PostgreSQL中文社区

2021 PostgreSQL China Conference

主办：PostgreSQL 中文社区

第 11 届 PostgreSQL 中国技术大会

开源论道 × 数据驱动 × 共建数字化未来

Citus插件实现原理解读

阿里云PolarDB PG数据库团队| 陈佳昕（步真）

什么是Citus

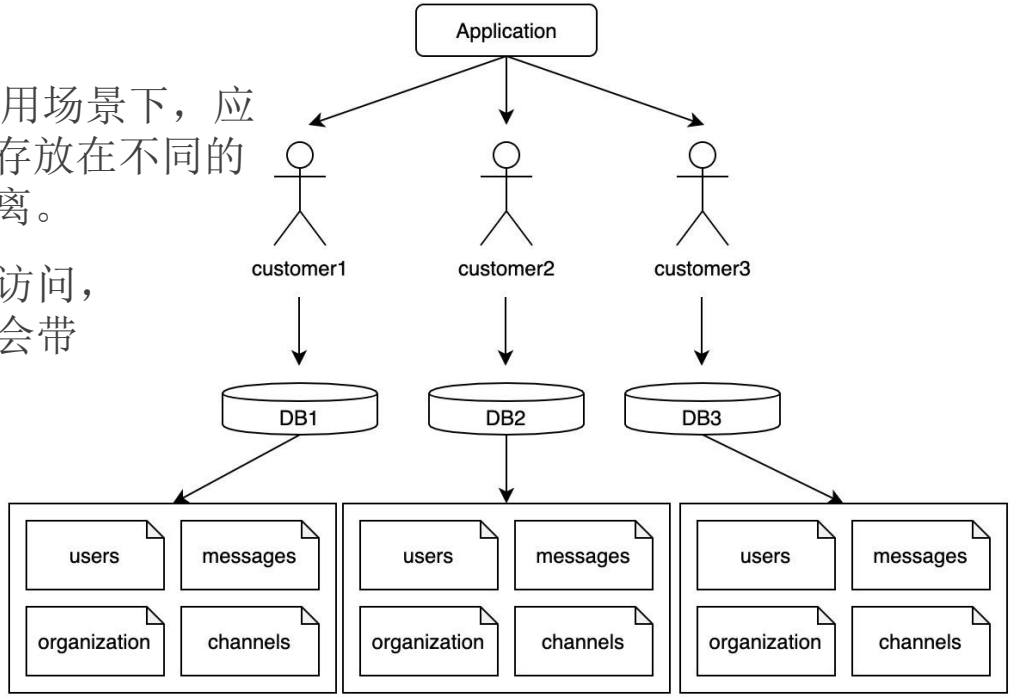
- Citus是一款基于PostgreSQL数据库的开源插件，用于将单集群的PG数据库横向扩展为多集群的分布式PG数据库。其提供了分布式数据库常用的两种表类型：
  - 分布表
  - 复制表
- 主要作用：
  - 通过分布式多路路由，提升数据库事务处理能力（OLTP）
  - 通过并行化处理机制，提升数据库分析处理能力（OLAP）
- 来源：由CitusData公司研发，后被微软公司收购，保持跟进最新的开源PostgreSQL社区分支功能特性。

Citus的4种目标业务场景

- 多租户场景（Multi-tenant）

背景：传统的基于SaaS云服务应用场景下，应用程序可能根据不同的租户数据，存放在不同的数据库下，从而实现租户间数据隔离。

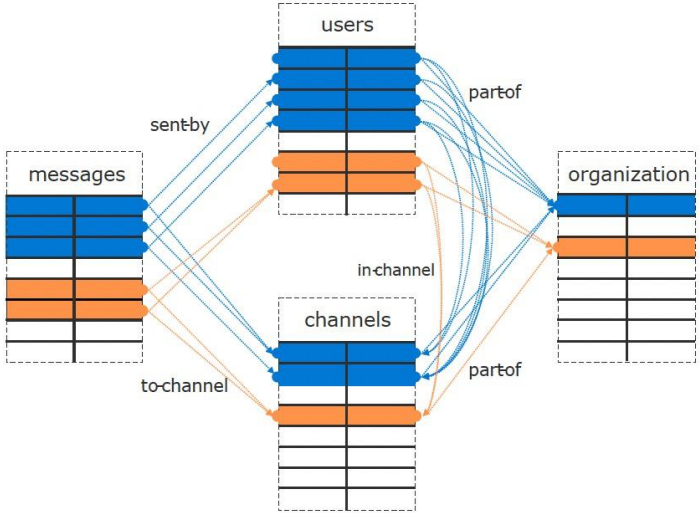
问题：成千上万的库之间的跨库访问，以及多个库基础设施组件的维护会带来极大的成本问题和管理问题。



Citus的4种目标业务场景

- 多租户场景（Multi-tenant）

Citus应对方案：基于租户id进行分片，相同分片的数据可以定向到同一节点上，绝大部分SQL都可以路由到指定单节点执行，能够继承PostgreSQL的查询能力，扩展性强。



Citus的4种目标业务场景

- 实时分析（Real-time Analytics）

  背景：大数据量下毫秒级响应需求查询，额外伴随数据表的更新（TP+AP场景）。

  PG数据库的优势：

  - 特有的堆表格式+MVCC机制保证了多并发读写的高性能
  - 支持COPY，可实现数据批量写
  - 特有的数据类型和索引类型（array，json和自定义数据类型）

  PG数据库的不足：

  - 单库承载的数据容量有限
  - 早期不支持并行查询，查询只能单进程进行。

  Citus应对方案： Parallel Bulk Insert， Parallel Insert ... Select， Parallel Select等并行功能

Citus的4种目标业务场景

- 高性能数据处理（High Performance CRUD）

  PG数据库的不足：

  - 缺乏undo机制使得数据更新会带来数据膨胀，auto vacuum使得高并发下数据库压力大
  - 大量连接会话引入内存过多，极易打满单库机器内存上限

  Citus应对方案：

  - 数据分片打散，提高硬件资源利用率。
  - 支持并行化auto vacuum。

Citus的4种目标业务场景

- 数据仓库（Data warehouse）

  Citus应对方案：

  - 支持并行分布式查询
  - 支持列存引擎。

Citus的实现架构

- SQL Objects

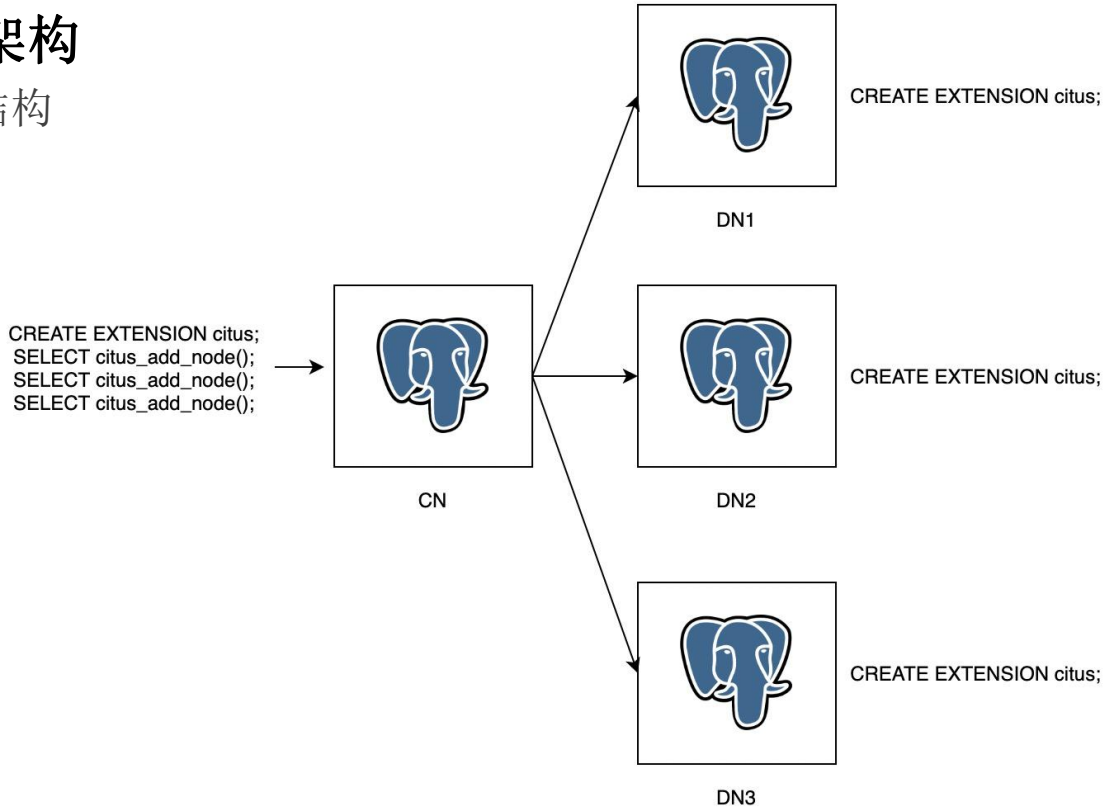
```
CREATE TABLE pg_dist_node(…);
CREATE TABLE pg_dist_partition(…);

CREATE FUNCTION citus_add_node(…)
RETURNS void LANGUAGE c
AS '$libdir/citus',
$function$citus_add_node$function$;

CREATE FUNCTION create_distributed_table(…)
RETURNS void LANGUAGE c
AS '$libdir/citus',
$function$create_distributed_table$function$;
```

- Shared Library

```
#include "postgres.h"

Datum citus_add_node(…)
{
                …
}

Datum create_distributed_table(…)
{
                …
}
```

Citus的实现架构

- 集群部署结构



Citus的实现架构

表类型

- 分布表（co-location)

SELECT create_distributed_table('items', 'user_id');

SELECT create_distributed_table('users', 'user_id',

colocate_with = 'items');

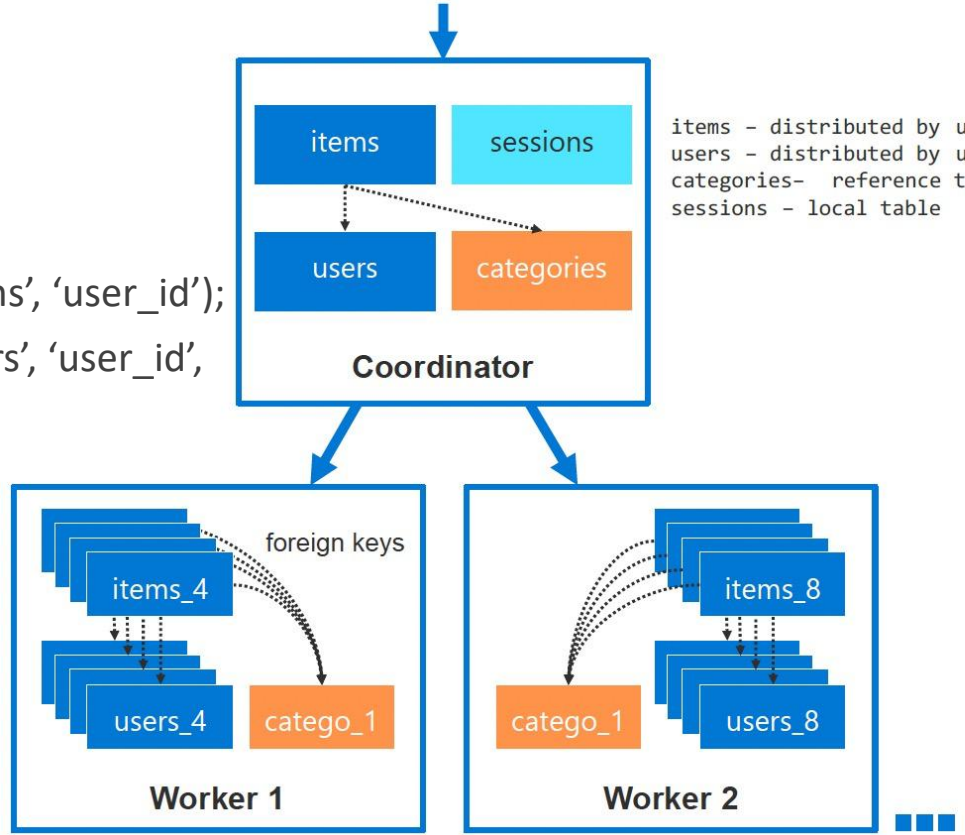
- 复制表

SELECT create_reference_table('

categories');

- 本地表

CREATE TABLE sessions();



Citus的实现架构

元数据管理

```
--CN
postgres=# create table test(id int);
CREATE TABLE

postgres=# select create_distributed_table('test','id');
 create_distributed_table
--------------------------

(1 row)

--CN
postgres=# \d
            List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+----------
 public | test        | table | postgres
(1 rows)
```

```
--DN1
postgres=# \d
            List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+----------
 public | test_102008 | table | postgres
(1 rows)

--DN2
postgres=# \d
            List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+----------
 public | test_102009 | table | postgres
(1 rows)
```

Citus的实现架构

元数据管理

```
postgres=# select * from pg_dist_shard;
 logicalrelid | shardid | shardstorage | shardminvalue | shardmaxvalue
--------------+---------+--------------+---------------+---------------
 test         |  102008 | t            | -2147483648   | -1
 test         |  102009 | t            | 0             | 2147483647
(2 rows)

postgres=# select * from pg_dist_placement;
 placementid | shardid | shardstate | shardlength | groupid
-------------+---------+------------+-------------+---------
           1 |  102008 |          1 |           0 |       1
           2 |  102009 |          1 |           0 |       2
(2 rows)

 postgres=# select * from pg_dist_node;
  nodeid | groupid | nodename  | nodeport | noderack | hasmetadata | isactive | noderole | nodecluster
 --------+---------+-----------+----------+----------+-------------+----------+----------+-------------
       1 |       1 | 127.0.0.1 |     5432 | default  | f           | t        | primary  | default
       2 |       2 | 127.0.0.1 |     5433 | default  | f           | t        | primary  | default
 (2 rows)
postgres=# select * from pg_dist_partition;
-[ RECORD 1 ]+----------------------------------------------------------------------------------------------------------------------------------
logicalrelid | test
partmethod   | h
partkey      | {VAR :varno 1 :varattno 1 :vartype 23 :vartypmod -1 :varcollid 0 :varlevelsup 0 :varnoold 1 :varoattno 1 :location -1}
colocationid | 1
repmodel     | c
```

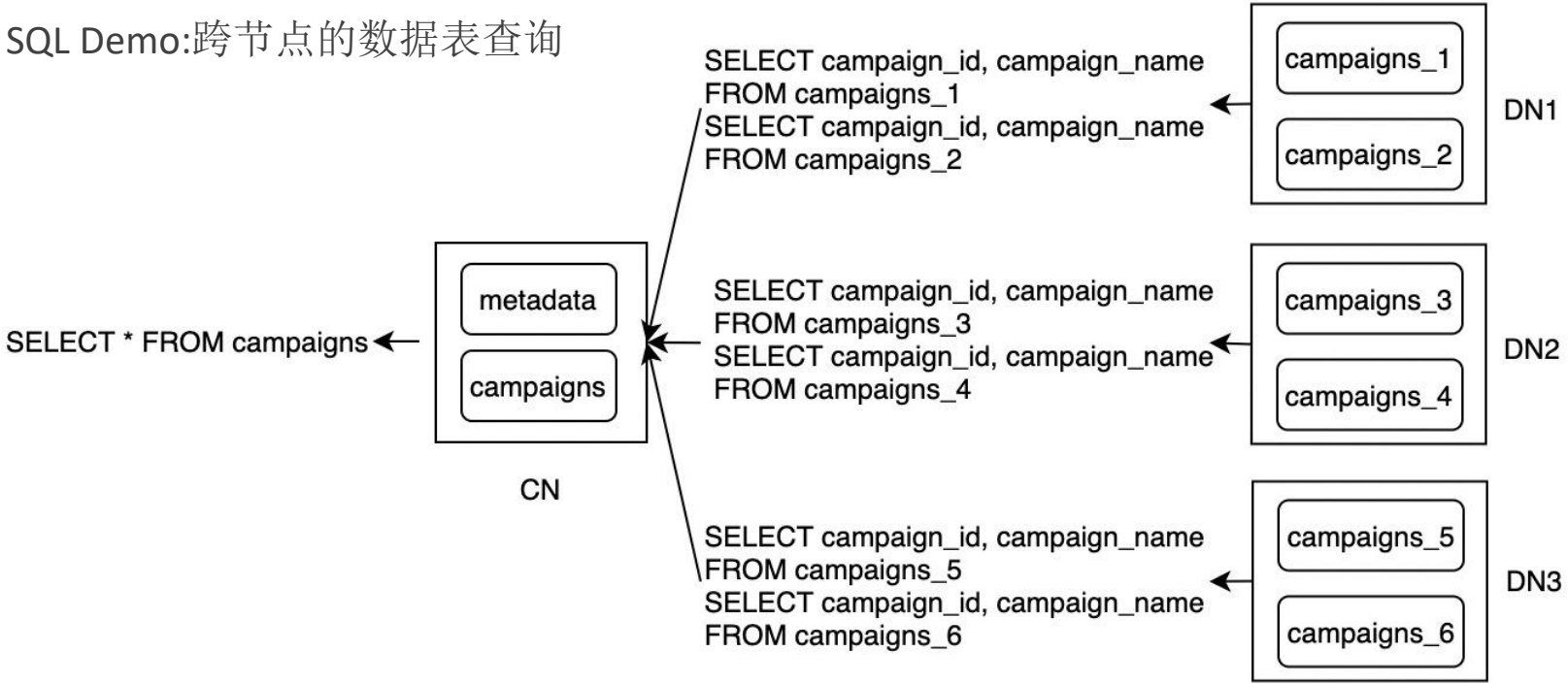
Citus的实现架构

PostgreSQL插件API

- 自定义SQL函数：插件的SQL函数以及对应的C函数。更新与管理citus元数据
- Planner/executor hook：分布式计划与分布式执行
- CustomScan节点：分发与执行分布式计划
- 事务回调：分布式事务，两阶段事务管理。
- Utility hook：分布式DDL、COPY。
- Citus Daemon进程：后台常驻守护进程。节点探活、全局死锁检测、crash recovery状态恢复。

Citus的实现架构

SQL Demo:跨节点的数据表查询



Citus的实现架构

- postgres.c

```
planner_hook_type planner_hook = NULL;

PlannedStmt *
planner(Query *parse, ...)
{
    PlannedStmt *result;
    if (planner_hook)
        result = (*planner_hook) (parse, ...);
    else
        result = standard_planner(parse, ...);
return result;
}
```

- citus.c

```
void _PG_init(void)
{
    ...
    planner_hook = distributed_planner;
    ...
}
PlannedStmt *
distributed_planner(Query *parse, ...)
{
    ...
}
```

Citus的实现架构

- 分布式计划

```
PlannedStmt *
distributed_planner(Query *parse, ...)
{
   bool needsDistributedPlanning = false;

   //1.预校验是否产生分布式计划产生
   List *rangeTableList = ExtractRangeTableEntryList(parse); //递归遍历查询树与子查询树
   needsDistributedPlanning = ListContainsDistributedTableRTE(rangeTableList);

   //2.满足fast path规则，通过FastPathPlanner产生计划树
   if (needsDistributedPlanning && FastPathRouterQuery(parse))
      result = FastPathPlanner(parse, ...);
   //3.不满足fast path规则，通过standard_planner产生计划树
   else
      result = standard_planner(parse, ...);

   //4.产生分布式计划
   if (needsDistributedPlanning)
      result = CreateDistributedPlannedStmt(result, parse,...)

   return result;
}
```

Citus的实现架构

分布式计划

- CreateDistributedPlannedStmt
  - 单表，SELECT查询，带分布键的过滤条件。
    select * from test where key = 100;

```
//2.满足fast path规则，通过FastPathPlanner产生计划树
if (needsDistributedPlanning && FastPathRouterQuery(parse))
    result = FastPathPlanner(parse, ...);
```

```
PlannedStmt *
GeneratePlaceHolderPlannedStmt(Query *parse)
{
    PlannedStmt *result = makeNode(PlannedStmt);
    SeqScan *seqScanNode = makeNode(SeqScan);
    Plan *plan = &seqScanNode->plan;
    Oid relationId = InvalidOid;

    AssertArg(FastPathRouterQuery(parse));

    /* there is only a single relation rte */
    seqScanNode->scanrelid = 1;

    plan->targetlist = copyObject(parse->targetList);
    plan->qual = NULL;
    plan->lefttree = NULL;
    plan->righttree = NULL;
    plan->plan_node_id = 1;

    /*  rtable is used for access permission checks */
    result->commandType = parse->commandType;
    result->queryId = parse->queryId;
    result->stmt_len = parse->stmt_len;

    result->rtable = copyObject(parse->rtable);
    result->planTree = (Plan *) plan;

    relationId = ExtractFirstDistributedTableId(parse);
    result->relationOids = list_make1_oid(relationId);

    return result;
}
```

Citus的实现架构

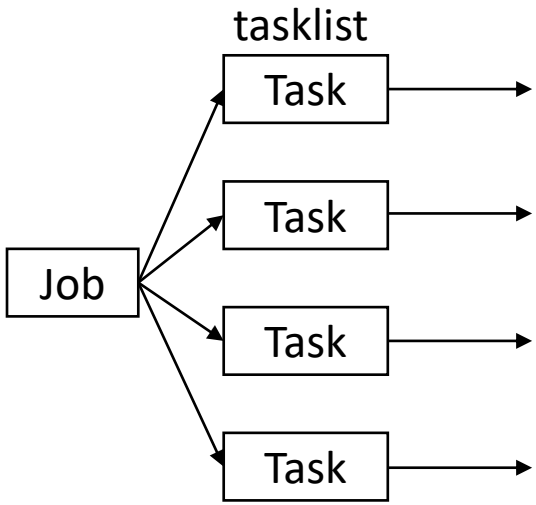
分布式计划节点CustomScan

- CustomScan：PG提供的一个自定义扫描节点。可实现一系列执行器的Begin，Exec，End，Re-scan等回调函数处理逻辑。
- Citus使用了该节点用于扩展实现分布式计划。
- CustomScan存储了分布式计划DistributedPlan。
- DistributedPlan中的workerJob，存储了每个DN节点的执行任务

```
//4.产生分布式计划
if (needsDistributedPlanning)
   result = CreateDistributedPlannedStmt(result, parse,...)
```

```
typedef struct DistributedPlan
{
    CitusNode type;

    /* unique identifier of the plan within the session */
    uint64 planId;

    /* specifies nature of modifications in query */
    RowModifyLevel modLevel;

    /* specifies whether a DML command has a RETURNING */
    bool hasReturning;

    /* a router executable query is executed entirely on a worker */
    bool routerExecutable;

    /* job tree containing the tasks to be executed on workers */
    Job *workerJob;

    /* local query that merges results from the workers */
    Query *masterQuery;

    /* query identifier (copied from the top-level PlannedStmt) */
    uint64 queryId;

    /* which relations are accessed by this distributed plan */
    List *relationIdList;
```

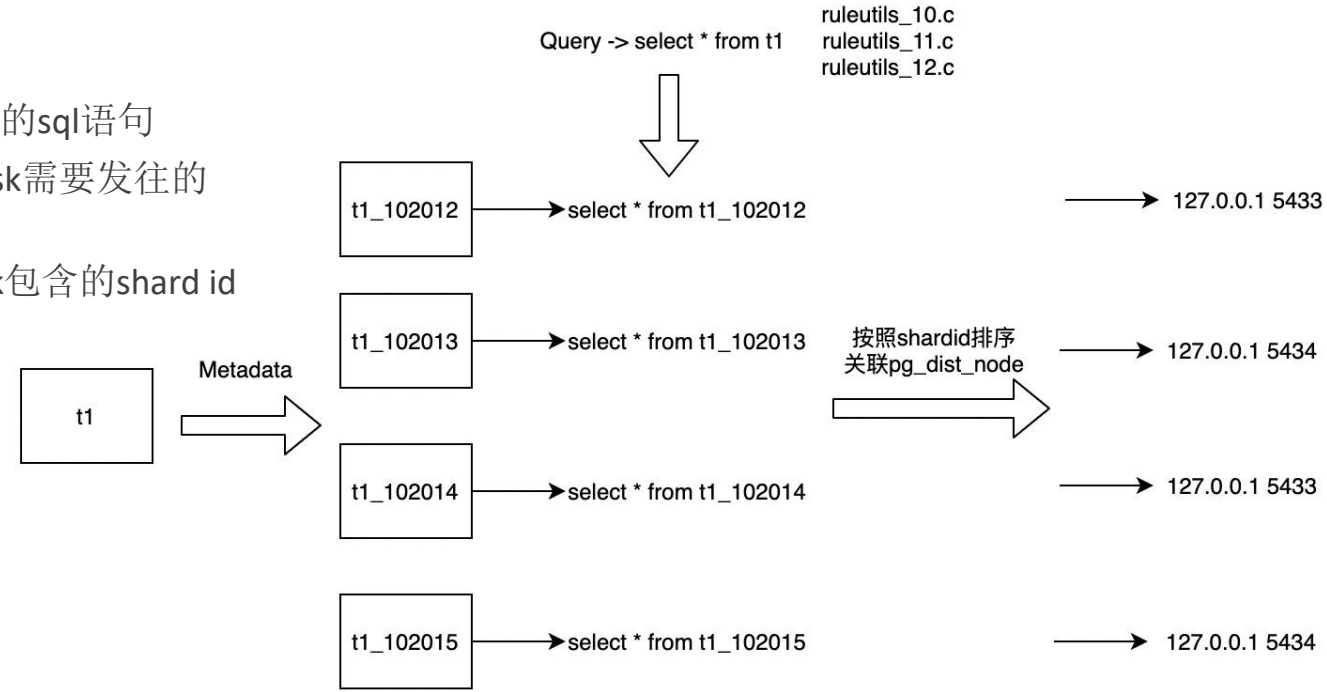
Citus的实现架构




```
postgres=# explain select * from t1;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Custom Scan (Citus Adaptive)  (cost=0.00..0.00 rows=0 width=0)
   Task Count: 4
   Tasks Shown: All
   ->  Task
         Node: host=127.0.0.1 port=5433 dbname=postgres
         ->  Seq Scan on t1_102012 t1  (cost=0.00..23.60 rows=1360 width=8)
   ->  Task
         Node: host=127.0.0.1 port=5434 dbname=postgres
         ->  Seq Scan on t1_102013 t1  (cost=0.00..23.60 rows=1360 width=8)
   ->  Task
         Node: host=127.0.0.1 port=5433 dbname=postgres
         ->  Seq Scan on t1_102014 t1  (cost=0.00..23.60 rows=1360 width=8)
   ->  Task
         Node: host=127.0.0.1 port=5434 dbname=postgres
         ->  Seq Scan on t1_102015 t1  (cost=0.00..23.60 rows=1360 width=8)
(15 rows)
```

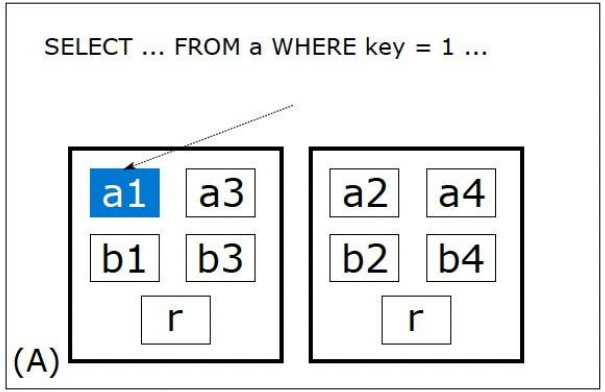
Citus的实现架构

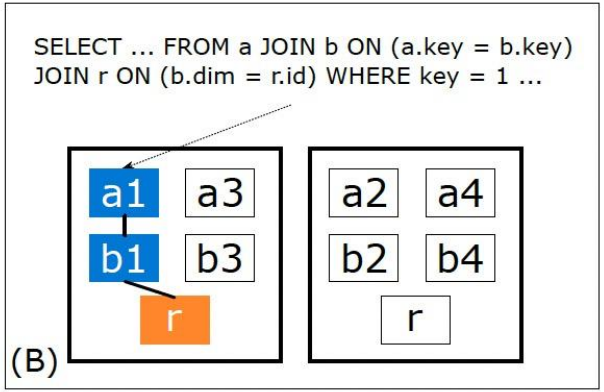
- Task是如何产生的
  - queryString：Task执行的sql语句
  - taskPlacementList：Task需要发往的DN节点路由
  - relationShardList：Task包含的shard id

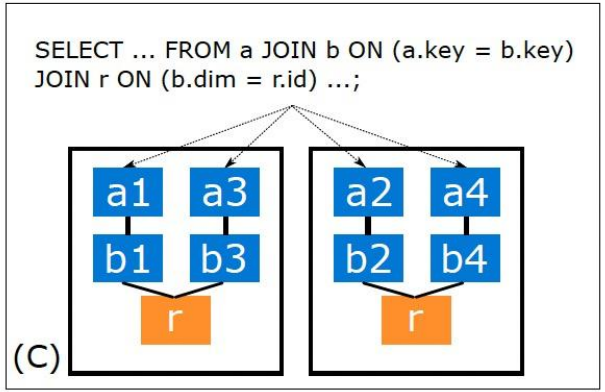


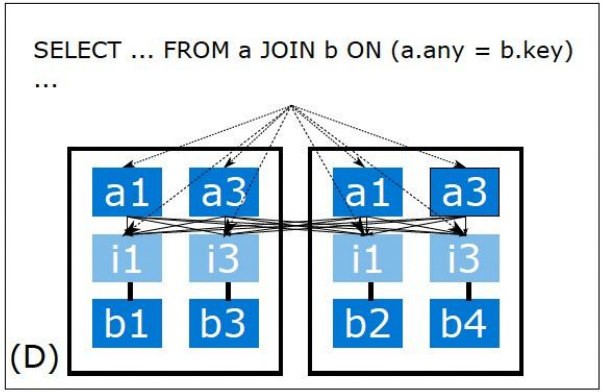
Citus的实现架构

- 分布式计划执行



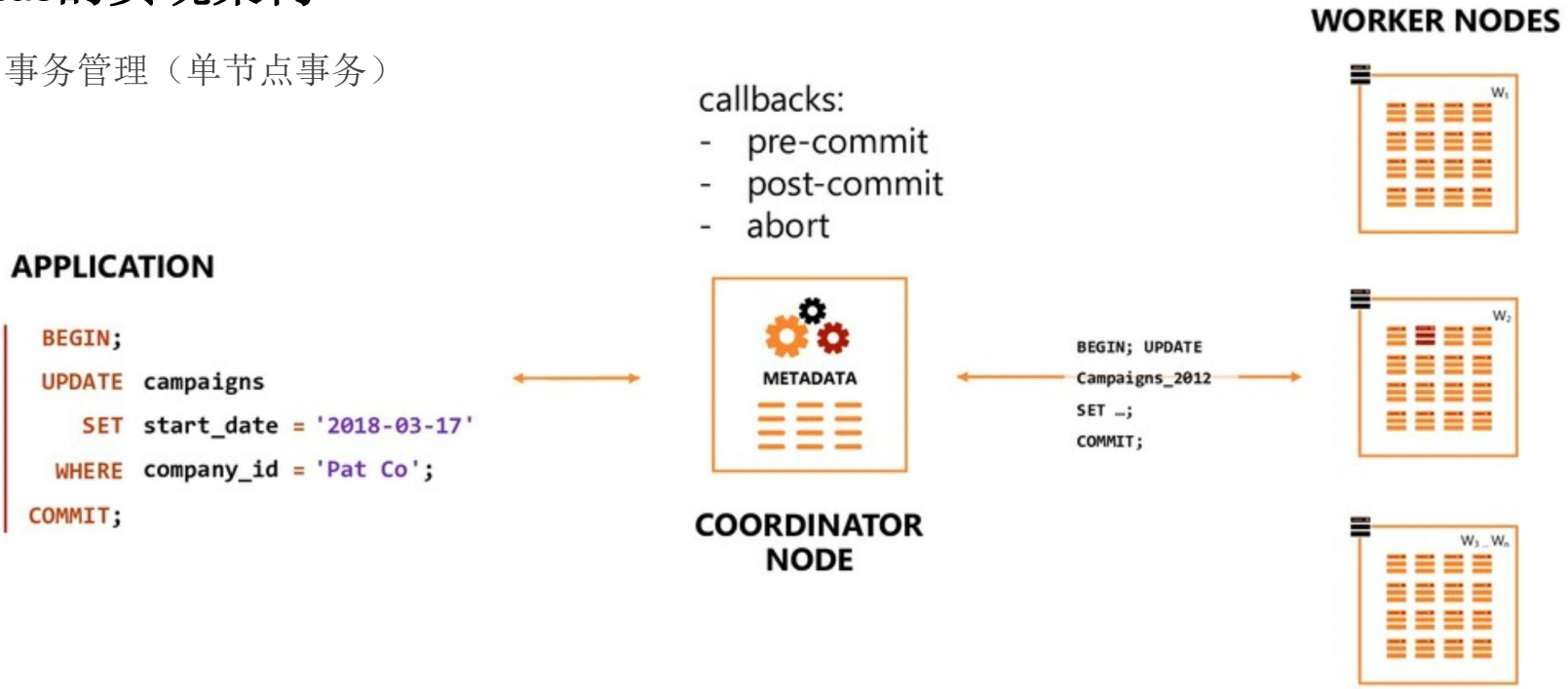






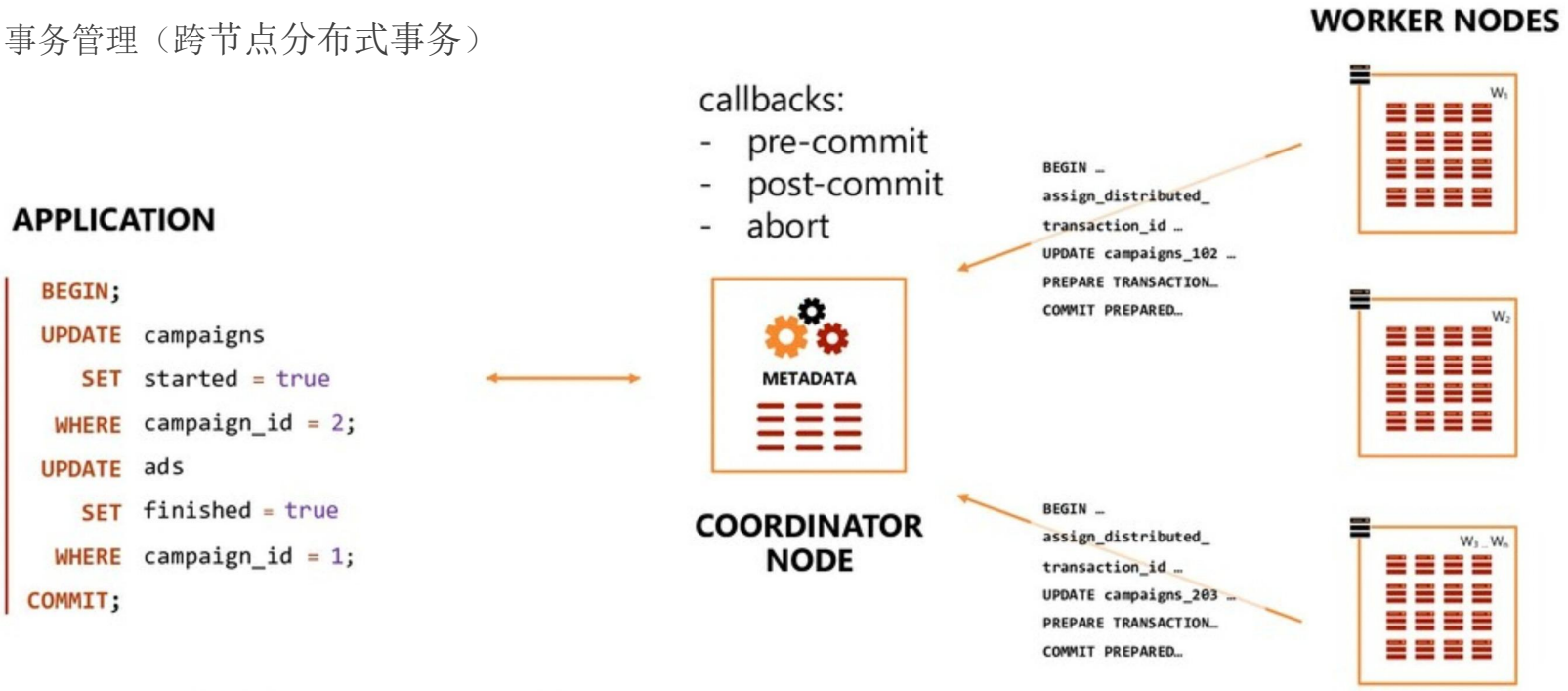
Citus的实现架构

- 事务管理（单节点事务）



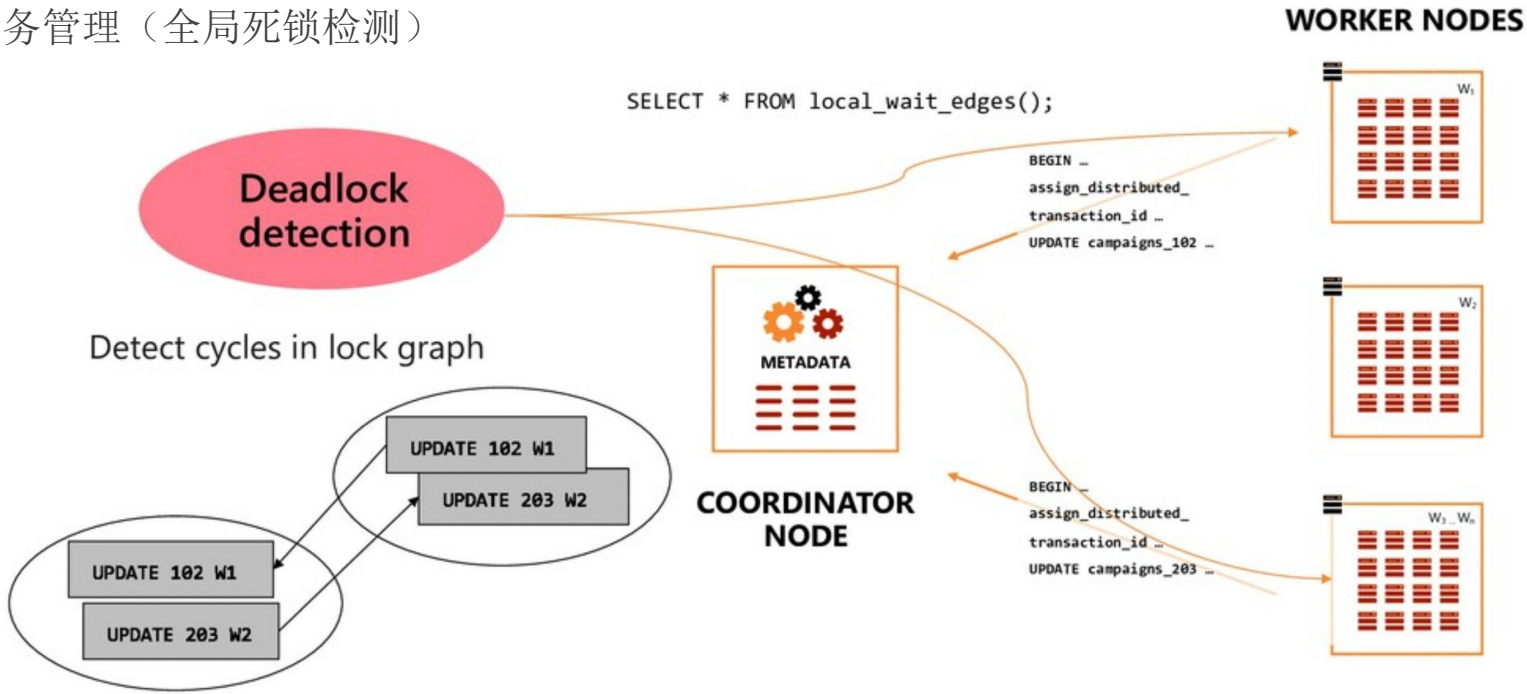
Citus的实现架构

- 事务管理（跨节点分布式事务）



Citus的实现架构

- 事务管理（全局死锁检测）



总结

Citus节点主要分为CN和DN，CN不存储真实数据，只存储数据分布的元信息，实际的数据被分成若干分片，打散到不同DN上，应用连接CN，CN进行SQL解析，生成分布式执行计划，下发到DN执行，CN将结果汇总返回客户端。

- 有两种表类型：分布表和复制表，复制表每个DN节点都有一份完整的副本，分布表则会打散分布到不同DN节点中。
- 可以通过增加多个CN读节点增加集群读的能力，实现读写分离，写CN和读CN之间使用流复制进行元数据同步。
- 支持MX模式，可以将元数据也存在某些DN中，这样使得DN能够直接提供写的能力，增加集群写的能力。
- DN节点之间可以通过流复制来实现数据高可用。
- 没有全局事务管理，故不能保证数据的实时读一致性。数据写一致性使用2PC来保证。

THANKS

谢谢观看