SuperSQL：数据湖时代的高性能SQL引擎

腾讯大数据-张韶全

1. **SuperSQL的背景定位**
2. SuperSQL的整体架构
3. SuperSQL的技术细节
4. SuperSQL的未来计划

项目背景：跨数据源分析

异构数据源：

数据存放在**异构**数据源，但需要**全部**数据来挖掘数据背后的信息

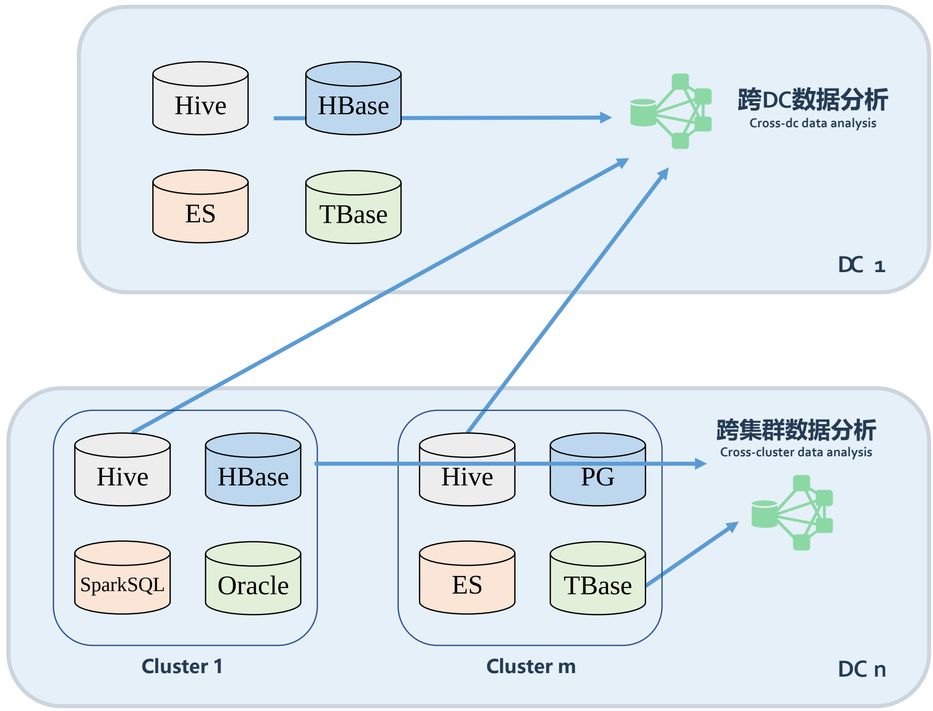
不同版本数据源：

数据存放在**不同版本**的数据源，但版本之间**不兼容（API、数据格式等）**

项目背景：跨DC、跨集群分析

数据存放在**不同集群**或者**不同DC**

传统的数据搬迁方式浪费存储和带宽资源



项目背景：计算引擎各有所长

擅长

Operational DDL

例如，库表增删改、权限等元数据操作

毫秒

擅长

Batch SQL

例如，ETL、数据分析、报表

分钟到小时

擅长

Interactive SQL

例如，数据采样、交互式查询

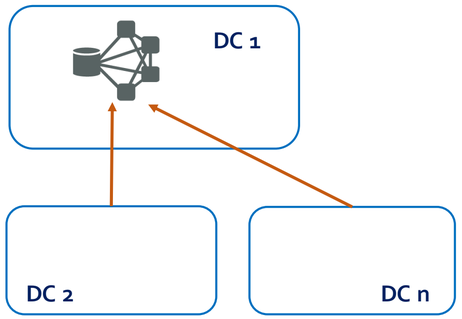
秒到分钟

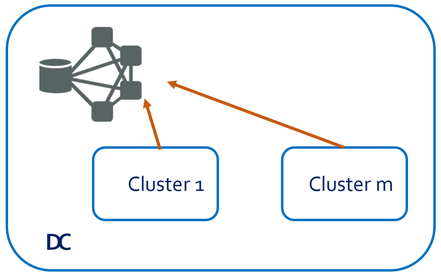
用户面对不同的计算引擎选择，计算对用户**不透明**

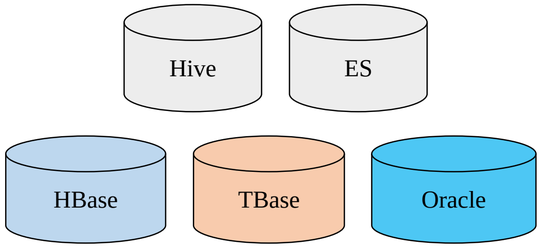
同一优化规则，开发适配不同的计算引擎，**工作重复**

SuperSQL：跨DC，跨数据源，跨计算引擎的大数据SQL引擎

- ✓ 解决数据孤岛，降低数据使用壁垒
- ✓ 优化资源使用，提升数据使用效率
- ✓ 计算最优选择，提升数据使用效率

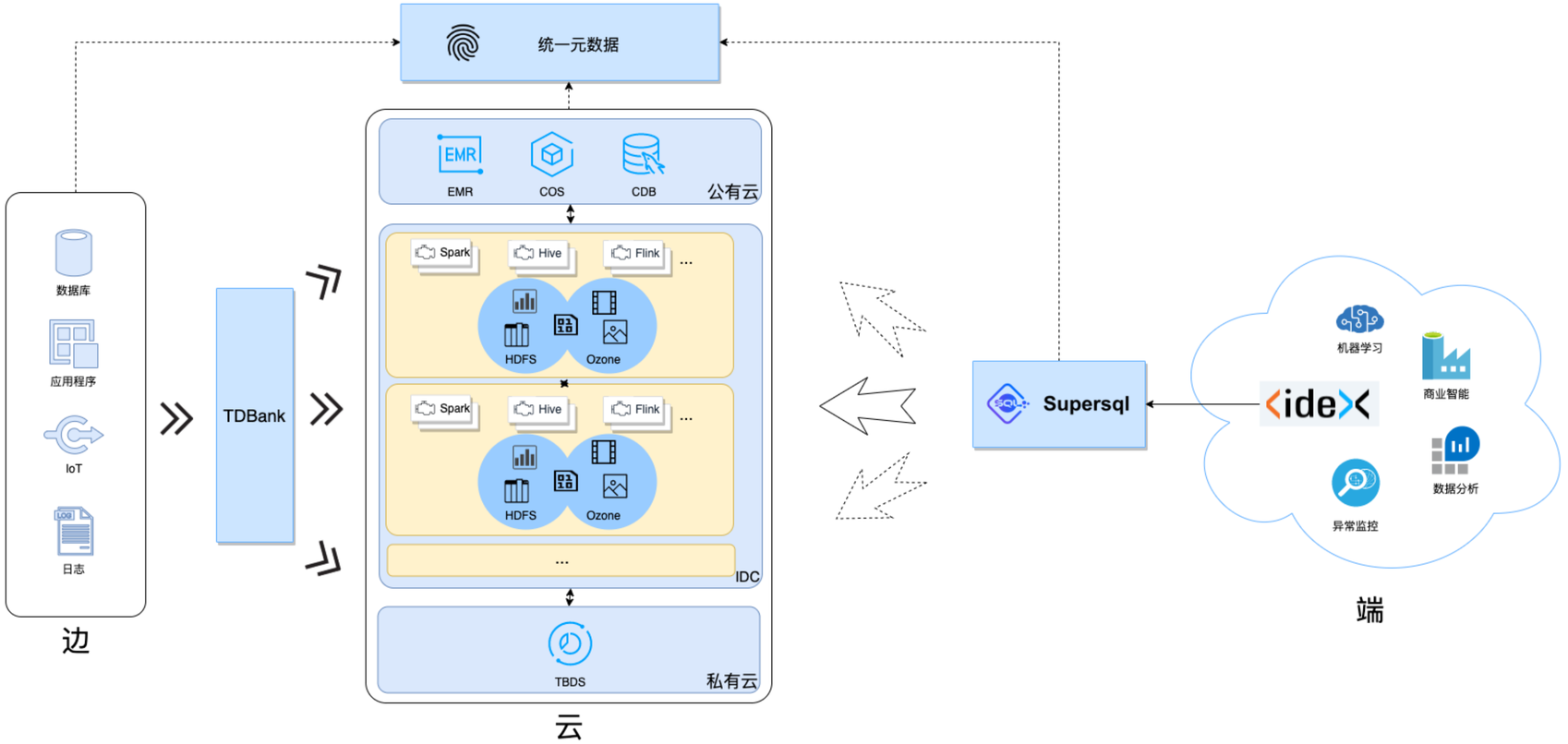






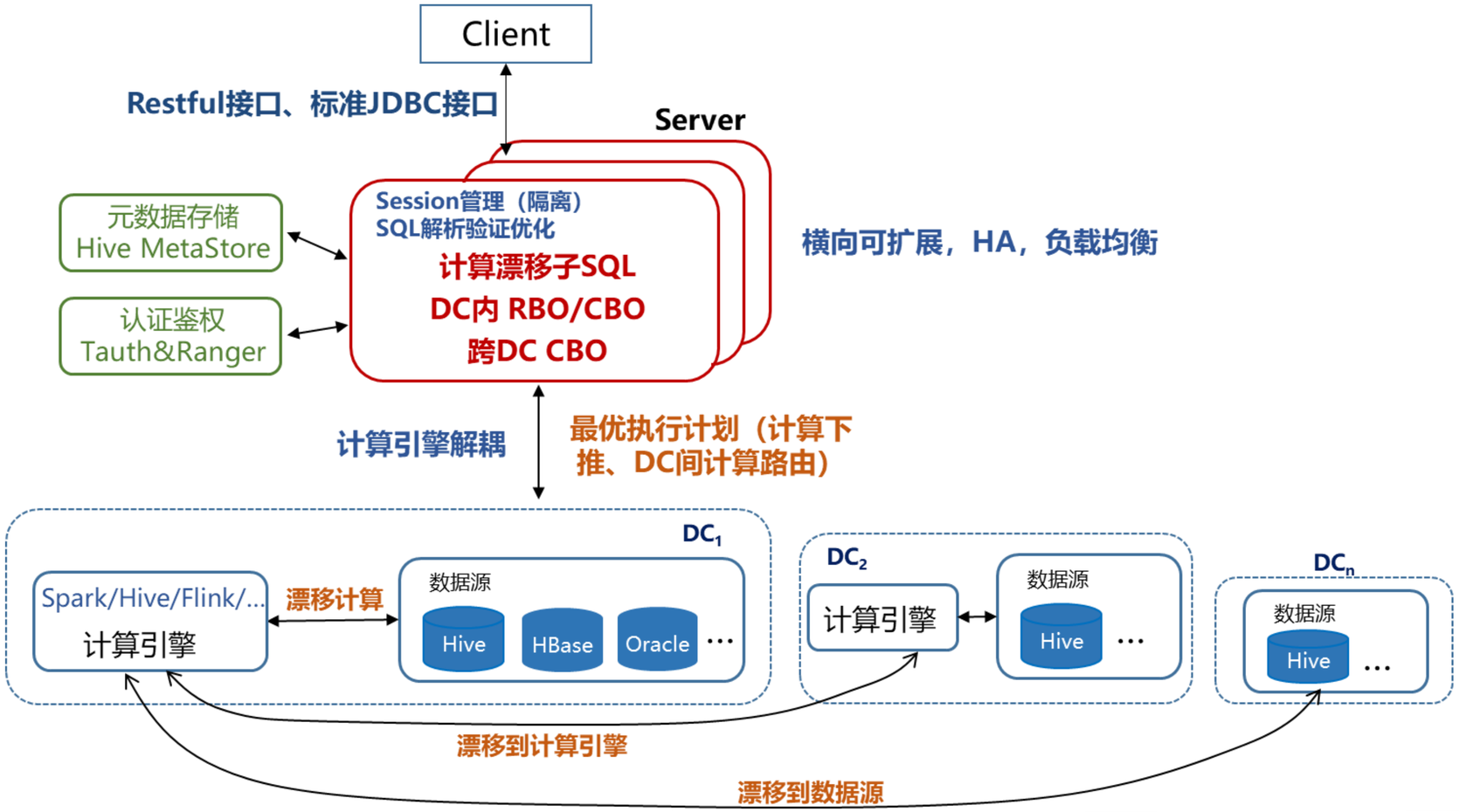
SuperSQL在腾讯大数据中的定位

连接端与存储的
统一的SQL引擎



1. SuperSQL的背景定位
2. **SuperSQL的整体架构**
3. SuperSQL的技术细节
4. SuperSQL的未来计划

整体架构


Client
Restful接口、标准JDBC接口
Server
元数据存储
Hive MetaStore
认证鉴权
Tauth&Ranger
Session管理（隔离）
SQL解析验证优化
计算漂移子SQL
DC内 RBO/CBO
跨DC CBO
横向可扩展，HA，负载均衡
计算引擎解耦
最优执行计划（计算下推、DC间计算路由）
DC1
Spark/Hive/Flink/...
计算引擎
漂移计算
数据源
Hive
HBase
Oracle
...
DC2
计算引擎
数据源
Hive
...
DCn
数据源
Hive
...
漂移到计算引擎
漂移到数据源

1. SuperSQL的背景定位
2. SuperSQL的整体架构
3. SuperSQL的技术细节
4. SuperSQL的未来计划

数据源元数据管理(技术1/5)

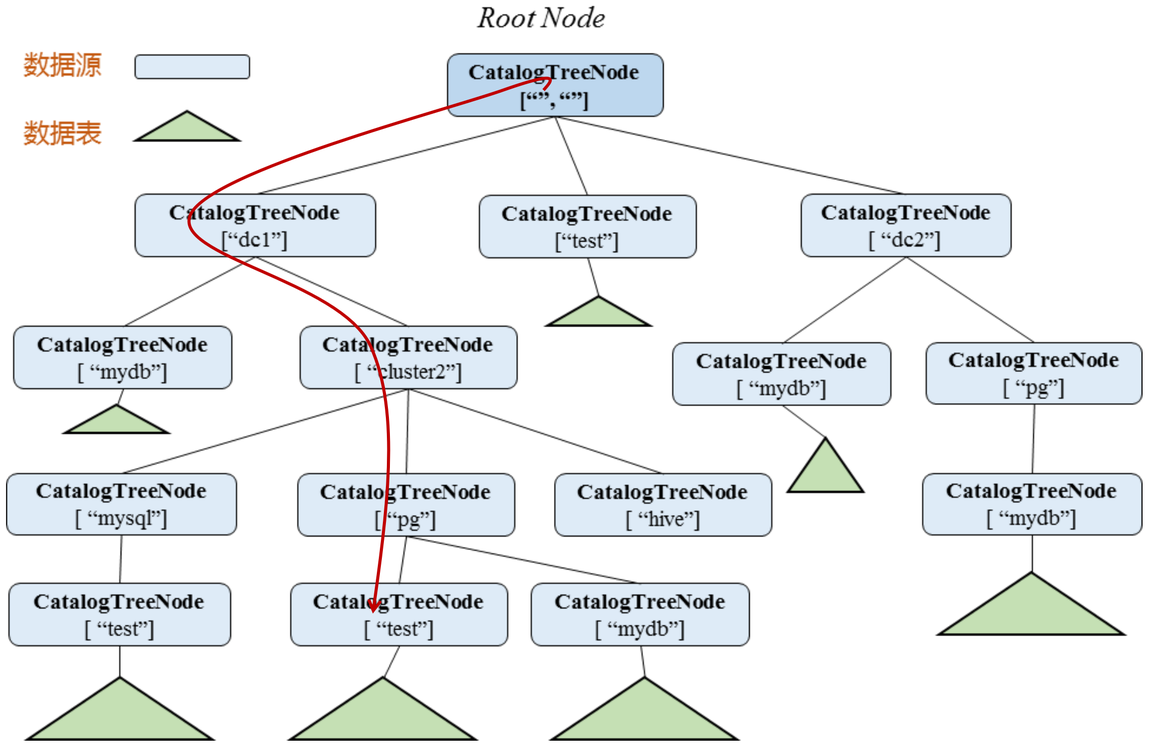
元数据模型设计

- ✓ **树结构**：每个节点对应的子树形成了一个子命名空间
- ✓ **树节点：**包含父节点、子节点列表、节点路径、名称，以及配置信息

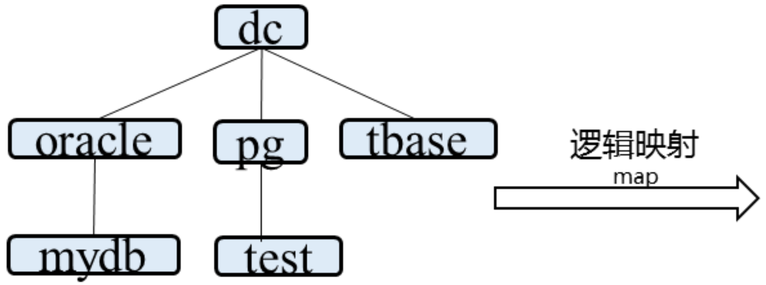
```
select *
from test.mytable
```

⇩

```
select *
from dc1.cluster2.pg.test.mytable
```



数据源元数据存储



path(String)	config(Map<String, String>)
dc.oracle.mydb	config_mydb
dc.oracle	config_oracle
dc	config_dc
dc.pg.test	config_test
dc.pg	config_pg
dc.tbase	config_tbase

(1) One path is one Table
(2) Config is stored as Table Parameters

Hive MetaStore Service / OMS

RDBMS

PG/MySQL

元数据持久化

- ✓ **逻辑映射**：节点的路径映射到该节点的所有配置信息
- ✓ **物理存储：**多种存储方式，包括基于内存(重启后丢失)、基于Json文件(只读)和基于Hive MetaStore (增删查改)的实现，默认采用Hive MetaStore

SQL算子下推概述(技术2/5)

问题1. 如何减少数据的拉取?

将数据源相关的计算下推到数据源进行计算，充分利用数据源计算能力的同时，只拉取**必需**的数据

问题2. 如何判断哪些计算下推?

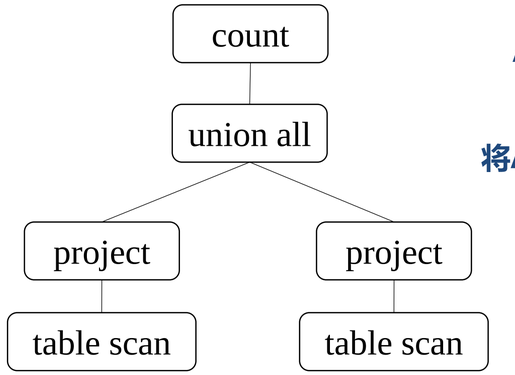
利用**RBO**，通过规则变换计划树，将有效的算子下推

问题3. 如何实现计算下推?

将需要下推的计算翻译为**下推子SQL**，在计算引擎中创建临时视图映射

计算下推规则

举例：
```
select count(*) from
    ( select col1, col2 from mysql.table
      union all
      select col1, col2 from pg.table )
```



规则应用前

Aggregation push down规则

匹配Aggregation-Union类型

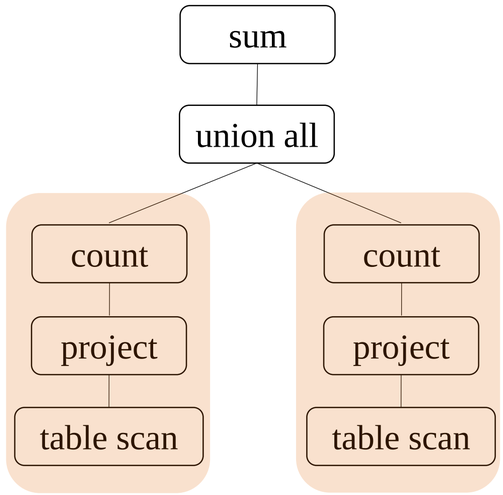
将Aggregation尽可能下推到数据源

保证语义的正确

1. count变换为总sum,子count

2, avg变换为sum/count

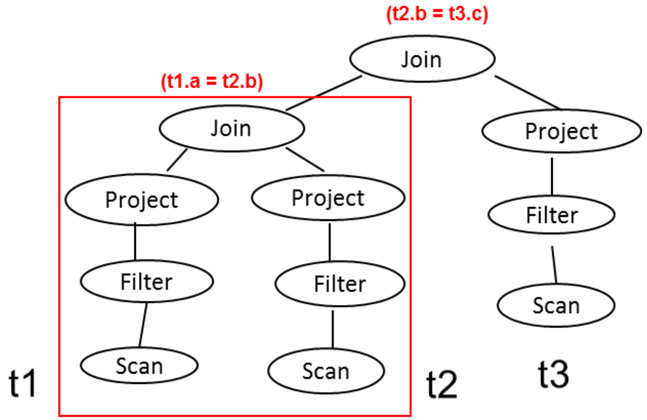
3. distinct变换为group by



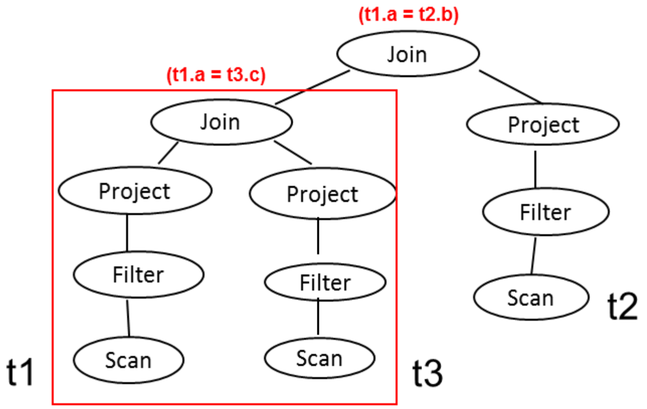
规则应用后

将count下推到数据源，可以大大减少数据的拉取

不仅仅RBO:JOIN算子示例



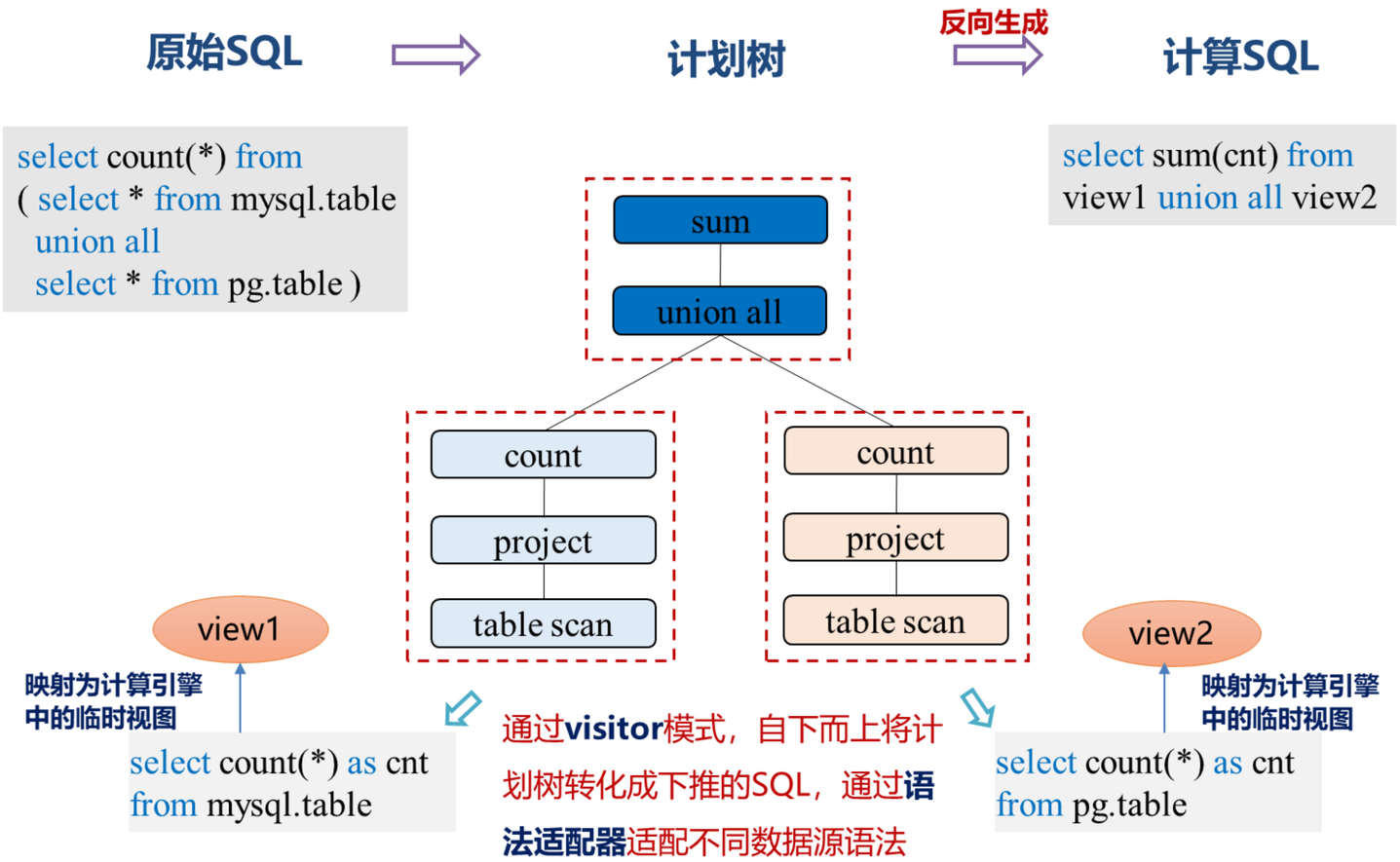
物理计划树变换
physical plan transform



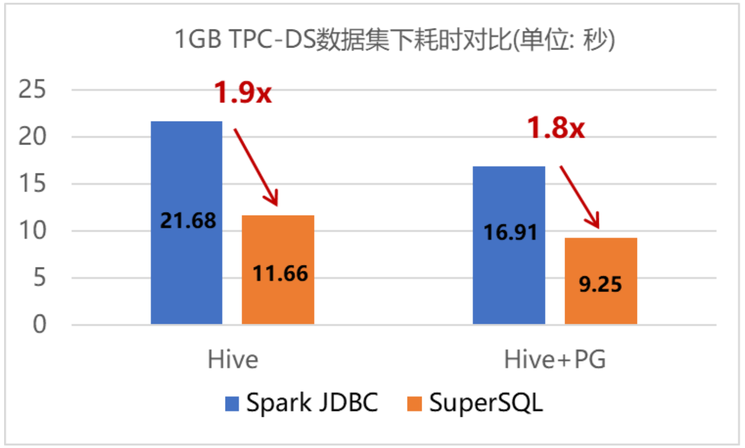
t1和t2来自不同数据源，无法下推
假设Join的执行次序一定是从左到右

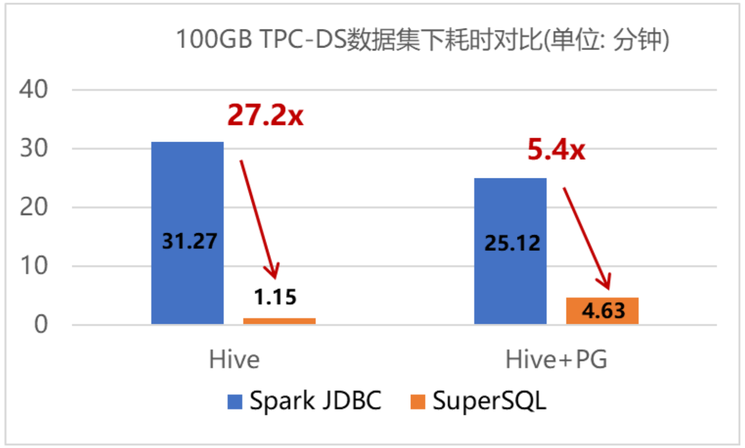
t1和t3来自同一数据源
并且t1和t3 join的结果没有**膨胀**(通过CBO判断)

下推计算架构实现


原始SQL
计划树
反向生成
计算SQL
select count(*) from
(select * from mysql.table
union all
select * from pg.table)
select sum(cnt) from
view1 union all view2
sum
union all
count
project
table scan
count
project
table scan
view1
view2
映射为计算引擎
中的临时视图
映射为计算引擎
中的临时视图
select count(*) as cnt
from mysql.table
通过visitor模式，自下而上将计
划树转化成下推的SQL，通过语
法适配器适配不同数据源语法
select count(*) as cnt
from pg.table

下推计算性能表现





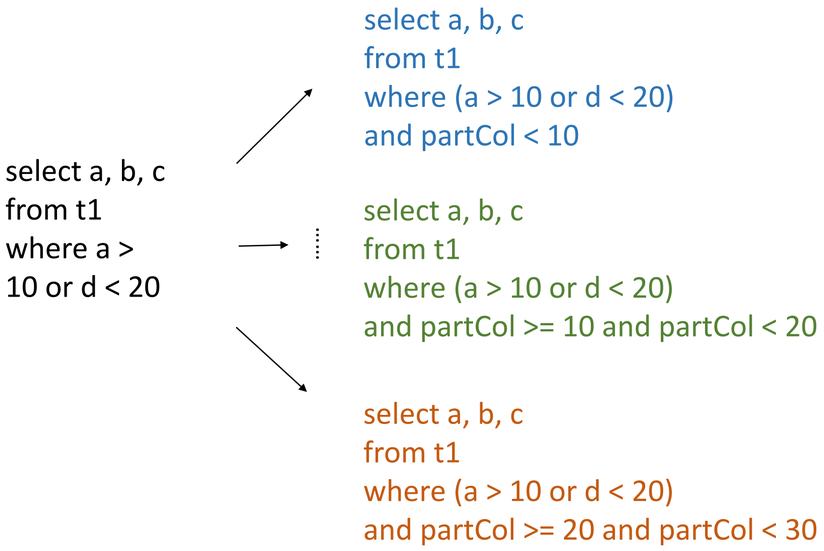
✓ 下推计算性能优势明显，最多**27倍**的提升

✓ 数据量越大，下推计算性能优势越明显

下推计算并发优化(技术3/5)

动机：当下推某一数据源的SQL返回数据量很大时（如千万条记录），单个JDBC链接（线程）获取结果耗时长

适用场景：基于统计信息，发现数据源表有合适的分区（如Hive）或Unique索引（如PG）

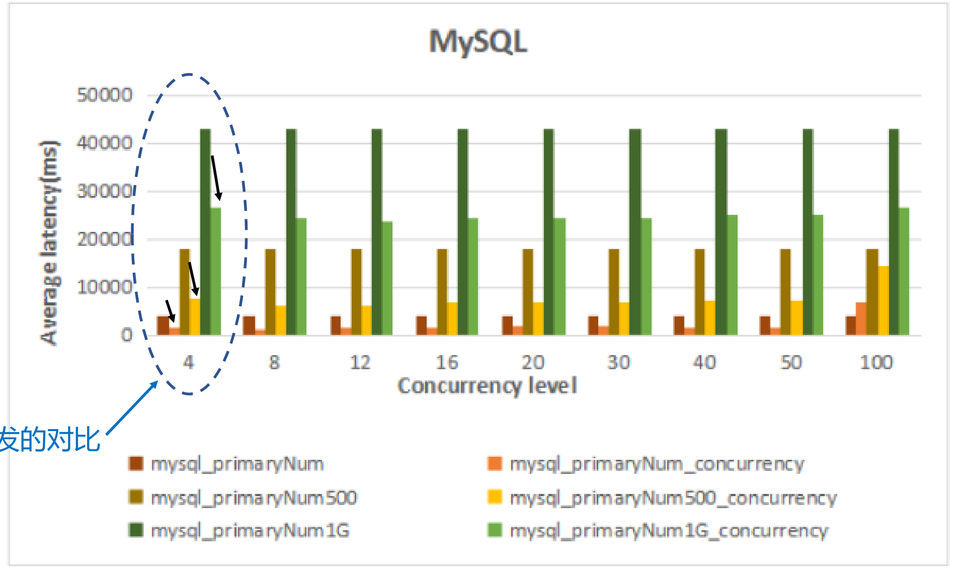


原理：

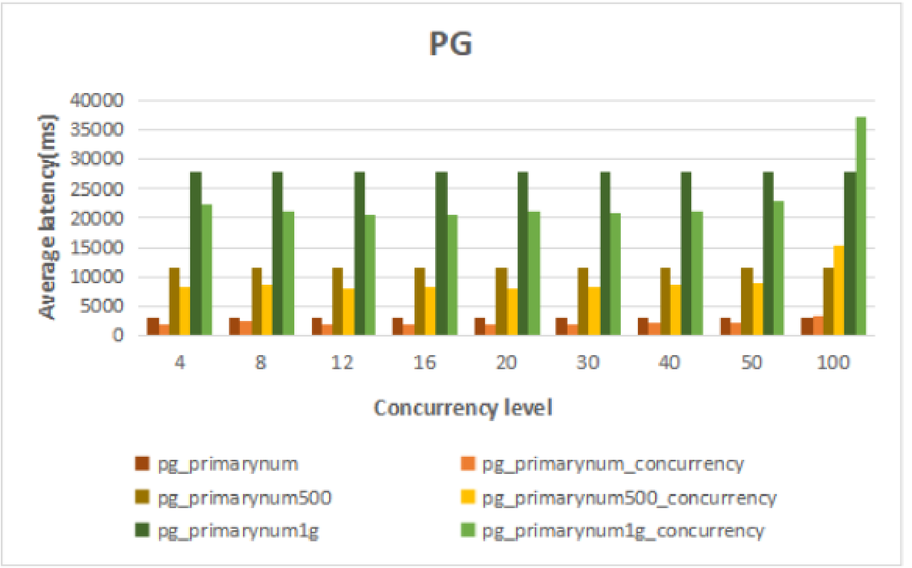
1. 根据分区、索引或其它列统计信息，为下推数据源表挑选一个“最好的切分列”，同时基于该列的最大最小值生成N个disjoint的分区条件（N可以固定也可以动态根据数据分布确定）
2. 将分区条件拼接到原始下推SQL的Where条件中，生成N个对应的子查询
3. 并发多线程执行N个子查询，合并结果

局限：语义复杂时，无法确保拆分子查询结果合并的正确性

下推并发性能提升



相同数据量下，并发执行与非并发的对比



- ✓ 并发下推计算，最多**3倍**的提升
- ✓ 并发度=8~16时性能最佳

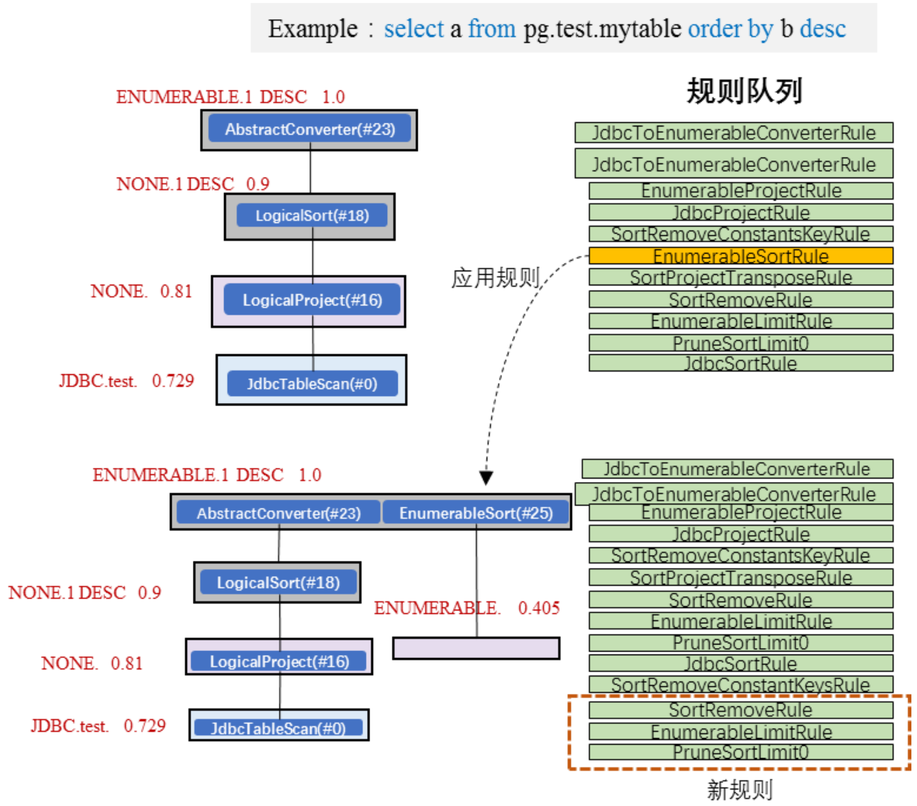
CBO：现网框架问题（技术4/5）

- **Volcano Planner**：采用代价优化思想的执行框架，基于“最优成本假设”，动态规划算法，自底向上或者自顶向下逐步变换执行计划
- **规则优先队列**：新的执行计划节点“触发”对应可用的规则，并将这些规则添加到队列中

现有框架问题：

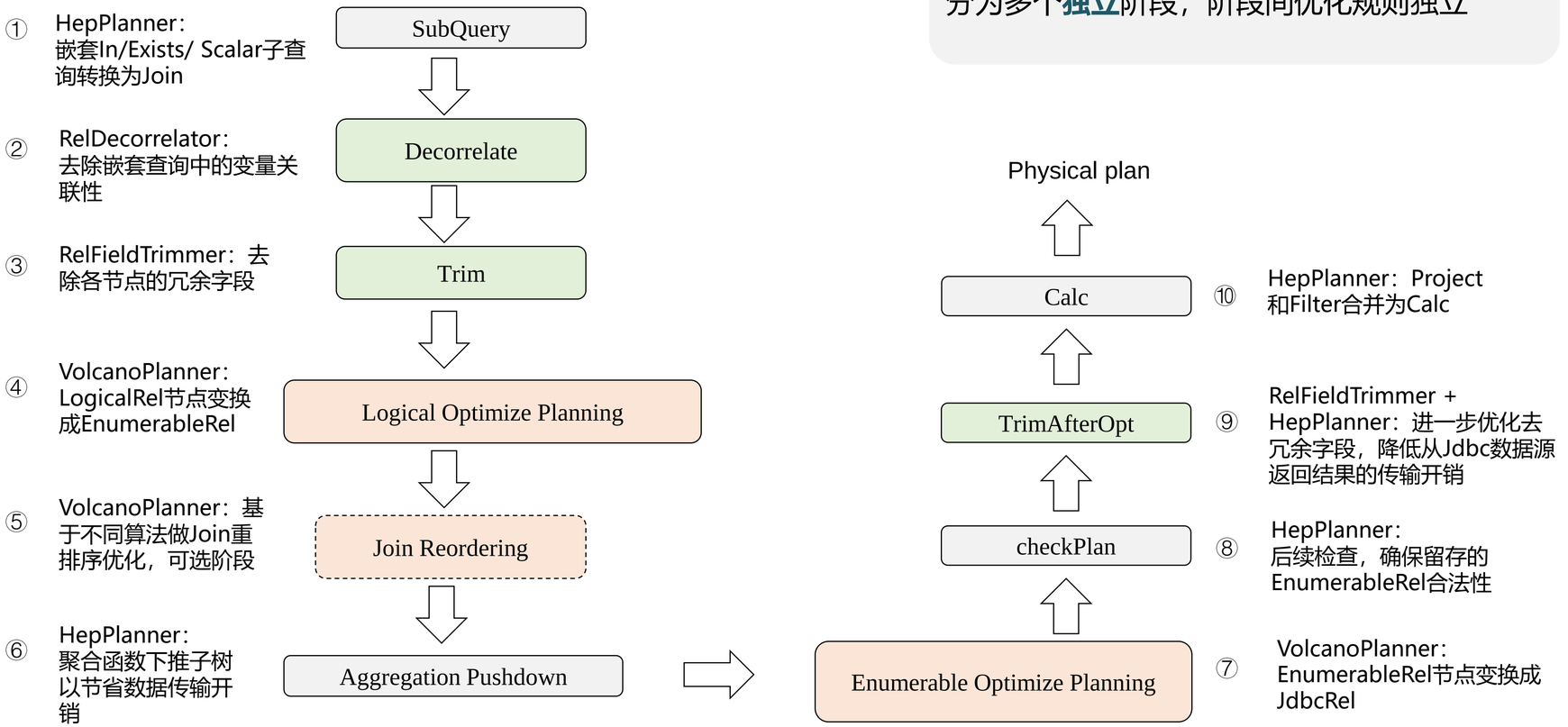
单一队列，规则重复甚至冲突，导致CBO耗时高

例如，对10表Join，执行**250157**条规则，耗时**169.574 s**

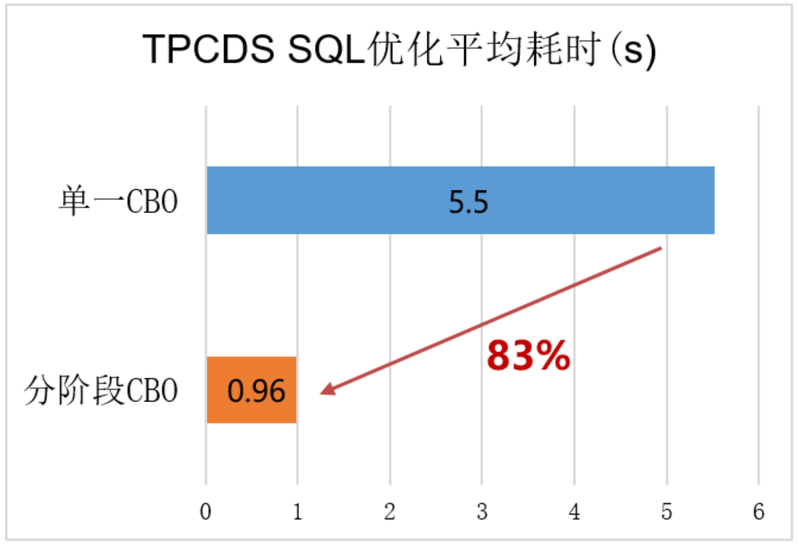


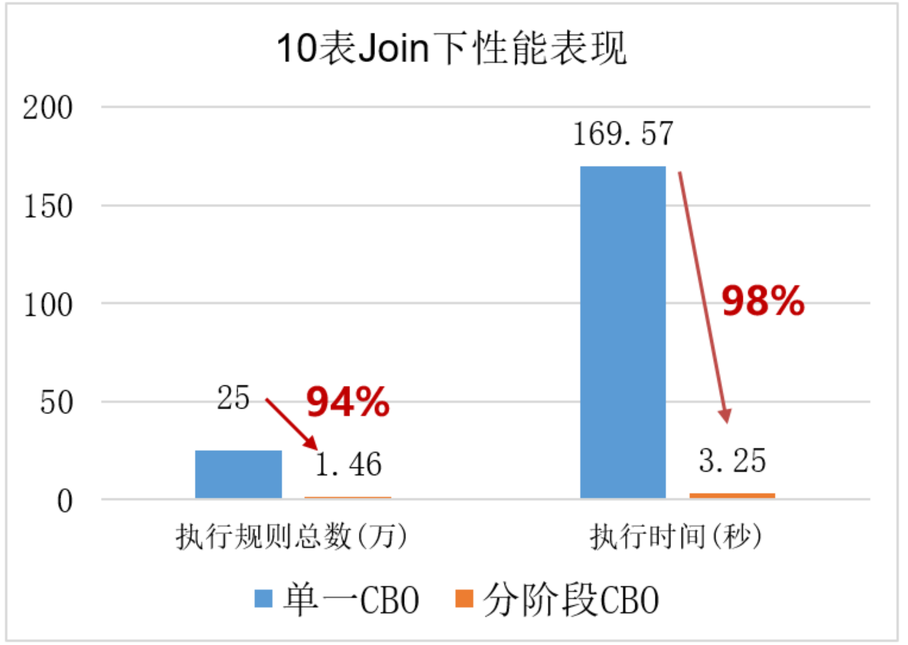
分阶段CBO

方案主要思想：按照规则作用对象和功能，拆分为多个**独立**阶段，阶段间优化规则独立



分阶段CBO效果

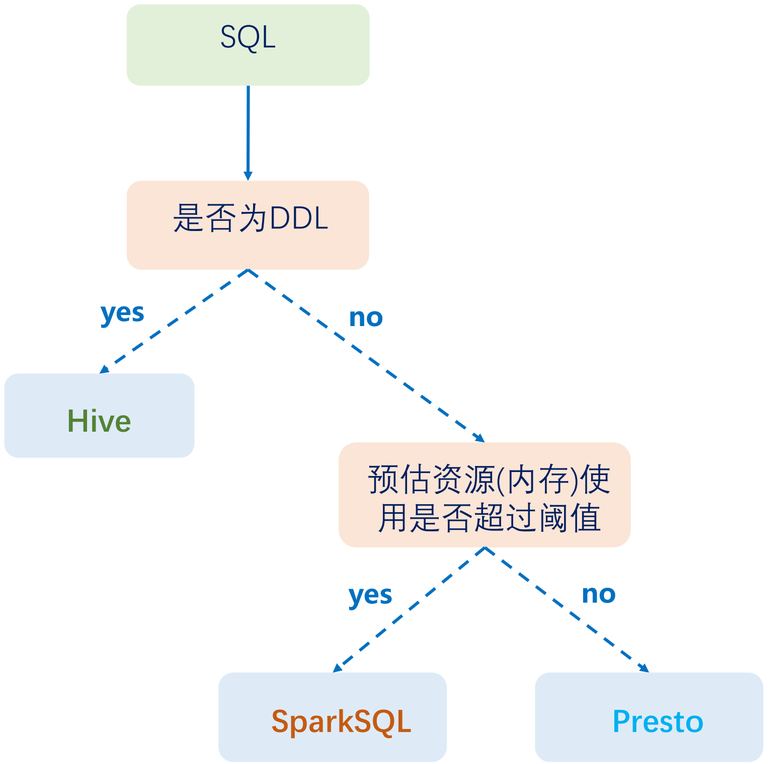




- ✓ CBO平均耗时减少**83%**
- ✓ 复杂SQL CBO耗时减少**98%**

计算引擎选择(技术5/5)

以访问Hive库表数据为例，通过规则决定合适的计算引擎



如何预估资源使用

1. 预估算子处理的数据量

   1. 没有统计信息
   递归计算子算子处理的数据量
   父算子处理的数据量 = 子算子之和
   数据膨胀的的算有统计子乘以膨胀系数，比如distinct
   2. 有统计信息
   利用CBO预估数据量

2. 预估SQL使用的资源

   并发度确定的前提下，确定每个并发task处理的数据量，以此确定每个task的内存使用

1. SuperSQL的背景定位
2. SuperSQL的整体架构
3. SuperSQL的技术细节
4. SuperSQL的未来计划

Future Work

- ✓ **数据源**：实现针对HBase，ES数据源的新API Connector，显著提升数据拉取稳定性和性能
- ✓ **数据Stats**：基于历史查询负载的智能采集，提升统计信息采集效率
- ✓ **安全计算**：数据安全计算框架，解决跨多方数据联合计算的安全问题
- ✓ **智能计算**：通过学习的方式解决计算选择的问题

THANKS