式

高质量

通过引入MySQL测试案例集，采用CI等手段，显著提高了产品质量。

高易用

支持分布式DDL、复杂语句、自动扩缩容等，显著提升了产品的易用性。

高性能

基于创新架构，我们继承了简单语句处理的高性能优势，并进一步叠加了性能优化方案。

高安全

一方面采用了更严格的用户管理与会话管理体系；另一方面，引入了SQL指纹技术进一步强化了安全特征。

银联云，具有四大特色


稳定可靠
Stable and reliable
金融级安全
Financial security
开放共享
open and share
自主可控
Independently controllable

中国银联期待你的加入

数据库开发方向

开源组件方向

THANKS