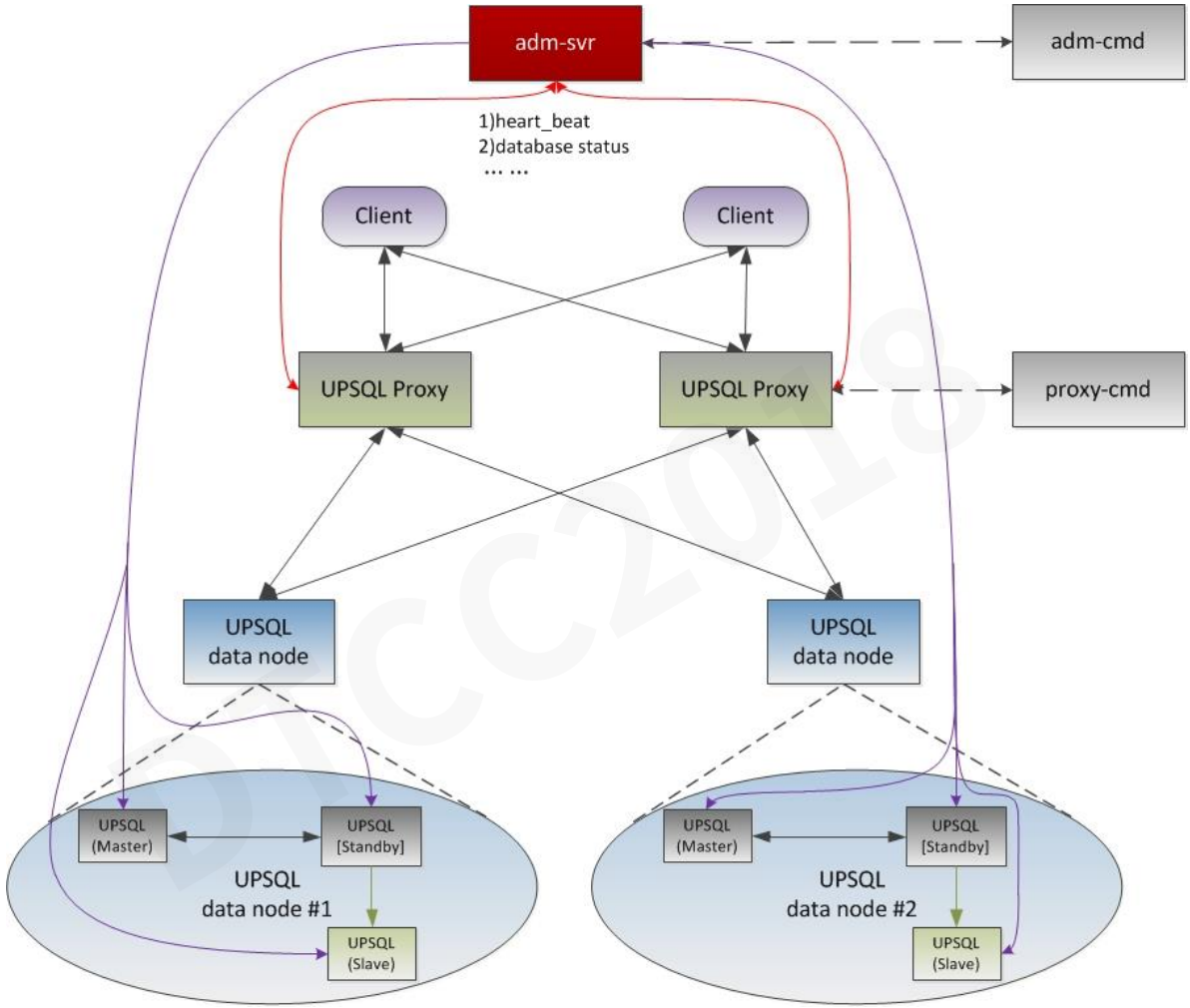
中国银联分布式数据库实践

中国银联 周家晶

分布式数据库

初始架构



Spanner VS Proxy

Raft

SQL

MapReduce

Database

CAP

NoSQL

BASE

OLTP

OLAP

TrueTime

Paxos

ACID

NewSQL

MVCC

MPP

2PC

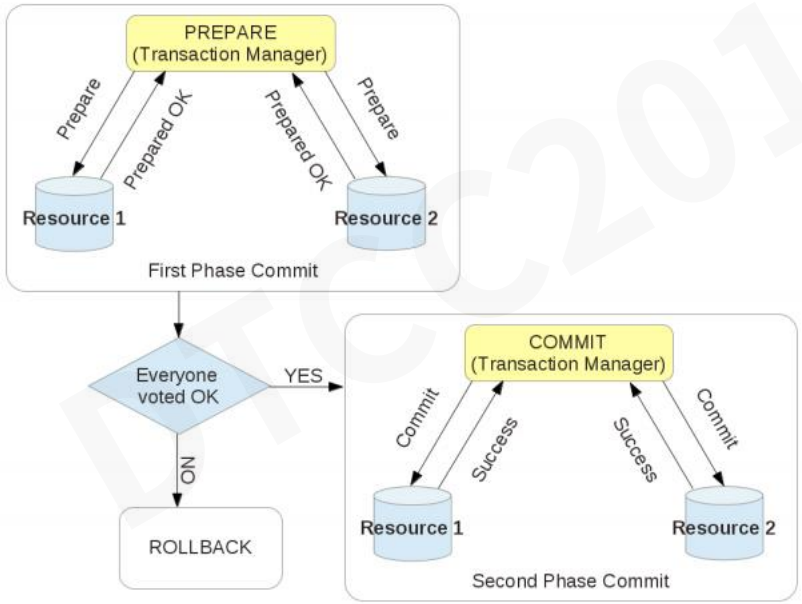
MGR

Distributed Index

External Consistency

分布式事务

- 核心思路
  - 使用后端数据库存储XA日志
  - 使用最后参与者策略



分布式事务 \ 方案对比

		友商方案			银联方案	
Client		Set_1	Set_2		Datanode_1	Datanode_2
begin;						
insert into t1 values(1);		xa start 'xa-gtid-1';			xa start 'xa-gtid-1';	
		insert into t1 values(1);			insert into t1 values(1);	
insert into t1 values(2);			xa start 'xa-gtid-1';			xa start 'xa-gtid-1';
			insert into t1 values(1);			insert into t1 values(1);
commit;		xa end 'xa-gtid-1';	xa end 'xa-gtid-1';		insert into xa.commit_log...;	xa end 'xa-gtid-1';
		xa prepare 'xa-gtid-1';	xa prepare 'xa-gtid-1';		xa end 'xa-gtid-1';	xa prepare 'xa-gtid-1';
		insert into xa.commit_log...;			xa commit 'xa-gtid-1' one phase;	
		xa commit 'xa-gtid-1';	xa commit 'xa-gtid-1';			xa commit 'xa-gtid-1';

- 主要优势
  - 不需要使用单独会话进行xa日志记录 （友商方案中向xa日志表的写入是自动提交的另一个会话）
  - 减少一个两阶段事务
- 主要缺点
  - 涉及分布式事务的后端用户，都需要配置xa日志表权限，因而存在运维风险。

分布式事务 \ 分布式死锁

- 分布式死锁：
  - 场景（互斥、请求与保持、不剥夺、循环等待）：
    - a & b 开启分布式事务
    - time 1: a write db1.resource1
    - time 2: b write db2.resource2
    - time 3: a write db2.resource2 -> 锁：a 等待b的 db2.resource2
    - time 4: b write db1.resource1 -> 锁：b 等待a的 db1.resource1
  - db1和db2上的本地写锁，在分布式事务场景下，出现了循环等待，成为了分布式死锁
- （与传统数据库一样）死锁的解决路径：
  - 死锁检测：剥夺
  - 死锁预防：避免循环等待

分布式事务 \ 死锁检测

- 冲突图(Conflict Graph)
- 死锁检测的核心是获取锁信息，但MySQL没有提供XAID与锁的关联信息。
- 解决方案：扩展innodb_trx表，增加XAID信息，与innodb_locks关联即可进行分布式死锁检测

```
mysql> desc innodb_trx;
+---------------------------+---------------------+------+-----+---------------------+-------+
| Field                     | Type                | Null | Key | Default             | Extra |
+---------------------------+---------------------+------+-----+---------------------+---+
… …
| trx_xa_format_id          | int(10)             | NO   |     | 0                   |       |
| trx_xa_gtrid_length       | int(2)              | NO   |     | 0                   |       |
| trx_xa_bqual_length       | int(2)              | NO   |     | 0                   |       |
| trx_xa_data               | tinyblob            | NO   |     | NULL                |       |
+---------------------------+---------------------+------+-----+---------------------+-------+
28 rows in set (0.01 sec)
```

分布式事务 \ 死锁预防

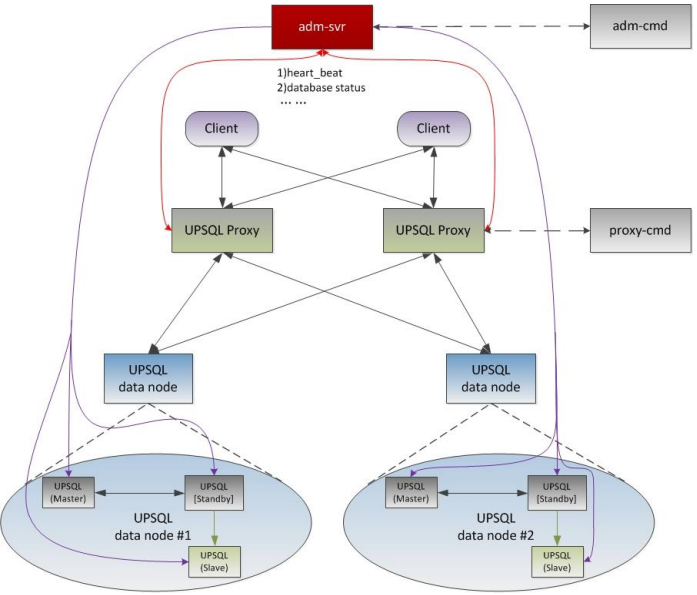
- 分布式死锁预防策略：时间戳(Timestamp ordering, TO)
  - 保证单一时序的锁等待(只接受新等旧，或旧等新)
- 实现方案如下：
  - 带有时间戳的XAID生成规则：
    - 事务开始时间 + TM(proxy)事务序号 + TM编号 + datanode + ...
  - 修改MySQL死锁检测策略，增加分布式死锁预防策略
    - 在DeadlockChecker::search() 中获取xaid，根据死锁预防策略进行处理

分布式事务\进一步优化

		当前方案			改进方案	
Client		Datanode_1	Datanode_2		Datanode_1	Datanode_2
begin;						
insert into t1 values(1);		xa start 'xa-gtid-1';			xa start 'xa-gtid-1';	
		insert into t1 values(1);			insert into t1 values(1);	
insert into t1 values(2);			xa start 'xa-gtid-1';			xa start 'xa-gtid-1';
			insert into t1 values(1);			insert into t1 values(1);
commit;		insert into xa.commit_log...;	xa end 'xa-gtid-1';		insert into xa.commit_log...;	xa prepare 'xa-gtid-1';
		xa end 'xa-gtid-1';	xa prepare 'xa-gtid-1';			
		xa commit 'xa-gtid-1' one phase;			xa commit 'xa-gtid-1' one phase;	
			xa commit 'xa-gtid-1';			xa commit 'xa-gtid-1';

- 最后参与者退化为一阶段提交，但仍然有多一次xa end操作，可将该步骤省略。
- 实现方法
  - 修改bool Sql_cmd_xa_commit::trans_xa_commit(THD *thd)
  - 增加预处理操作：当m_xa_opt == XA_ONE_PHASE时，将事务状态XA_ACTIVE直接修改为XA_IDLE
- 同样的，我们也可以将xa prepare前的xa end省略掉。

分布式事务 \ 架构演变

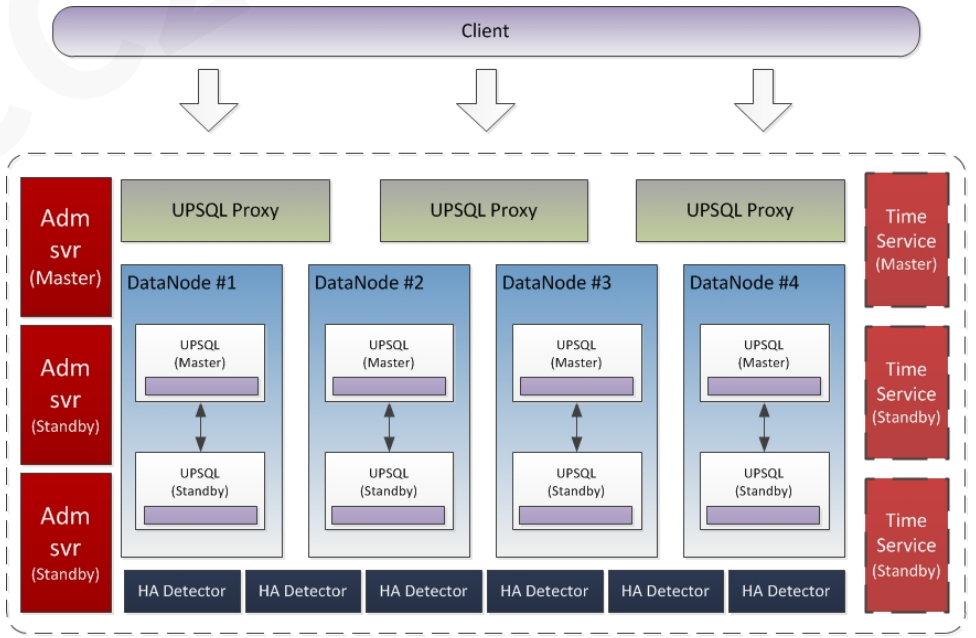


初期

- 数据库高可用
- 读写分离
- 数据拆分
- stmt协议(prepare)

中期

异步连接池 -
分布式事务 -



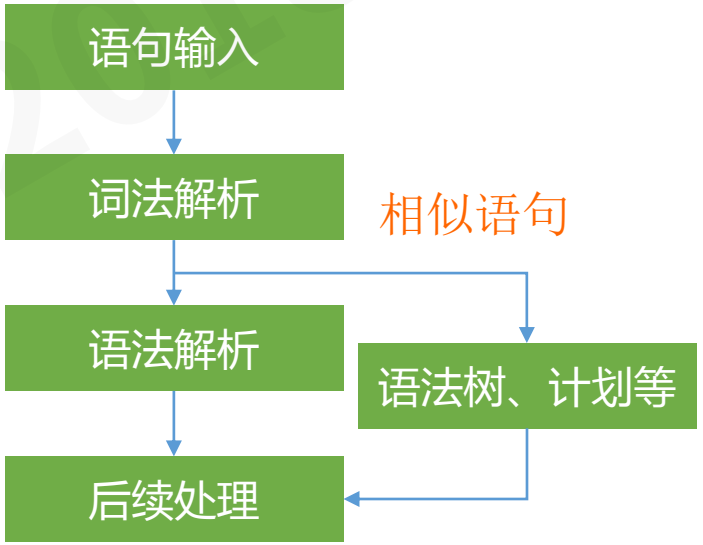
XA Distributed Log

SQL解析优化

- 为避免硬解析，业界已提供了两种方法：
  - Prepare：做一次语句解析后，使用语句编号与变量值进行交互
  - 软解析：相同的语句复用语法解析树、执行计划等。

SQL解析优化 \ 相似性优化

- 相似性解析优化：在词法解析后进行相似性分析来复用语法解析树、逻辑执行计划等。
- 如何进行相似性分析？
- 如何进行复用？



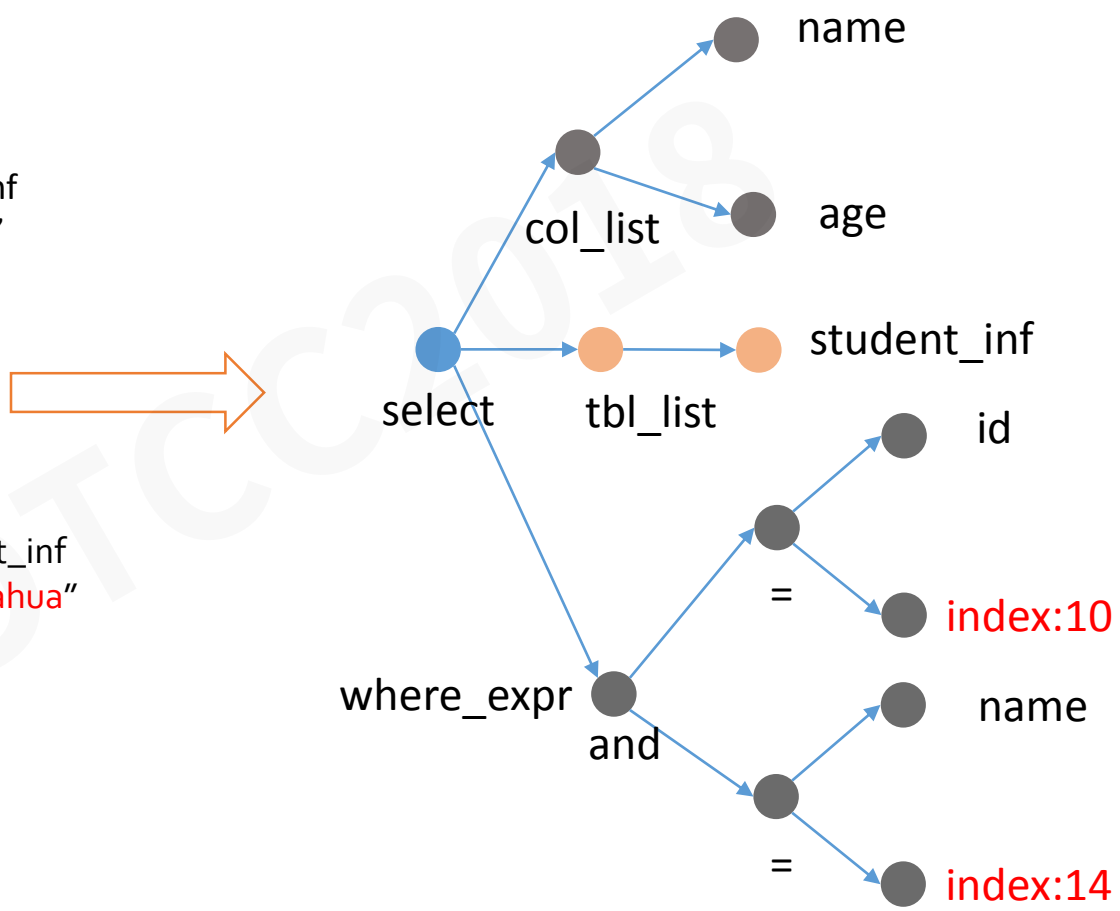
相似性解析优化

SQL解析优化 \ 语句示例

1

Select name, age from student_inf
where id =1 and name = ”dabao”
~

2

Select name, age FROM student_inf
where id =2 and name = “huahua”
~~~


~~~

SQL解析优化 \ 相似性分析

- 词法分析的结果：
  - 一组单词序列
  - 每个单词由类型和值组成
- 词法解析流程调整：
  - MySQL的SQL解析过程，词法和语法解析耦合在一起，即语法解析的移进规约动作触发词法解析，而不是先完成词法解析，然后开始语法解析
  - 修改为：先完成词法解析获取单词序列，然后进行语法解析
- 相似性规则：
  - 对比2个SQL的单词序列，单词一一对比满足下列2个条件即说明语句相似：
    - 单词类型相同
    - 如果类型非参数，则要求单词的值相等(忽略大小写)
  - 同样的我们根据上述2个约束，设计了一个相似性hash算法，用于提升相似性查找性能

SQL解析优化 \ 相似性复用

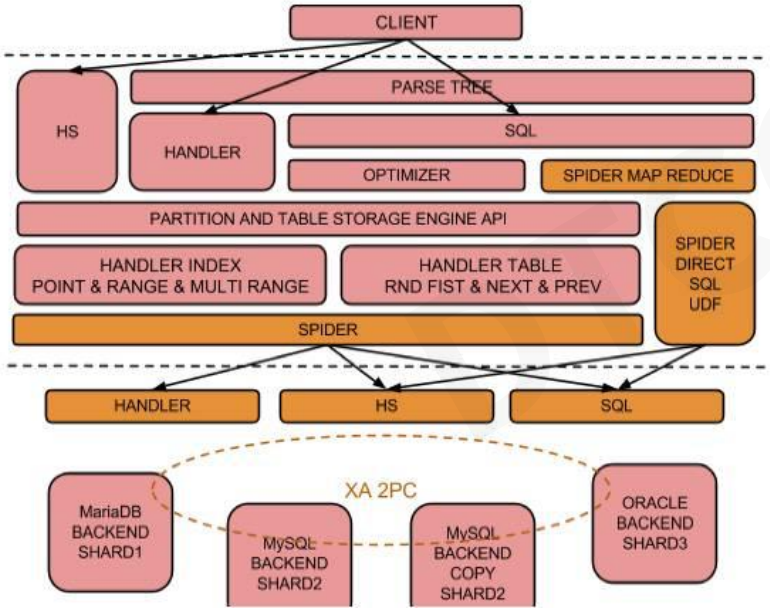
- LEX_STRING是词法分析和语法解析的最基础数据，增加2个字段：
  - index:其在词法序列中的位置
  - next_lex_string:需要合并的单词
- 新语句，通过相似语句的语法树、逻辑执行计划和当前词法序列，即可以进行相应计算操作，实现复用。

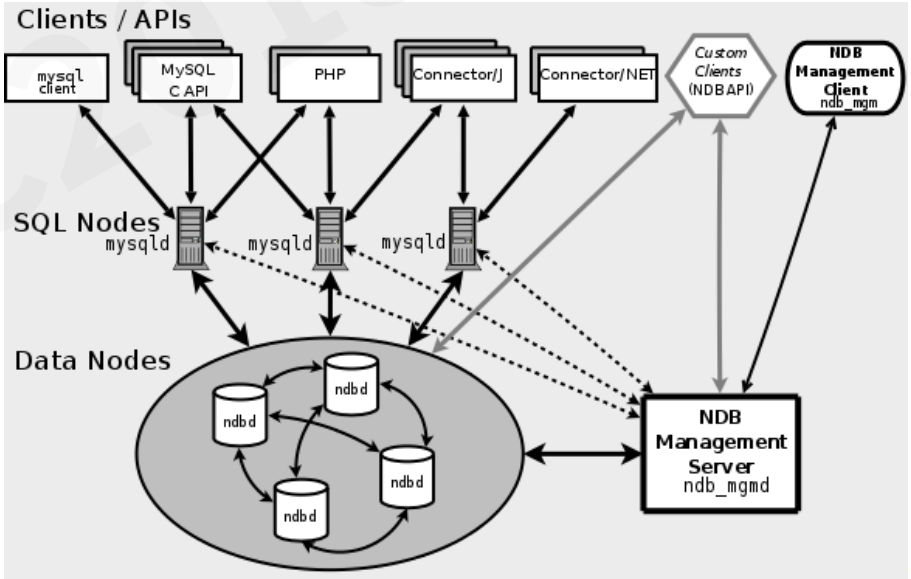
```
struct st_mysql_lex_string
{
  char *str;
  size_t length;

  int index; /* 在词法分析结果内，单词序列的位置 */
  struct st_mysql_lex_string *next_lex_string; /* 需要合并的单词 */
};
```

复杂查询

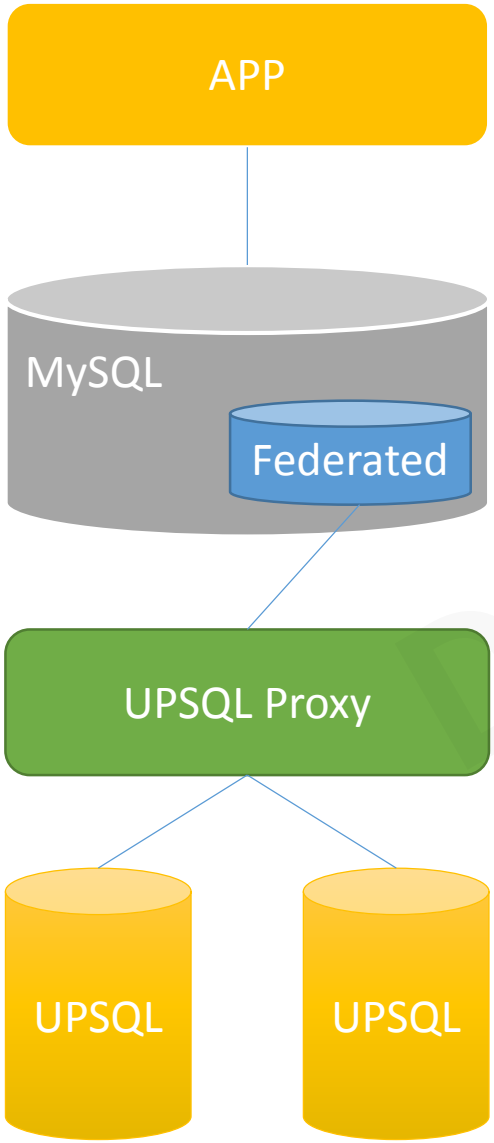
数据拆分下，如何解决复杂查询(例如典型的跨库Join)？

- ✓ 强化Proxy层构建支持复杂查询的执行计划
  - 代价高昂
- ✓ 使用分布式存储引擎：MariaDB Spider、 MySQL NDB
  - MariaDB Spider：同步调用较多、性能不佳
  - MySQL NDB：缺乏实践经验



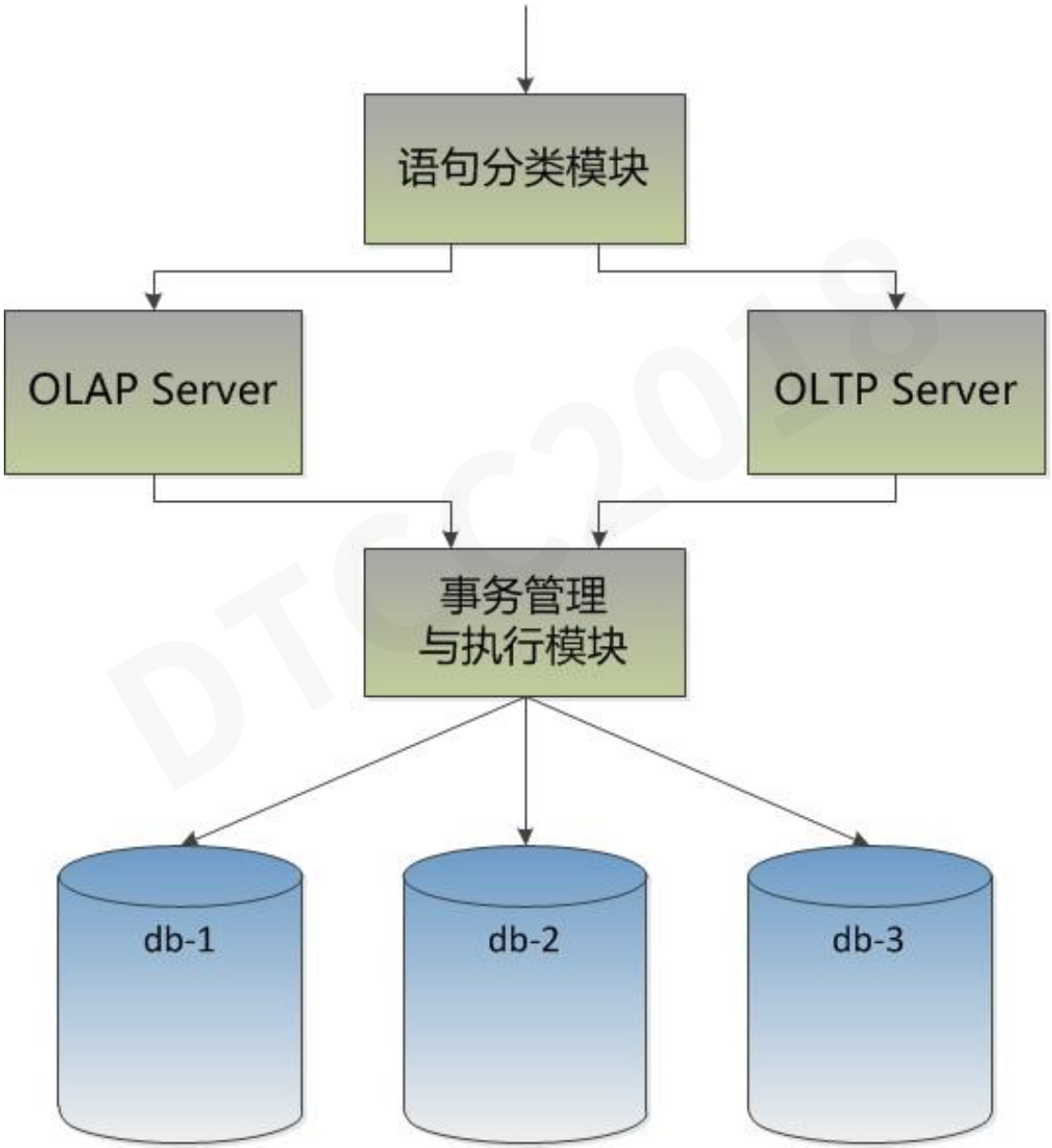


复杂查询 \ 设计思路(Federated+Proxy)

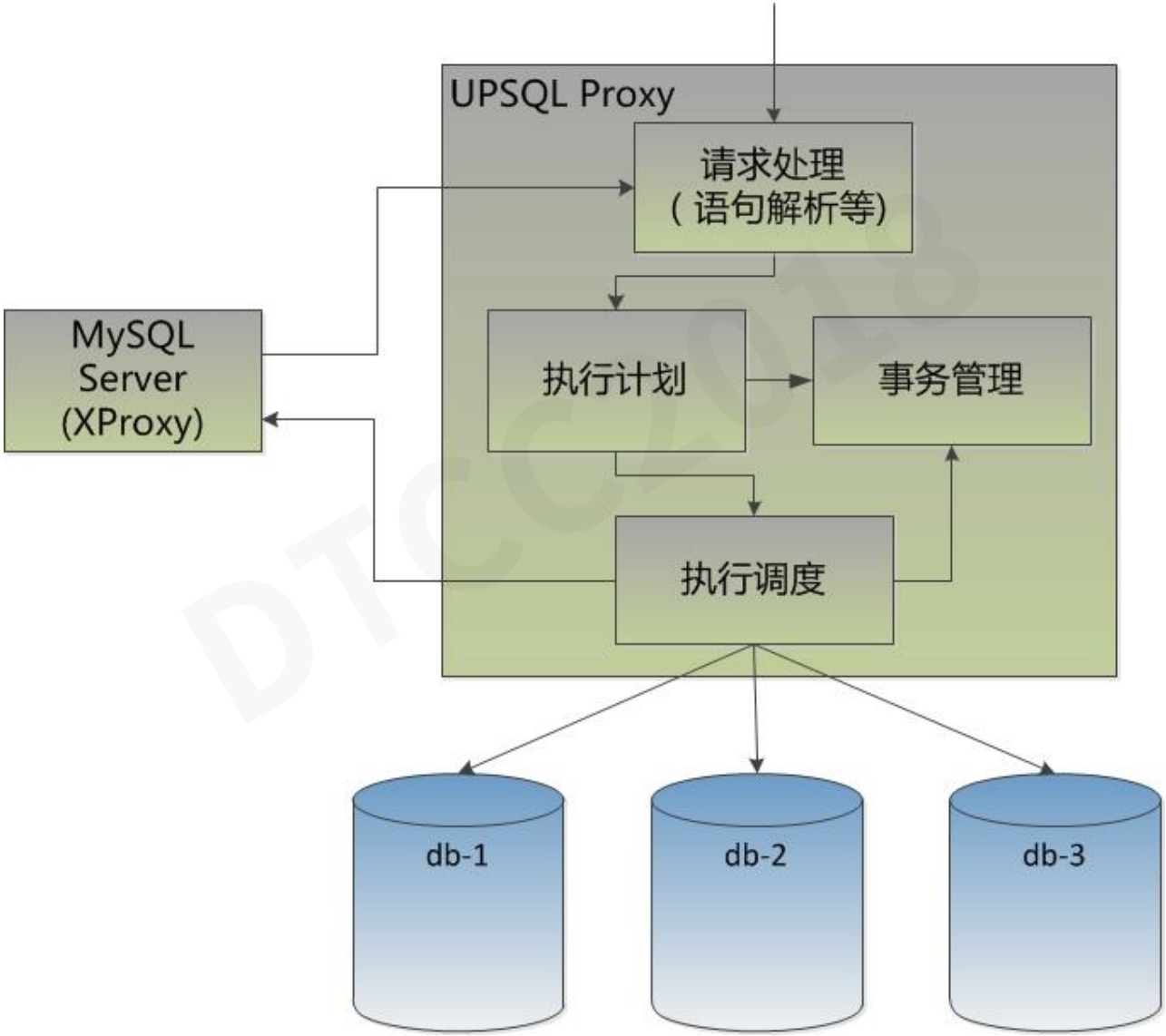


- 类比MariaDB Spider
- 核心思路：
  - 由Federated引擎外挂UPSQL Proxy
  - 由UPSQL Proxy实现数据拆分和事务管理
- 优点：
  - 实现逻辑简单
  - 功能可控
- 缺点：
  - 性能差

复杂查询 \ 设计思路(语句分类路由)

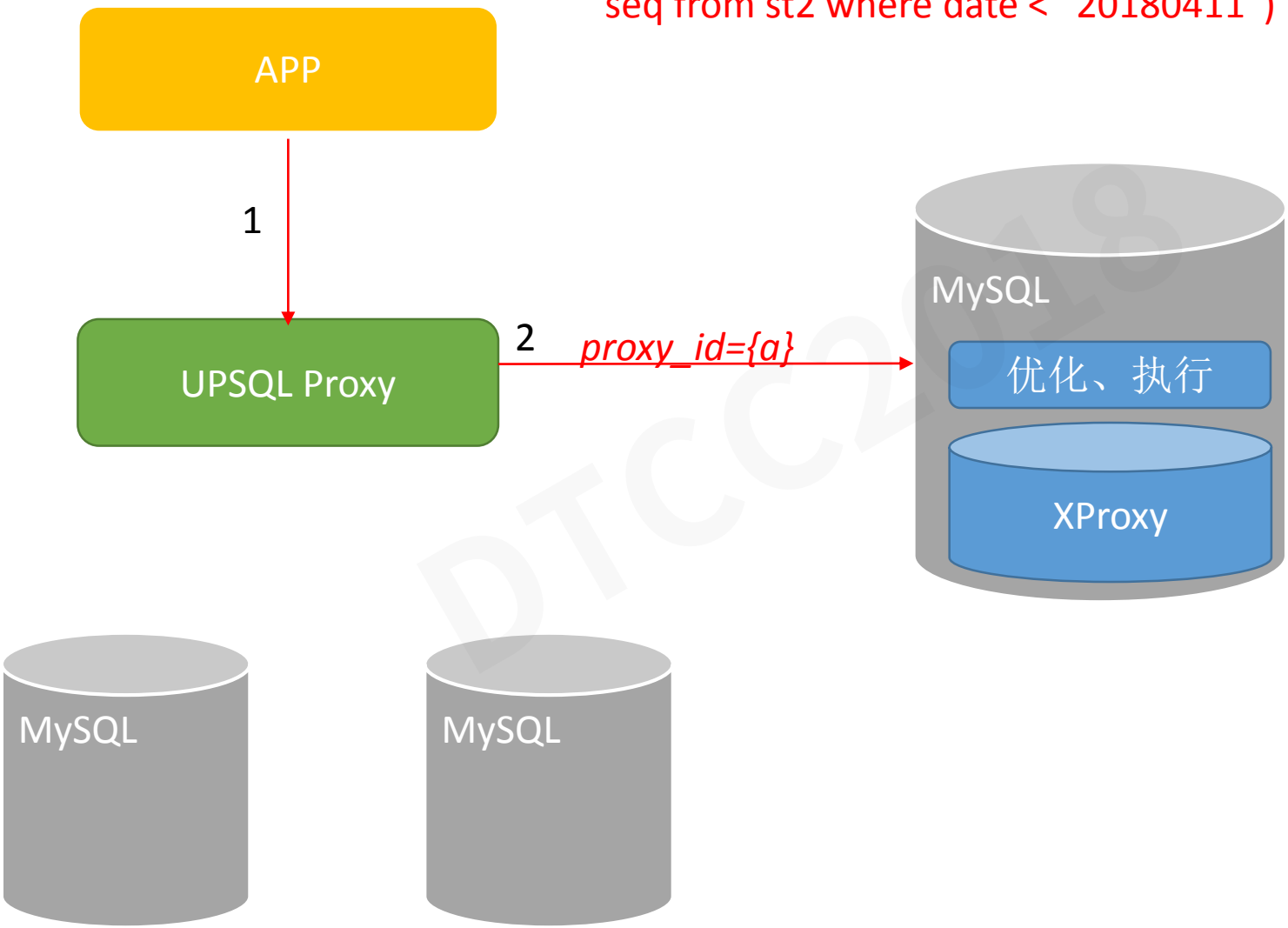


复杂查询 \ 设计思路(协处理器：XProxy)



复杂查询 \ 设计思路(执行)

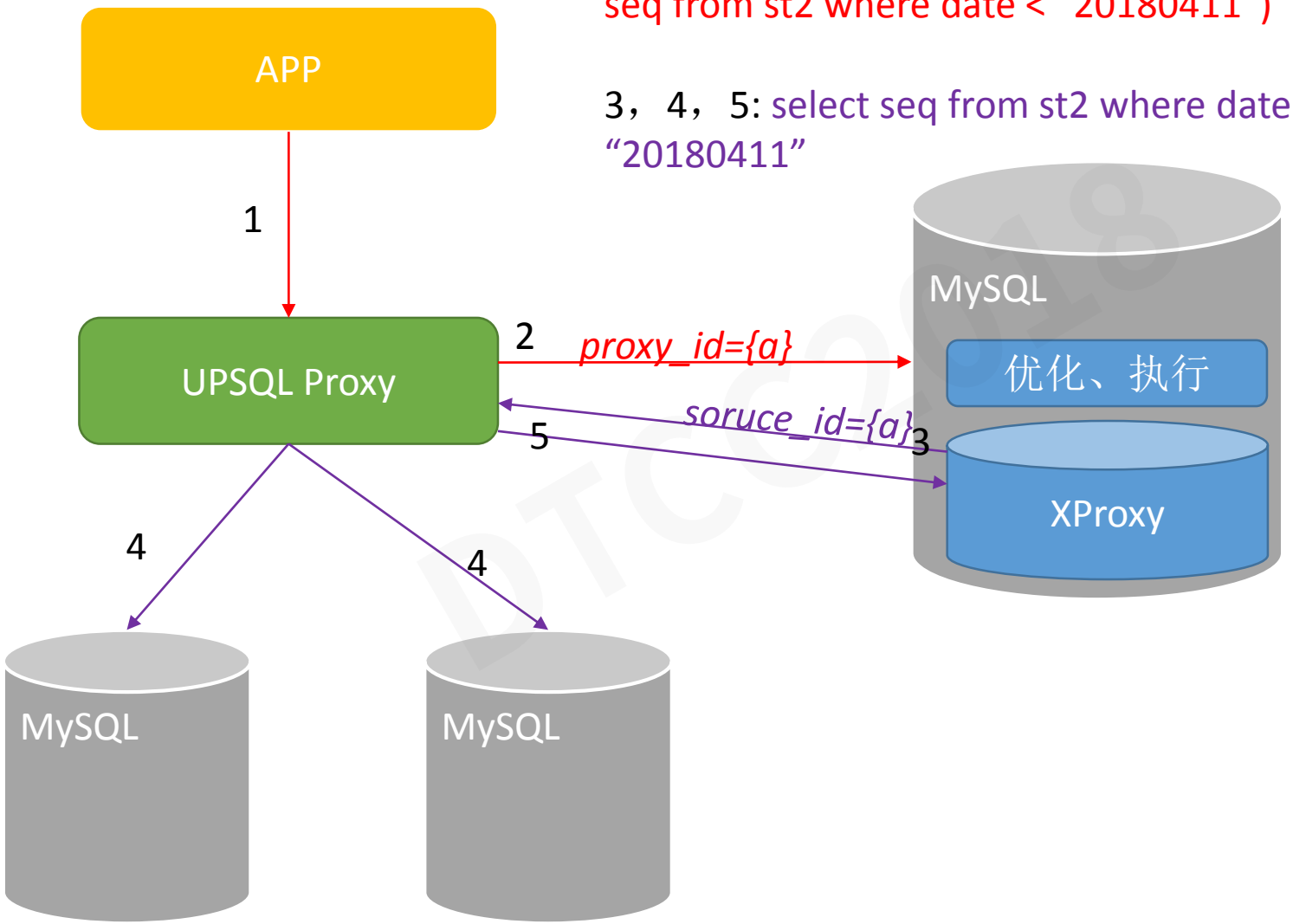
1，2 ：select * from st3 where seq in (select seq from st2 where date < “20180411”)



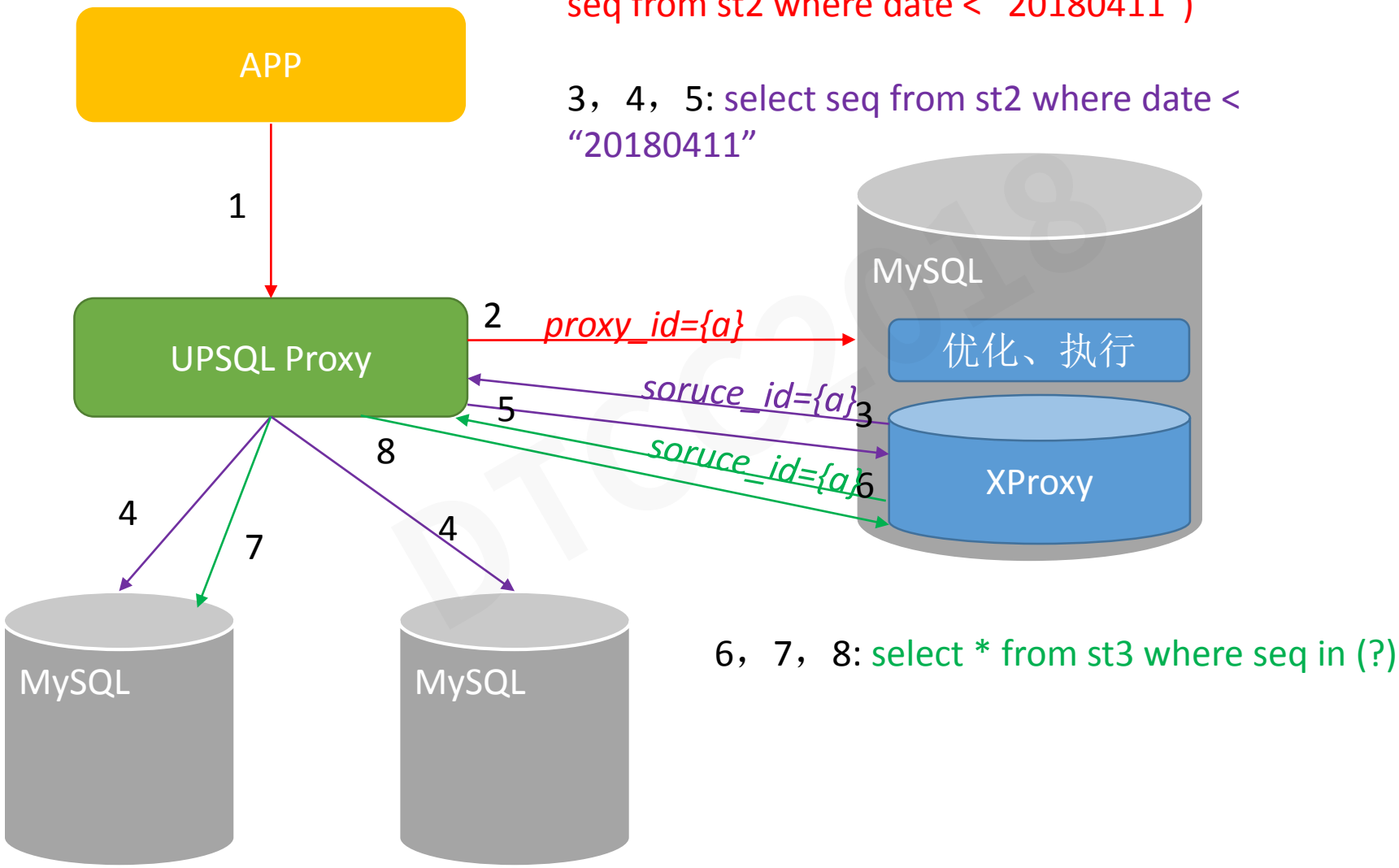
复杂查询 \ 设计思路(执行)

1，2 : select * from st3 where seq in (select seq from st2 where date < “20180411”)

3，4，5: select seq from st2 where date < “20180411”



复杂查询 \ 设计思路(执行)



1，2 : select * from st3 where seq in (select seq from st2 where date < “20180411”)

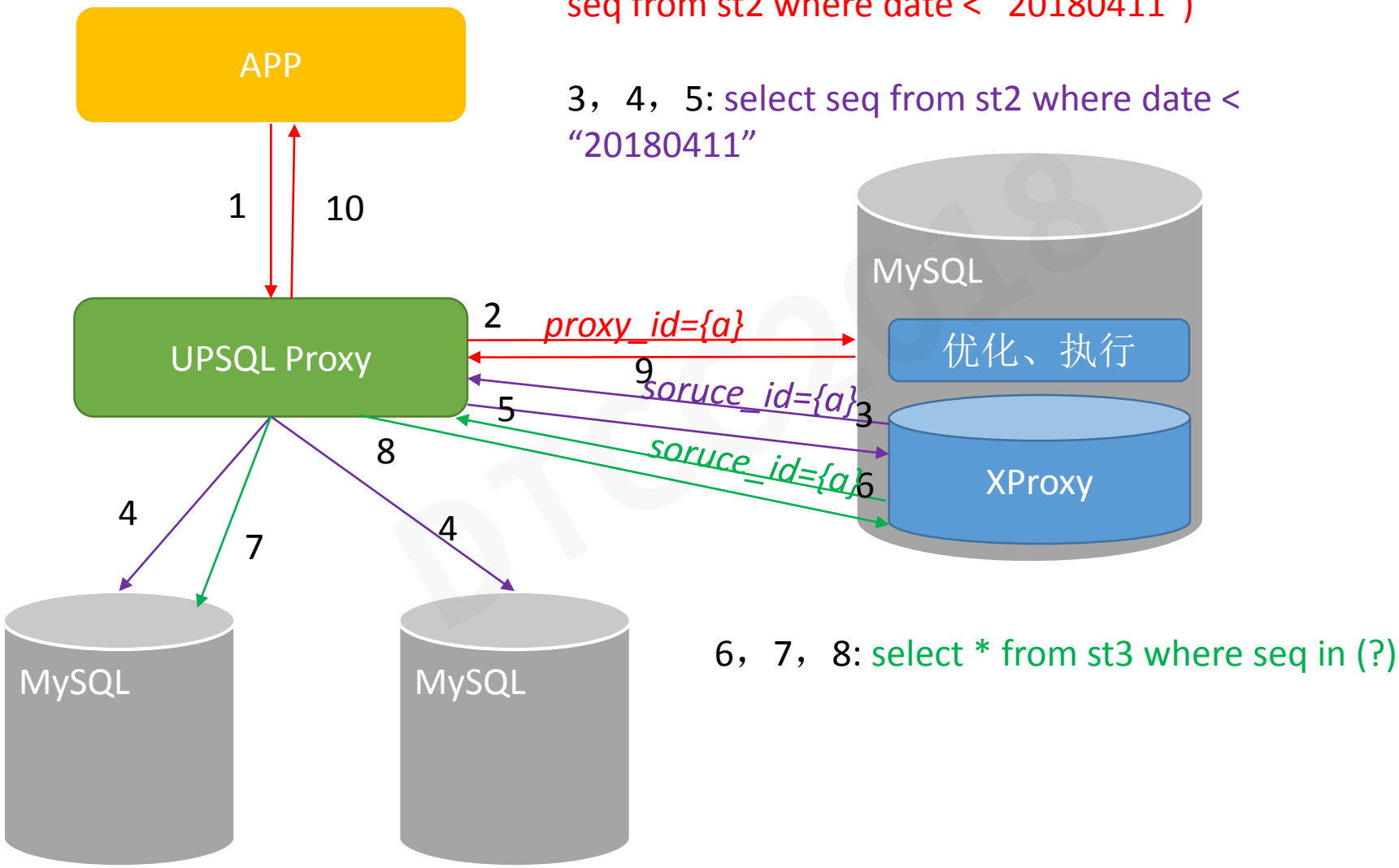
3，4，5: select seq from st2 where date < “20180411”

6，7，8: select * from st3 where seq in (?)

复杂查询 \ 设计思路(执行)

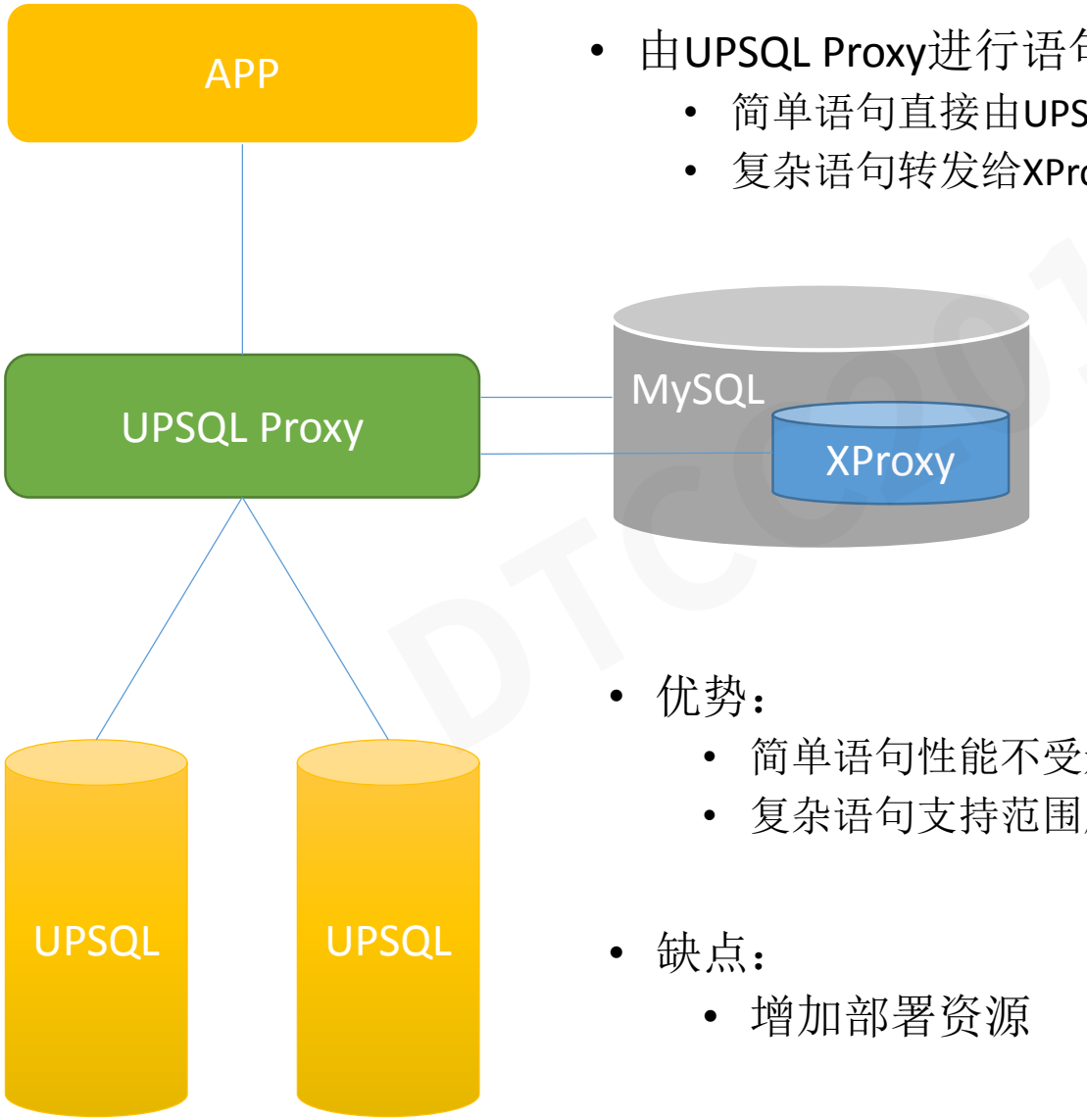
1，2，9，10: select * from st3 where seq in (select seq from st2 where date < “20180411”)

3，4，5: select seq from st2 where date < “20180411”



6，7，8: select * from st3 where seq in (?)

复杂查询 \ 伙伴部署

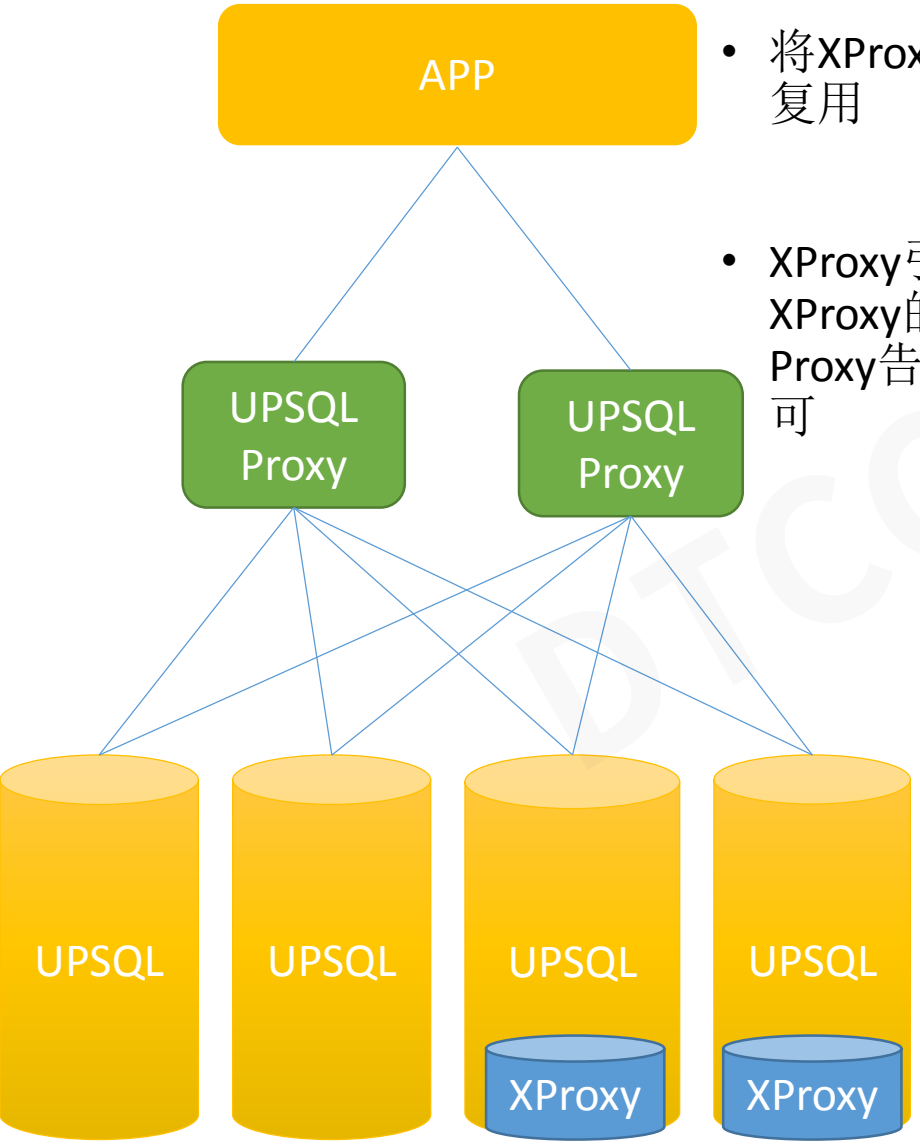


- 由UPSQL Proxy进行语句路由：
  - 简单语句直接由UPSQL Proxy处理
  - 复杂语句转发给XProxy引擎处理

- 优势：
  - 简单语句性能不受影响
  - 复杂语句支持范围广

- 缺点：
  - 增加部署资源

复杂查询 \ 集群部署



- 将XProxy引擎下沉到数据存储节点，实现资源复用
- XProxy引擎下沉，主要是改变了UPSQL Proxy与XProxy的一一对应关系，所以只需要UPSQL Proxy告知XProxy自身的物理地址和认证信息即可

复杂查询 \ 示例(事务)

```
mysql> begin;
Query OK, 0 rows affected (0.01 sec)

mysql> select count(*) from mtrtest.st2, mtrtest.st3 limit 2;
+----------+
| count(*) |
+----------+
|     3024 |
+----------+
1 row in set (0.05 sec)

mysql> insert into st2 (seq,date) values (10001, 1);
Query OK, 1 row affected (0.00 sec)

mysql> select count(*) from mtrtest.st2, mtrtest.st3 limit 2;
+----------+
| count(*) |
+----------+
|     3078 |
+----------+
1 row in set (0.06 sec)

mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql> select count(*) from mtrtest.st2, mtrtest.st3 limit 2;
+----------+
| count(*) |
+----------+
|     3024 |
+----------+
1 row in set (0.06 sec)

mysql>
```

复杂查询 \ XProxy DDL语法

~~mysql> create server test_connection~~
~~foreign data wrapper mysql~~
~~options (user 'shard', password 'shard', host '172.21.34.96' , port 4048);~~

```
mysql>  create table `st2` (
  `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='Federated' 'XProxy'
default charset=utf8 connection='test_connection';
```

复杂查询 \ 总结

- XProxy实现要点：
  - 使用STMT协议访问UPSQL Proxy
  - 会话关联(UPSQL Proxy与XProxy双向话关联)，事务管理器关联
  - 在线DDL

- 进一步优化(生产实践优先级低)：
  - ICP(Index Condition Pushdown)
  - ECP(Engine Condition Pushdown)

- 该方案在如下场景下性能极差，复杂查询下：
  - 不带过滤条件的分页 – 需要增加存储引擎接口
  - 统计运算 - 可以借鉴MariaDB Spider的MapReduce方案

Q&A

谢谢！

THANKS


HANA
Hadoop
SQL
MySQL
DB2
BigDate
Spark
SQL Server
NewSQL
Oracle
MPP
NoSQL
OLAP

讲师申请
联系电话（微信号）：18612470168

关注“ITPUB”更多
技术干货等你来拿~

与百度外卖、京东、魅族等先后合作系列分享活动

微学堂

让学习更简单

微学堂是以ChinaUnix、ITPUB所组建的微信群为载体，定期邀请嘉宾对热点话题、技术难题、新产品发布等进行移动端的在线直播活动。

截至目前，累计举办活动期数60+，参与人次40000+。

ITPUB学院

ITPUB学院是盛拓传媒IT168企业事业部（ITPUB）旗下
企业级在线学习咨询平台
历经18年技术社区平台发展
汇聚5000万技术用户
紧随企业一线IT技术需求
打造全方式技术培训与技术咨询服务
提供包括企业应用方案培训咨询（包括企业内训）
个人实战技能培训（包括认证培训）
在内的全方位IT技术培训咨询服务

ITPUB学院讲师均来自于企业
一些工程师、架构师、技术经理和CTO
大会演讲专家1800+
社区版主和博客专家500+

培训特色

无限次免费播放
随时随地在线观看
碎片化时间集中学习
聚焦知识点详细解读
讲师在线答疑
强大的技术人脉圈

八大课程体系

基础架构设计与建设
大数据平台
应用架构设计与开发
系统运维与数据库
传统企业数字化转型
人工智能
区块链
移动开发与SEO

联系我们

联系人：黄老师
电　话：010-59127187
邮　箱：edu@itpub.net
网　址：edu.itpub.net
培训微信号：18500940168