
OpenPie
Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

OpenPie

Why We Need CloudDB ?

Why we need PieCloudDB

OpenPie

NoSQL 和数据湖已经不再是数据分析的主要平台

- 在世界范围内的统计信息显示，Nosql和数据湖已经不在数据分析领域占有主要市场
- Nosql和数据湖缺少很多支持数据分析的重要特性
  - 缺少在高并发场景下的隔离性和一致性
  - 和现有的BI工具很难集成
- 关系型数据库已经重新成为数据分析的主要平台

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

Why We Need PieCloudDB

OpenPie

NoSQL和数据湖很难胜任数据分析的工作场景

- Nosql本身对于高级分析支持差
  - 图形，地理信息
- Nosql对于复杂查询的支持差

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

Why We Need PieCloudDB

OpenPie

NoSQL和数据湖为基础的基础设施需要的分析工具不容易集成和部署

- 使用数据湖为基础进行数据分析需要多个组件进行集成部署，多个组件的配合需要大量的开发工作
- 许多缺乏 ANSI SQL 支持，需要专门的技术技能
- 专用引擎/工具（例如图形数据库）通常难以与记录系统集成，限制了分析和创新的操作化

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

Why We Need PieCloudDB

OpenPie

以关系型数据库为基础的数据仓库很难适应云环境

- 公有云无限的计算池可以提供理想的弹性计算资源
- 公有云廉价且无限容量的对象存储
- 传统数仓缺乏弹性和存算分离，难以利用公有云的优势

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

用户期望一个兼顾关系型数仓和公有云优势的产品

OpenPie

计算引擎方面

- 完备的SQL语言支持
- 高效的分布式计算能力
- 完备的事务支持，隔离性 一致性 原子性

公有云特性方面

- 存算分离
- 弹性的计算集群
- 只为必要的计算付费

Data Computing for New Discoveries

数 据 计 算 ， 只 为 新 发 现

PieCloudDB 能给用户带来什么？

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

对 SQL 的完备支持

OpenPie

ANSI 标准 SQL 的完备支持

- Agg
- Subplan
- Sublink
- Outer join
- Window agg
- Materialized view

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

高效的查询优化和与之匹配的执行器

OpenPie

专门为复杂查询设计的优化器

- 全面的逻辑优化（谓词下推，子查询子链接提升，外连接消除）
- 纯粹基于代价的物理优化
- 全面的数据分布特性描述，分布式代价估算，高效分布式表连接
- 多阶段的聚集

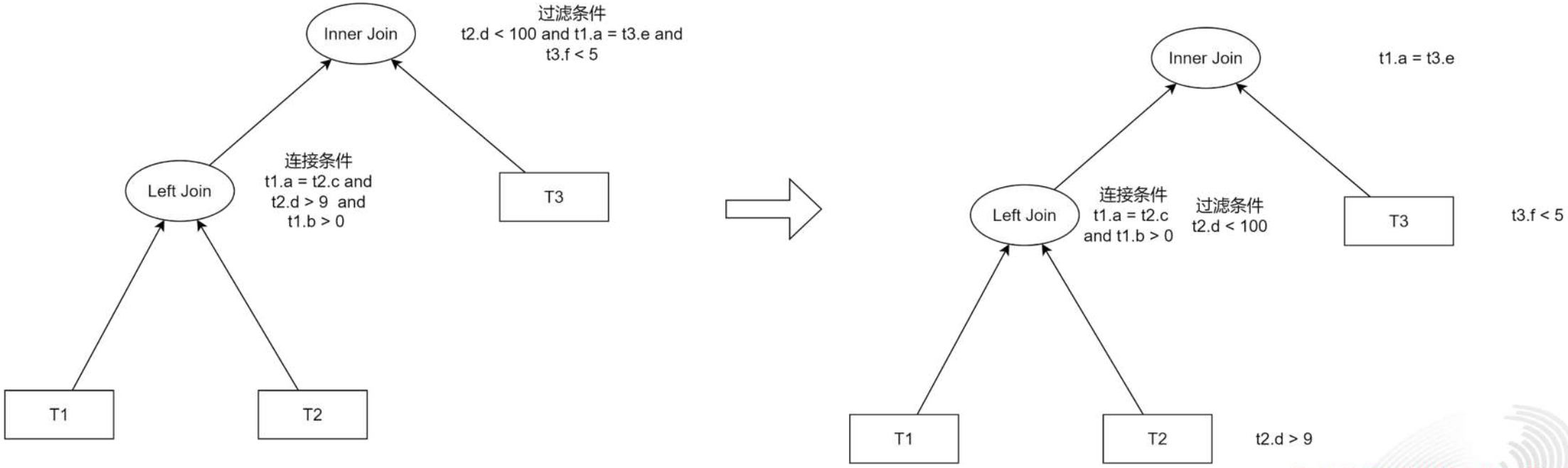
分布式环境高效执行器

- 多阶段执行模型
- 流式数据重分布

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

逻辑优化

select * from t1 left join t2 on t1.a = t2.c and t2.d > 9 and
t1.b > 0 inner join t3 where
t2.d < 100 and t1.a = t3.e and t3.f < 5;



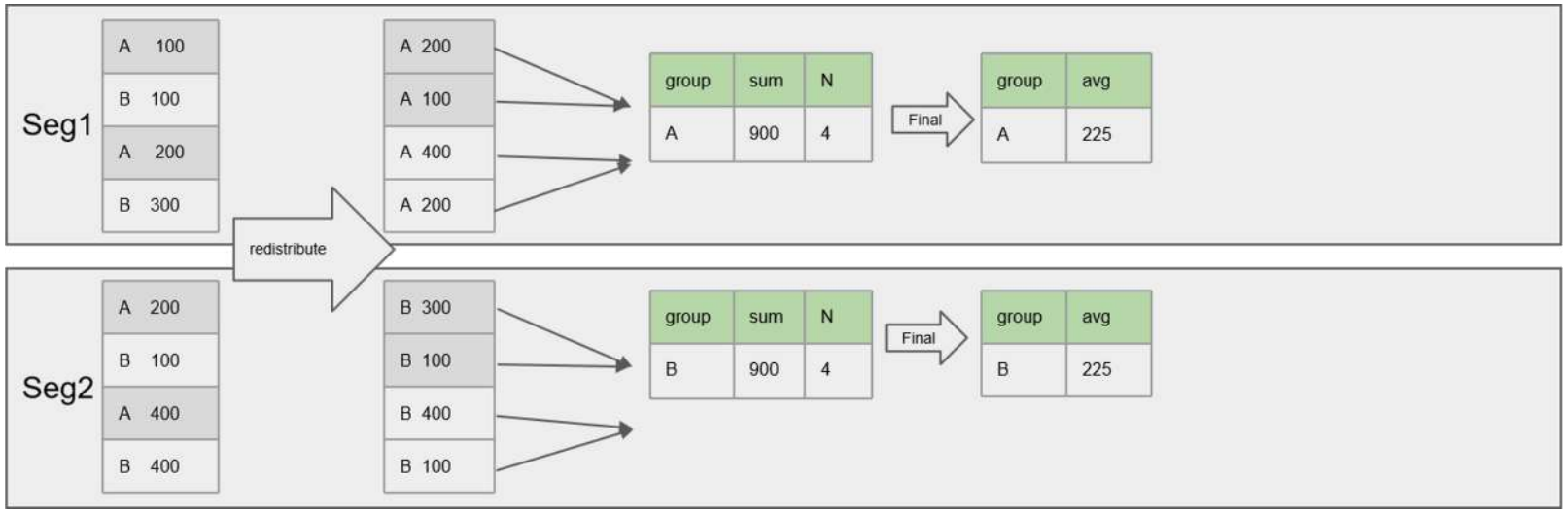
Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

多阶段聚集

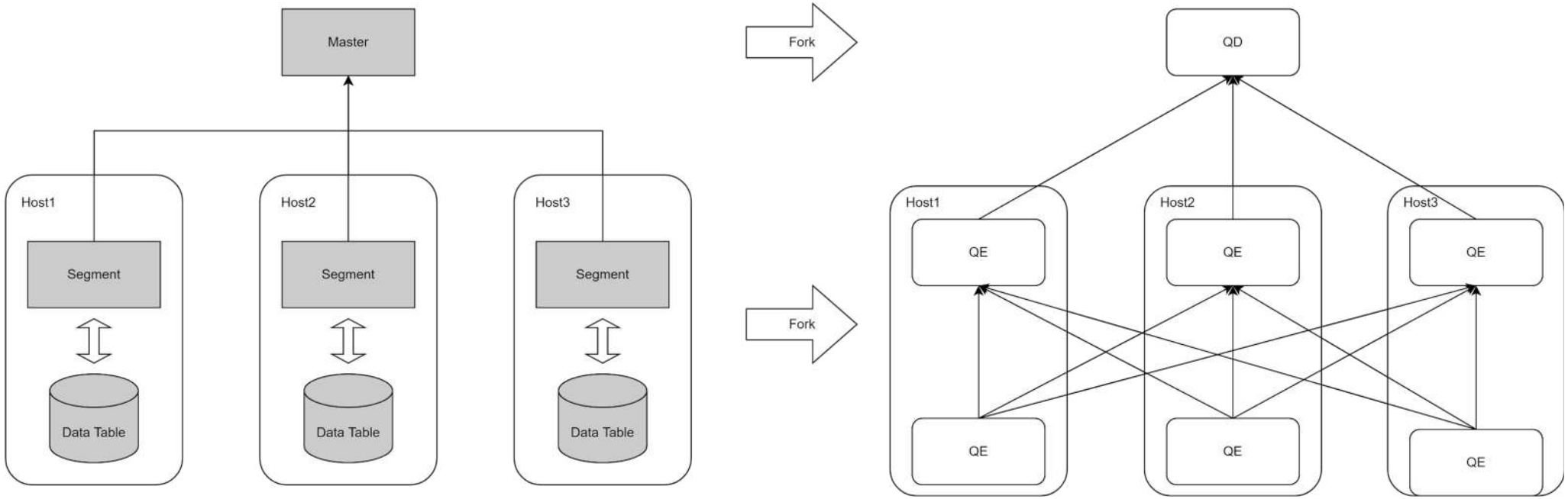
```
explain (costs false) select avg(quantity) from sales;
                QUERY PLAN
-------------------------------------------
---
 Aggregate
   ->  Gather Motion 3:1  (slice1; segments: 3)
         ->  Aggregate
               ->  Seq Scan on sales
```

```
explain (costs false) select avg(quantity) from sales group by brand;

                        QUERY PLAN
-----------------------------------------------------------
 Gather Motion 3:1  (slice2; segments: 3)
   ->  GroupAggregate
         Group Key: sales.brand
         ->  Redistribute Motion 3:3  (slice1; segments: 3)
               Hash Key: sales.brand
               ->  GroupAggregate
                     Group Key: sales.brand
                     ->  Seq Scan on sales
```

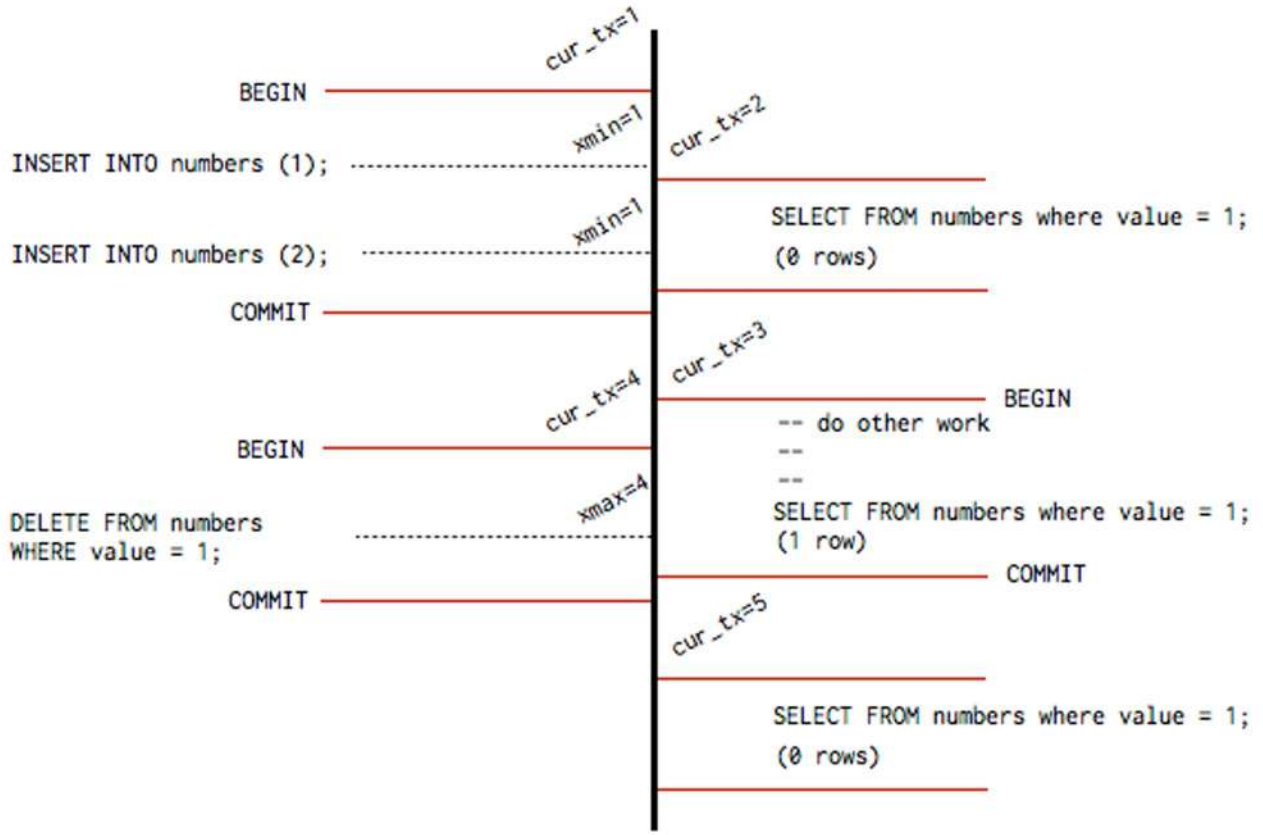


优秀的执行器



对事务（ACID）的完备支持

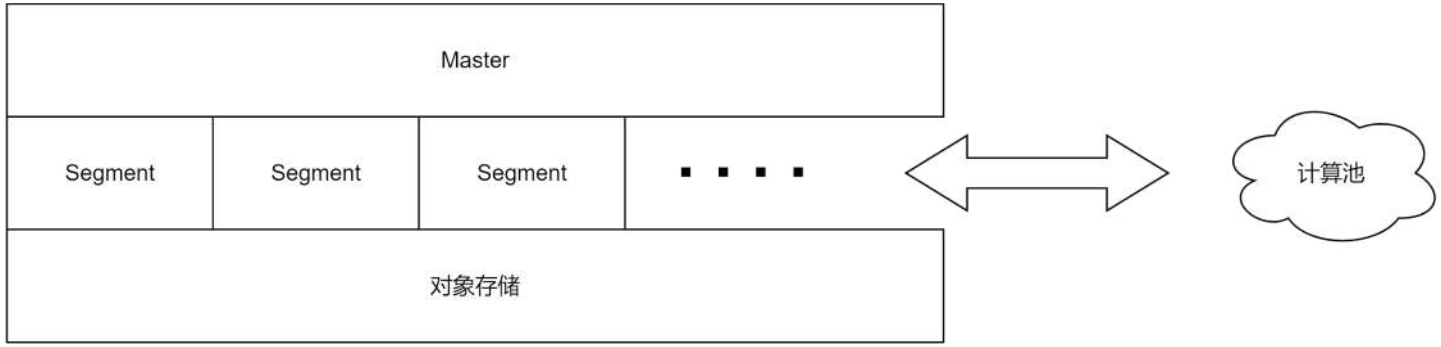
- 原子性
- 一致性
- 隔离性
- 持久性



Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

极致的计算集群弹性

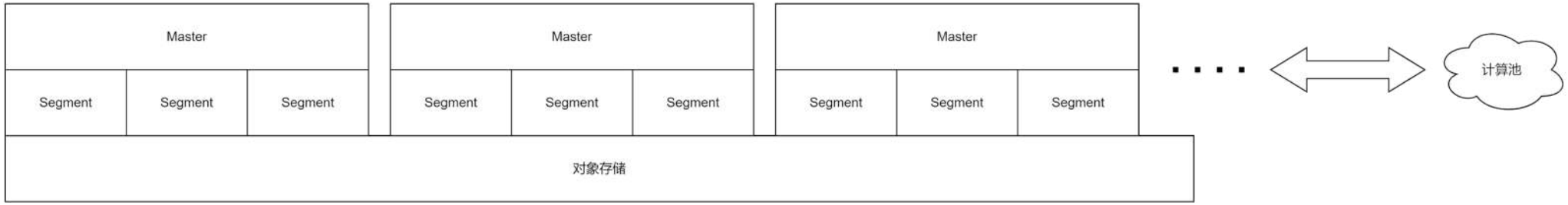
- Segment节点并不持有持久化的数据，在扩张/收缩的过程中不涉及数据的移动
- Segment节点不直接访问系统表，事务和锁
- 在扩张时只需要在新的虚拟机节点上部署二进制并向元数据服务注册



Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

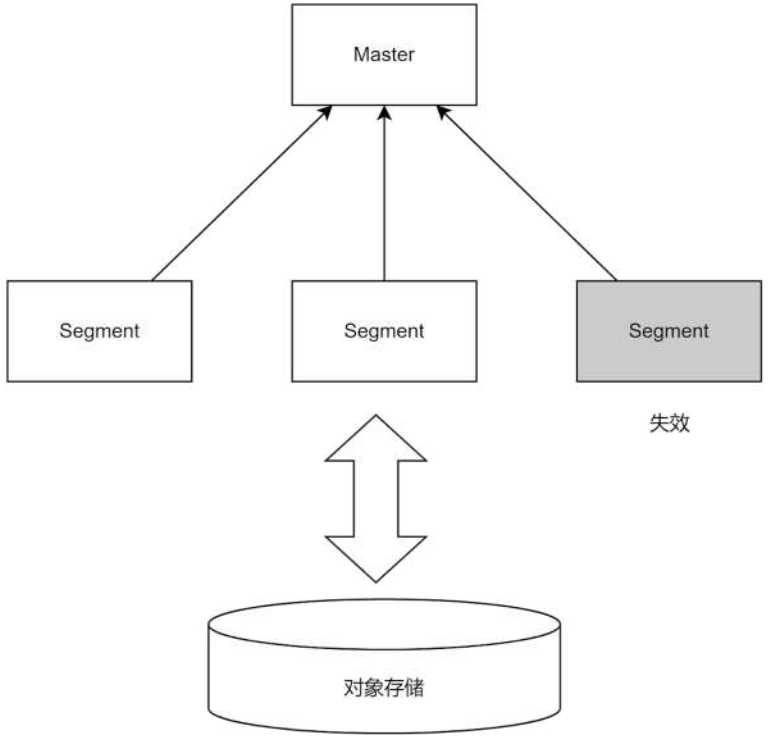
多集群

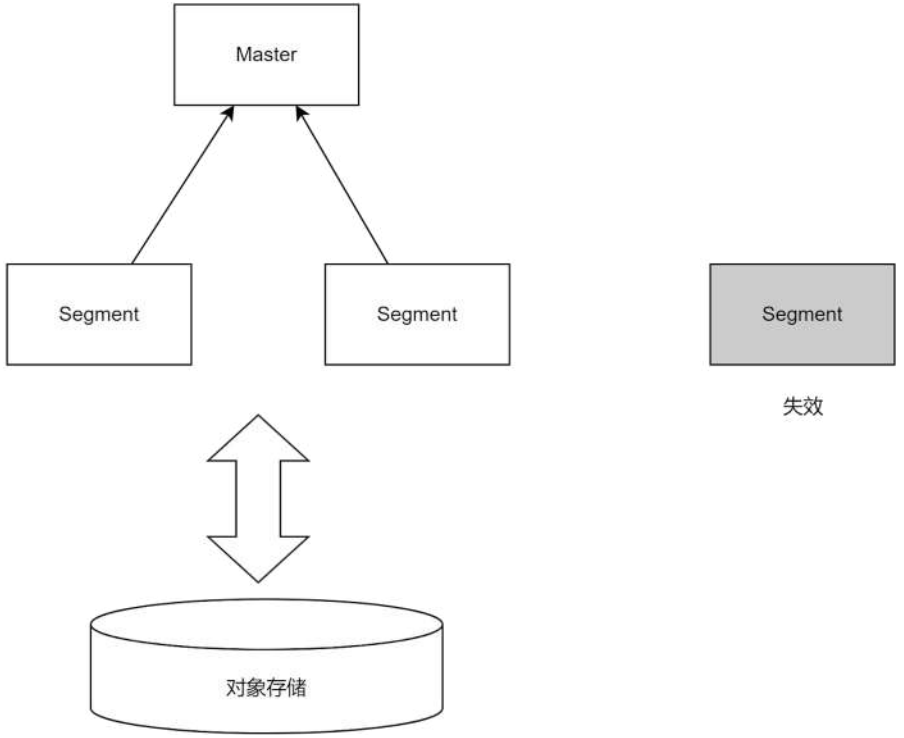
- Master 节点和 FoundationDB 通过事务的方式协同实现了分布式的事务和锁
- 系统表以 mstore 的方式存储在 FoundationDB 上
- Master节点本地不持有任何全局状态



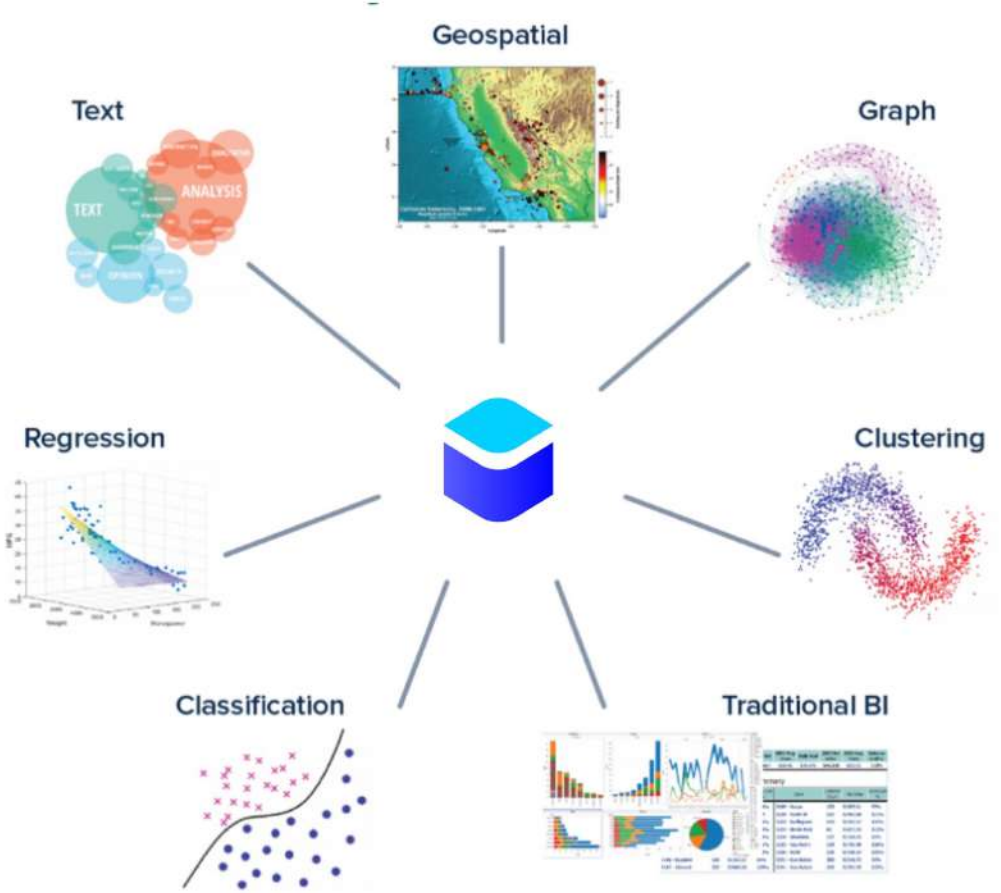
Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

高可用





兼容 Postgres 生态



Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

语言支持

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

给用户带来的好处

- 在 AP 场景下，像使用 postgres 一样使用 PieCloudDB
- 只为已经发生的计算和存储付费
- 按需启动的关闭多个不同大小的集群，以适应不同类型的任务
- 取得性能和开发效率的高度平衡

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

OpenPie

PieCloudDB 云原生架构

Data Computing for New Discoveries

数 据 计 算 ， 只 为 新 发 现

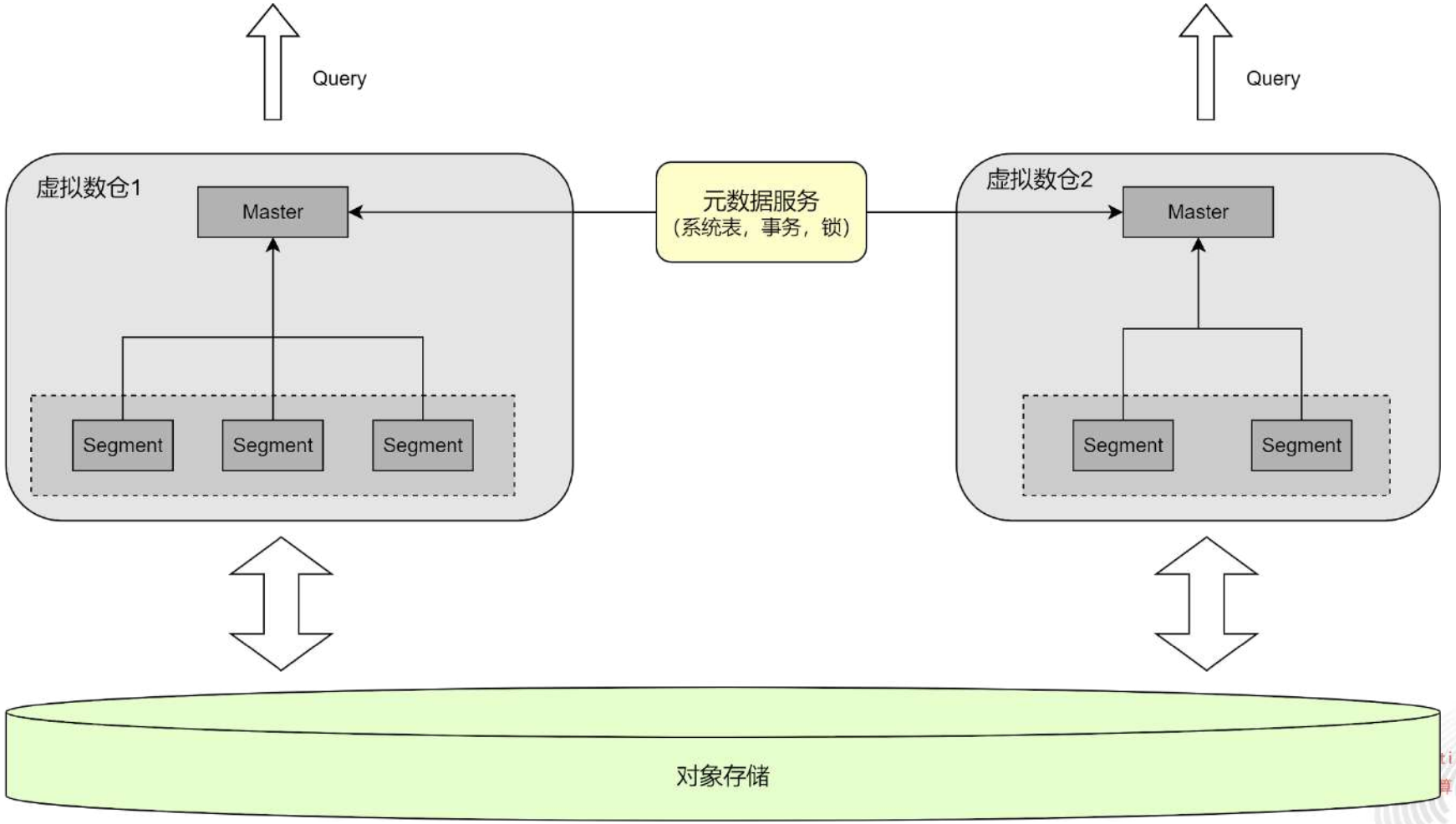
云原生特性的实现途径

OpenPie

- 弹性伸缩的集群
  - 完全无状态的Segment节点
- Multi-cluster
  - 独立的系统表
  - 分布式的锁和事务

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

虚拟数仓



系统表——mStore

- 将元组以key-value的形式存储到FoundationDB
- 使用原有的机制实现mvcc
- 使用foundationdb key的自然排序实现index

- Xmin：创建这个tuple的事务 id
- Xmax：删除这个tuple的事务id
- ctid：指向update的下一个tuple

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

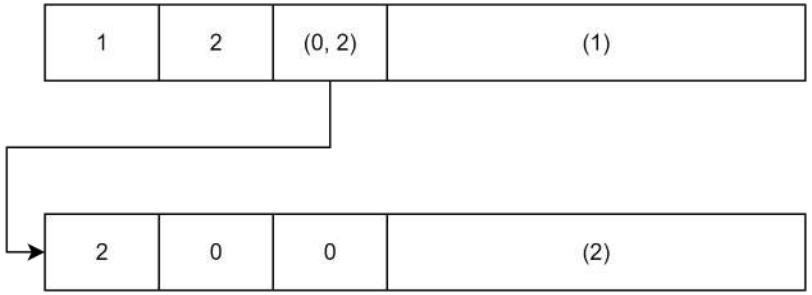
Insert

Transaction1

insert into t values(1);

Transaction2

update t set a = 2 where a = 1;



Scan

Transaction1

begin;
insert into t values(1);
commit

Transaction2

begin;
insert into t values(2);
commit;

Transaction3

begin;
insert into t values(3);

Transaction4

select * from t;
????

Transaction1: insert into t values(1);

1	0	0	(1)

Transaction2: insert into t values(2);

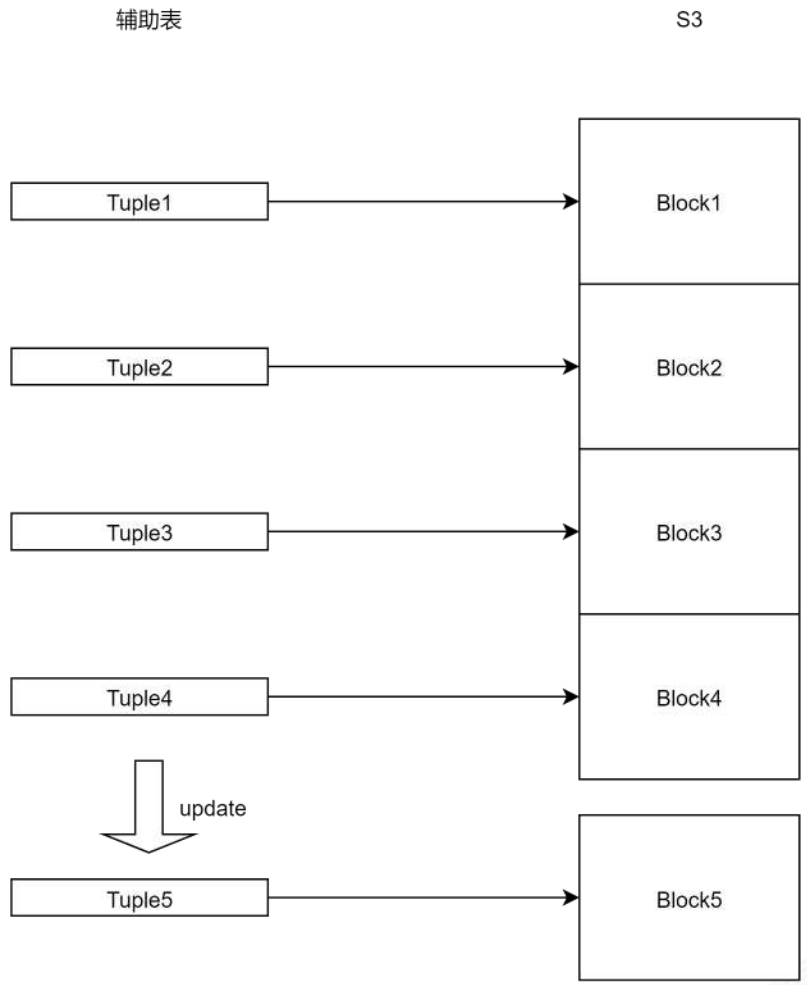
2	0	0	(2)

Transaction3: insert into t values(3);

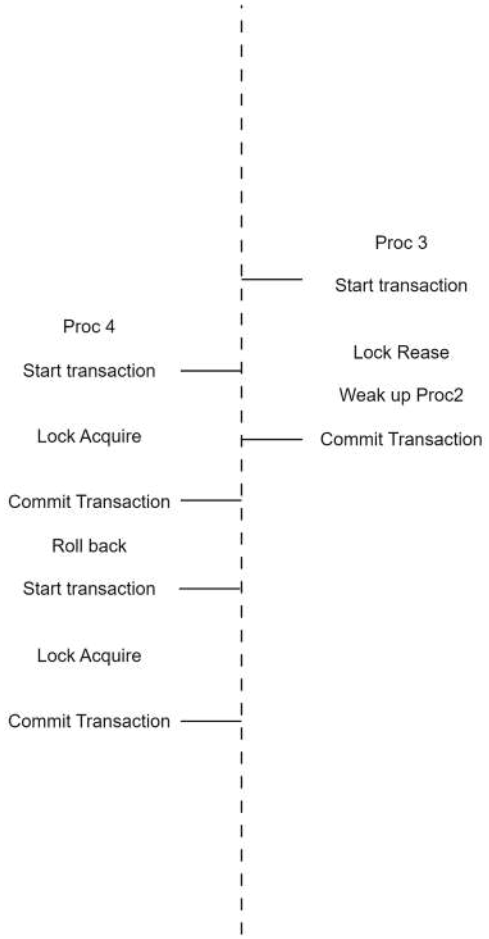
3	0	0	(3)

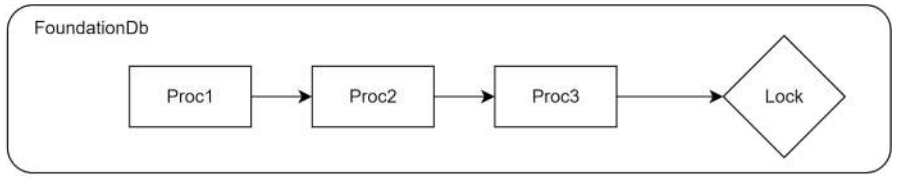
数据表 — Ostore

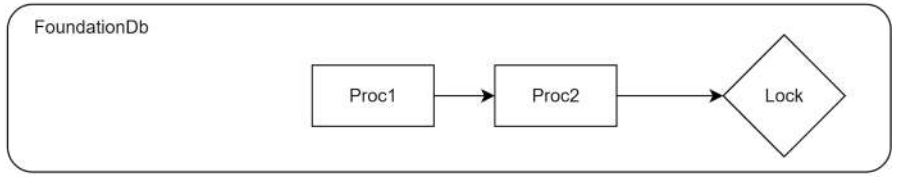
- 使用mstore作为辅助表实现mvcc
- 每个block在辅助表中对应一个tuple
- update/delete生成一个新的block

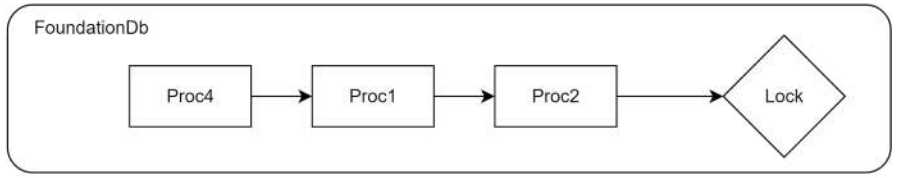


分布式锁

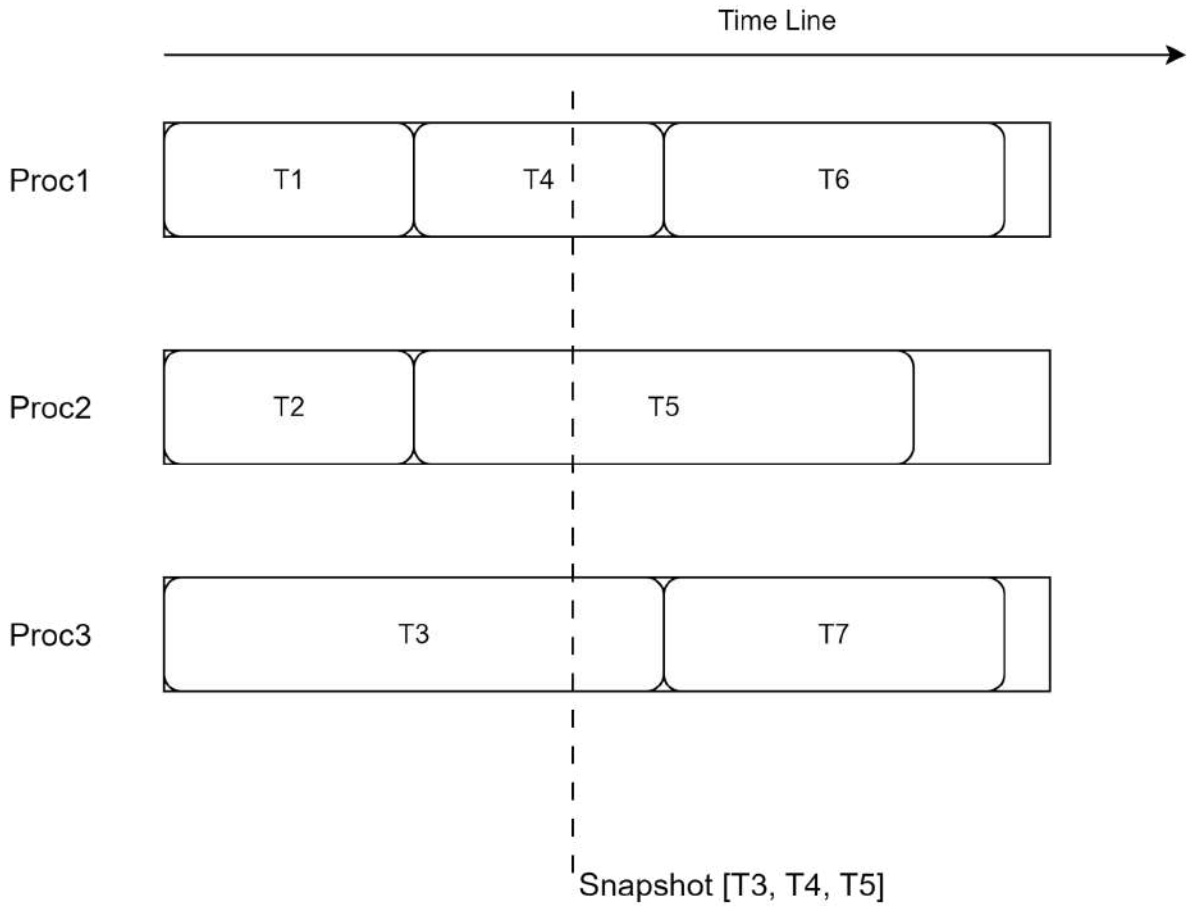








分布式事务



OpenPie

PieCloudDB Key Features

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

Time Travel

OpenPie

原始 SQL：

select * from t1, t2 where t1.a = t2.c;

Time travel 到时间点

select * from t1 at ‘2023-03-20 10:30:33’, t2 at ‘2023-03-19 11:32:23’ where t1.a = t2.c;

Time travel 到某个时间段之前

select * from t1 before ‘1 day’, t2 before ‘30 min’ on t1.a = t2.c;

恢复删除或者修改表

Drop table t1;

Restore table t;

Data Computing for New Discoveries

数 据 计 算 ， 只 为 新 发 现

Branch

branch t2 from t1;

只需要复制辅助表

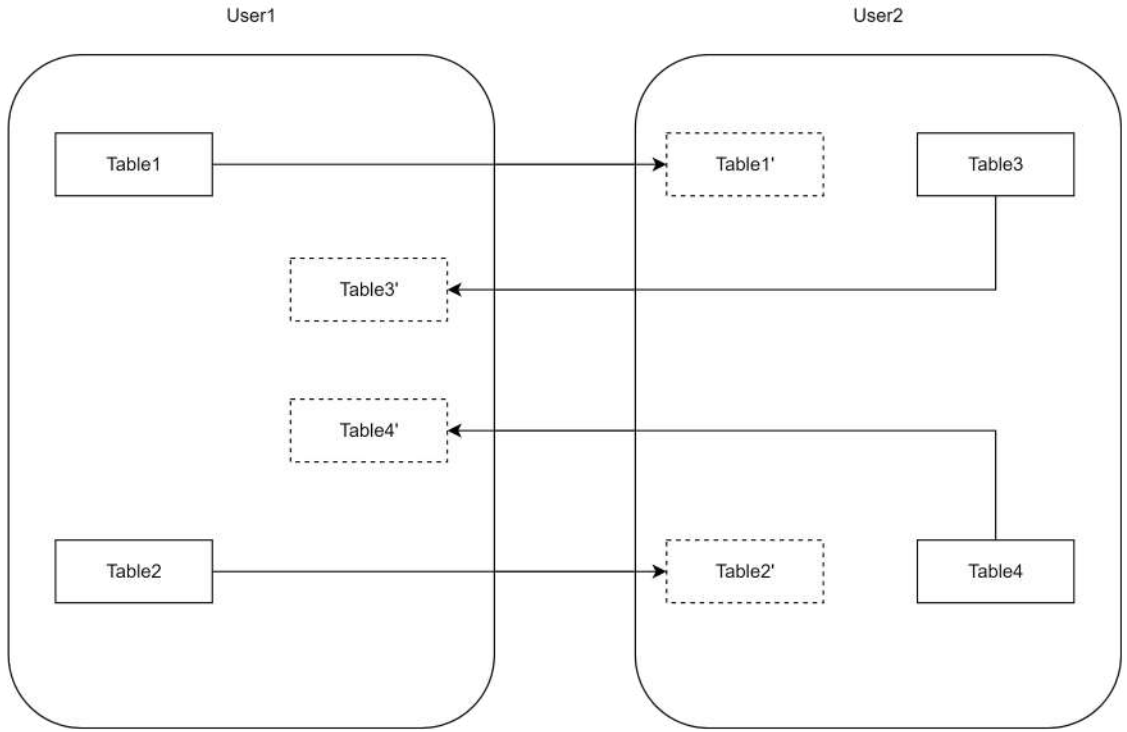
可以从某个历史版本branch

通过引用计数判断block是否删除

只有在vacuum和branch操作时需要访问引用计数

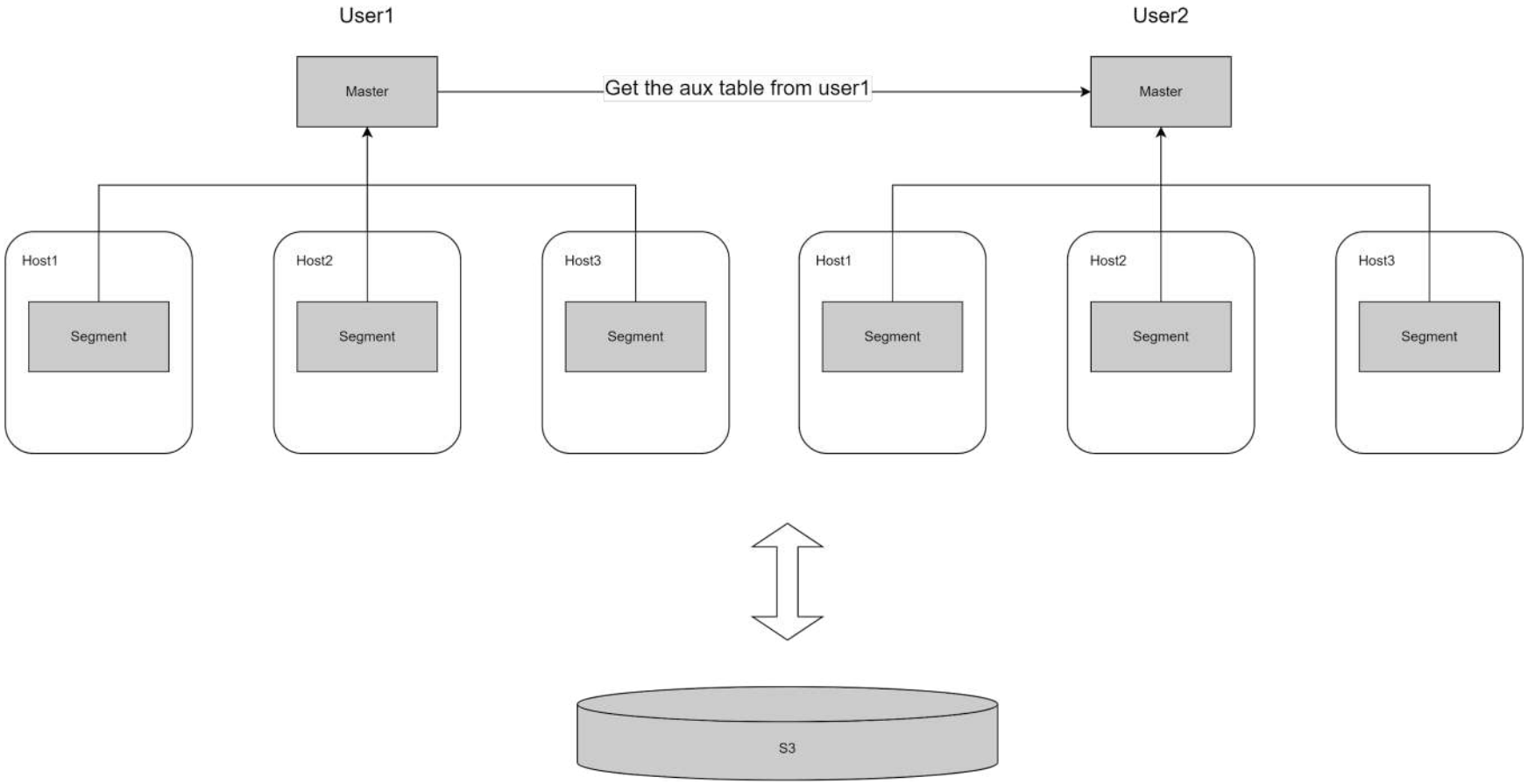
Data Sharing

- 数据的载体对象存储作为整个云的基础设施，每个用户都可以访问
- OStore的辅助表包含访问对象存储中数据的所有信息
- 辅助表的数据量仅为主表大小的百万分之一
- 不同的用户间可以通过分享辅助表的方式实现表共享



Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

Data Sharing

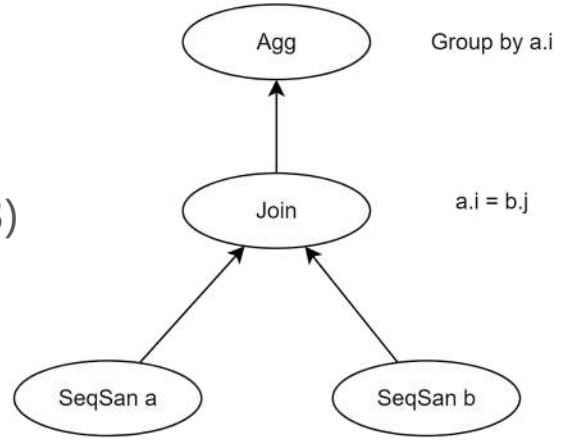


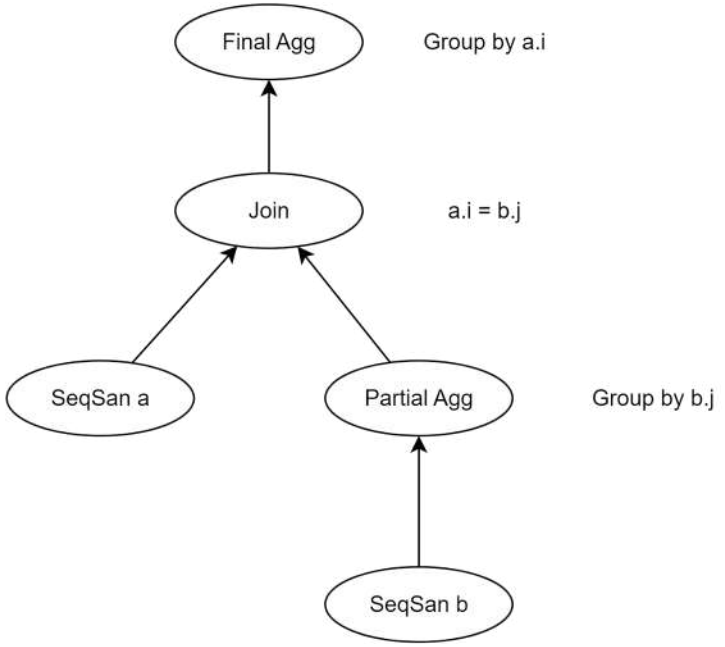
Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

聚集下推

```
EXPLAIN (COSTS OFF)
   SELECT a.i, avg(b.y)
   FROM a JOIN b ON a.i = b.j
   GROUP BY a.i;

   Gather Motion 3:1  (slice1; segments: 3)
    ->  Finalize GroupAggregate
        Group Key: a.i
        ->  Sort
            Sort Key: a.i
            ->  Hash Join
                Hash Cond: (a.i = b.j)
                ->  Seq Scan on a
                ->  Hash
                    ->  Partial HashAggregate
                        Group Key: b.j
                        ->  Seq Scan on b
```

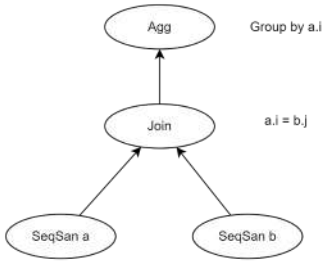


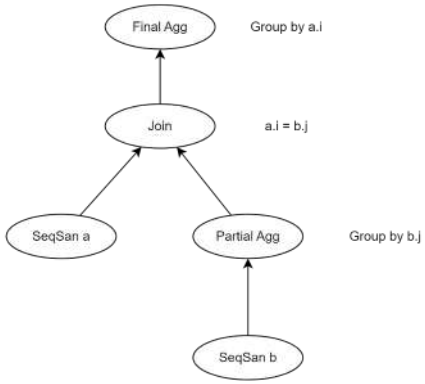


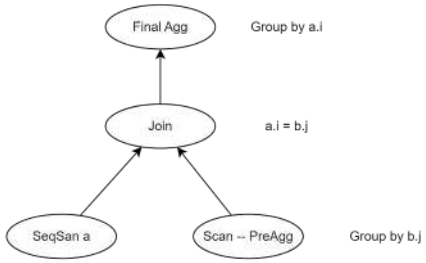
预聚集

```
EXPLAIN (COSTS OFF)
   SELECT a.i, avg(b.y)
   FROM a JOIN b ON a.i = b.j
   GROUP BY a.i;

   Gather Motion 3:1  (slice1; segments: 3)
     -> Finalize GroupAggregate
        Group Key: a.i
        ->  Sort
            Sort Key: a.i
            -> Hash Join
               Hash Cond: (a.i = b.j)
               ->  Seq Scan on a
               ->  Hash
                     -> Scan – Partial HashAggregate
                        Group Key: b.j
```







Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

Advanced Subquery/Sublink pull up

SELECT * FROM part p1 WHERE p1.p_size > 40 OR p1.p_retailprice > (SELECT avg(p2.p_retailprice) FROM part p2 WHERE p2.p_brand = p1.p_brand)

如果在一个大数据量查询中sublink不能提升，外表每扫描一个元组，subquery都要被执行一次，Query可能永远跑不出结果

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------------
 Gather Motion 3:1  (slice2; segments: 3)  (cost=0.00..50721322934218.25 rows=23179962 width=133)
   ->  Seq Scan on part p1  (cost=0.00..50721322934218.25 rows=7726654 width=133)
         Filter: p_size > 40 OR p_retailprice > ((subplan))
         SubPlan 1
           ->  Aggregate  (cost=990668.22..990668.23 rows=1 width=32)
                 ->  Result  (cost=905677.71..926157.35 rows=682655 width=8)
                       Filter: p2.p_brand = $0
                       ->  Materialize  (cost=905677.71..926157.35 rows=682655 width=8)
                             ->  Broadcast Motion 3:3  (slice1; segments: 3)  (cost=0.00..903629.75 rows=682655 width=8)
                                   ->  Seq Scan on part p2  (cost=0.00..903629.75 rows=682655 width=8)
 Settings:  optimizer=off
 Optimizer status: legacy query optimizer
(12 rows)
```

Pushing Predicates Below CTEs

OpenPie

CTE在SQL中的应用非常广泛（TPC-DS有48个query包含CTE）

CTE用于SQL的重用

在Postgres中谓词不会被下推到CTE中，这会影响性能

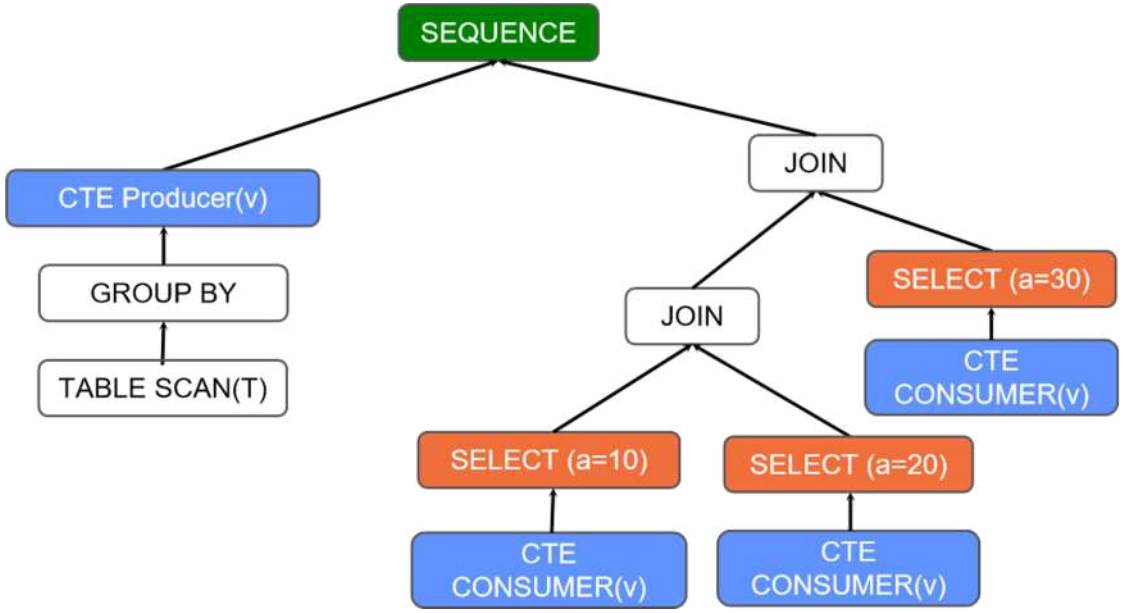
PieCloudDB实现了CTE的聚集下推

Data Computing for New Discoveries

数 据 计 算 ， 只 为 新 发 现

Pushing Predicates Below CTEs

```
WITH v AS (SELECT a, sum(b) as s FROM T GROUP BY a)
SELECT *
FROM v as v1, v as v2, v as v3
WHERE v1.a < v2.a
AND v1.s < v3.s
AND v1.a = 10
AND v2.a = 20
AND v3.a = 30;
```



Recursive CTE

```
WITH RECURSIVE subdepartment AS.
(
--non recursive term
SELECT * FROM departments WHERE name = 'A'
UNION ALL
--recursive term referring to "subdepartment"
SELECT d.* FROM department AS d, subdepartment AS sd
WHERE sd.id = d.parent_department
)
SELECT * FROM subdepartment;
```

id	parent_department	name
0	NULL	'ROOT'
1	0	'A'
2	1	'B'
3	2	'C'
4	2	'D'
5	0	'E'
6	4	'F'
7	4	'G'

Table 1: Table: DEPARTMENT

Dynamic Partition Elimination

```
bootcamp=# explain SELECT year FROM catalog_sales JOIN date_dim ON (date_id=date_dim.id) GROUP BY year;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------
 Gather Motion 2:1  (slice3; segments: 2)  (cost=0.00..863.06 rows=1 width=4)
   ->  GroupAggregate  (cost=0.00..863.06 rows=1 width=4)
         Group By: date_dim.year
         ->  Sort  (cost=0.00..863.06 rows=1 width=4)
               Sort Key: date_dim.year
               ->  Redistribute Motion 2:2  (slice2; segments: 2)  (cost=0.00..863.06 rows=1 width=4)
                     Hash Key: date_dim.year
                     ->  HashAggregate  (cost=0.00..863.06 rows=1 width=4)
                           Group By: date_dim.year
                           ->  Hash Join  (cost=0.00..863.05 rows=60 width=4)
                                 Hash Cond: catalog_sales.date_id = date_dim.id
                                 ->  Dynamic Table Scan on catalog_sales (dynamic scan id: 1)  (cost=0.00.. rows=5000 width=4)
                                 ->  Hash  (cost=100.00..100.00 rows=50 width=4)
                                       ->  Partition Selector for catalog_sales(dynamic scan id: 1)(cost=10... rows=50 width=4)
                                             Filter: catalog_sales.id = date_dim.id
                                             ->  Broadcast Motion 2:2  (slice1; segments: 2)  (cost=0.00..431.00 rows=12 width=8)
                                                   ->  Table Scan on date_dim  (cost=0.00..431.00 rows=6 width=8)
 Settings:  optimizer=on
 Optimizer status: PQO version 2.40.0
(19 rows)
```

Data Computing for New Discoveries
数 据 计 算 ， 只 为 新 发 现

OpenPie

关注PieCloudDB公众号

随时获得产品动态

加入PieCloudDB技术群

获得更多技术干货

Data Computing for New Discoveries

数 据 计 算 ， 只 为 新 发 现

OpenPie

THANKS

Data computing for New Discoveries

数 据 计 算 ， 只 为 新 发 现