GREENPLUM DATABASE®

阿里云 aliyun.com

Greenplum内核揭秘之执行引擎

钉钉直播 | 5月22日 16：00 - 17：00

GREENPLUM DATABASE

课程大纲

- 执行器介绍
- 并行化PLAN
- Dispatcher
- Ineterconnect

执行器介绍

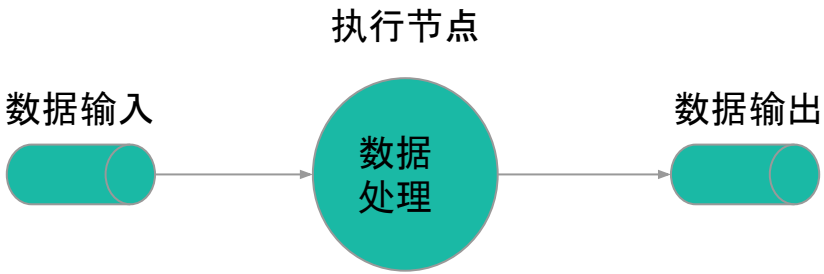
什么是执行器？

GREENPLUM DATABASE®

执行器处理一个由执行计划节点组成的树，并返回查询结果

PlanNode（执行计划节点）

本质上是数据处理



PlanTree（执行计划树）

GREENPLUM DATABASE

```
select t1.c1, t2.* from t1, t2 where t1.c1 =
t2.c2 and t1.c1 > 0 and t2.c2 > 0

Hash Join
   Output: t1.c1, t2.c1, t2.c2
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on public.t1
         Output: t1.c1, t1.c2
         Filter: (t1.c1 > 0)
   ->  Hash
         Output: t2.c1, t2.c2
         ->  Seq Scan on public.t2
               Output: t2.c1, t2.c2
               Filter: (t2.c2 > 0)
```

PlanTree（执行计划树）

GREENPLUM DATABASE

```
select t1.c1, t2.* from t1, t2 where t1.c1 =
t2.c2 and t1.c1 > 0 and t2.c2 > 0

Hash Join
   Output: t1.c1, t2.c1, t2.c2
   Hash Cond: (t1.c1 = t2.c2)
   ->  Seq Scan on public.t1
         Output: t1.c1, t1.c2
         Filter: (t1.c1 > 0)
   ->  Hash
         Output: t2.c1, t2.c2
         ->  Seq Scan on public.t2
               Output: t2.c1, t2.c2
               Filter: (t2.c2 > 0)
```

原发性的扫描节点

PlanTree（执行计划树）

GREENPLUM DATABASE

```
select t1.c1, t2.* from t1, t2 where t1.c1 =
t2.c2 and t1.c1 > 0 and t2.c2 > 0

Hash Join
    Output: t1.c1, t2.c1, t2.c2
    Hash Cond: (t1.c1 = t2.c2)
    ->  Seq Scan on public.t1
          Output: t1.c1, t1.c2
          Filter: (t1.c1 > 0)
    -> Hash
          Output: t2.c1, t2.c2
          ->  Seq Scan on public.t2
                Output: t2.c1, t2.c2
                Filter: (t2.c2 > 0)
```

非原发性扫描节点

执行模型：迭代模型（pipeline模型，Pull方式）

GREENPLUM DATABASE®

每一个执行节点实现一个next函数，并遵循：

1. 每一次调用，返回**一个tuple**或者返回**NULL**。
2. 实现一个循环，每次调执行**子节点**的**next函数**作为输入并处理。

优点：易懂，资源使用少，通用性好
缺点：迭代次数多，代码局部性差，CPU cacheline不友好

执行模型：向量化模型（VECTORIZATION Model）

GREENPLUM DATABASE®

和迭代模型一样，每一个执行节点实现一个next函数，区别在于

每一次迭代，执行节点返回一组tuple，而非一个tuple

优点：减少迭代次数，可以利用新的硬件特性如SIMD。单次更多的tuple对列存更友好（可以利用 压缩特性等）

执行模型：PUSH执行模型

每一个执行节点定义两个函数

produce函数：对原发性扫描节点，该函数名副其实，生产数据，并调用上层节点的consume函数。对非原发性扫描节点，produce函数更像一个控制函数，用于 调用子节点的produce函数，快速定位到数据源 头。

consume函数：被下层节点调用，接收子节点数据进行处理，然后调用父节点的consume函数消费本节点的数据。

Push模型举例

GREENPLUM DATABASE

```
⋈.produce          ⋈.left.produce; ⋈.right.produce;
⋈.consume(a,s)     if (s==⋈.left)
                    print "materialize tuple in hash table";
                   else
                    print "for each match in hashtable["
                       +a.joinattr+"]";
                    ⋈.parent.consume(a+new attributes)
σ.produce          σ.input.produce
σ.consume(a,s)     print "if "+σ.condition;
                    σ.parent.consume(attr,σ)
scan.produce       print "for each tuple in relation"
                   scan.parent.consume(attributes,scan)
```

Push模型的优势

GREENPLUM DATABASE

下层驱动模型相对容易转换成由数据驱动的代码：

```
for each tuple t1 in R1
     materialize t in hash table of HTAB(a=b)
for each tuple t2 in R2
     if t2.b == HTAB(a=b)[t1.a]
          output (t1, t2)
```

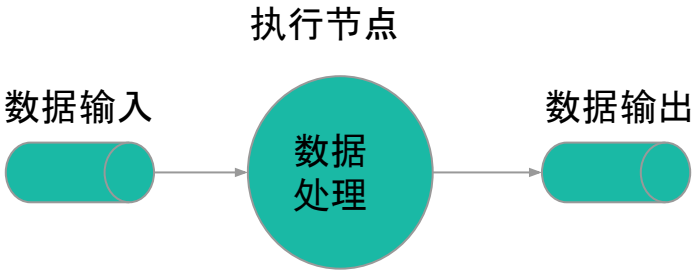
好处一：上层的操作变成本节点的一个算子，增加了代 码的局部性
好处二：这样的代码更方便进一步转换成一个纯计算代码，例如使用LLVM优化。

缺点：个人理解，通用性不 强，可能只能局部 优化。

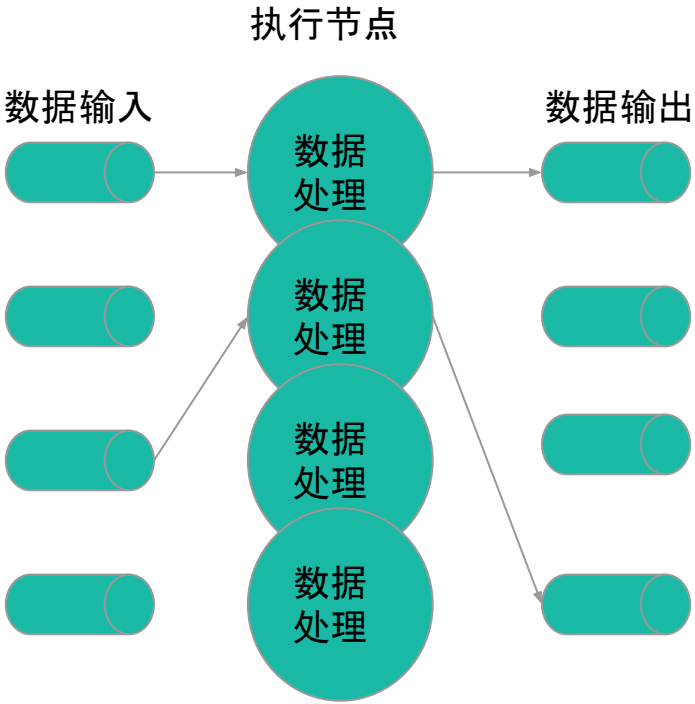
GPDB使用的是迭代模型

但是，GPDB正在积极探索向量化模型和PUSH模型。

GPDB执行器面临更大的挑战

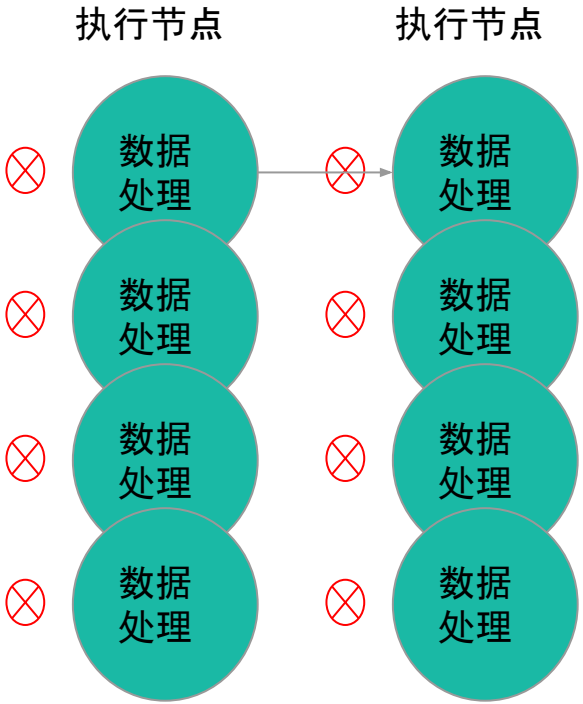


MPP（Massive Parallel Process）



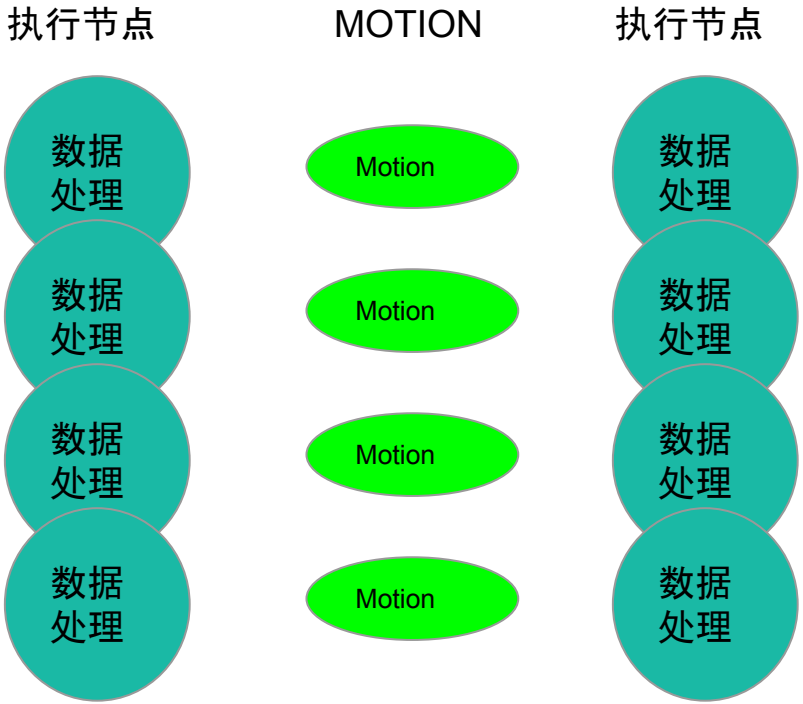
GPDB执行器面临更大的挑战

Shared-Nothing



GPDB的解决方案：

新的执行节点MOTION



包含Motion的执行计划树

GREENPLUM DATABASE

```
select t1.c1, t2.* from t1, t2 where t1.c1 = t2.c2 and t1.c1 > 0 and t2.c2 > 0
;
```

```
Gather Motion 3:1  (slice1; segments: 3)
  ->  Hash Join
        Hash Cond: (t2.c2 = t1.c1)
        ->  Redistribute Motion 3:3  (slice2; segments: 3)
              Hash Key: t2.c2
              ->  Seq Scan on t2
                    Filter: (c2 > 0)
        ->  Hash
              ->  Seq Scan on t1
                    Filter: (c1 > 0)
```

并行化PLAN

并行化PLAN

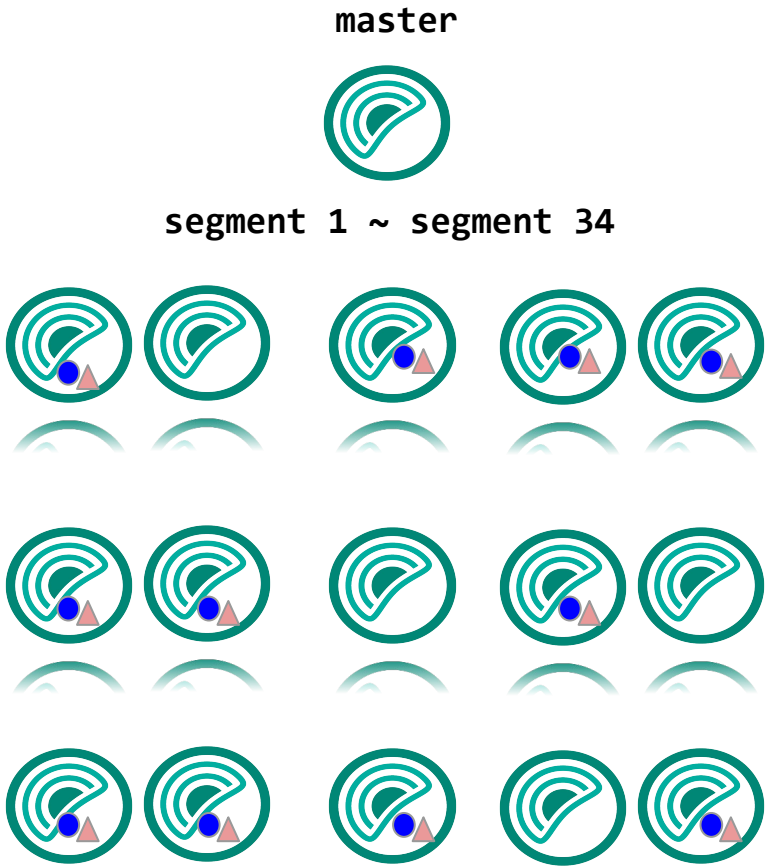


表1:单身男
姓名 | 籍贯 | 年龄 | ...

表2:单身女
姓名 | 籍贯 | 年龄 | ...

查询:

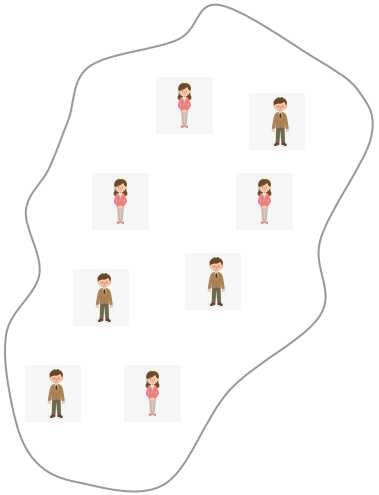
select 相亲(t1, t2) from 单身男 t1, 单身女 t2
where t1.籍贯 = t2.籍贯;

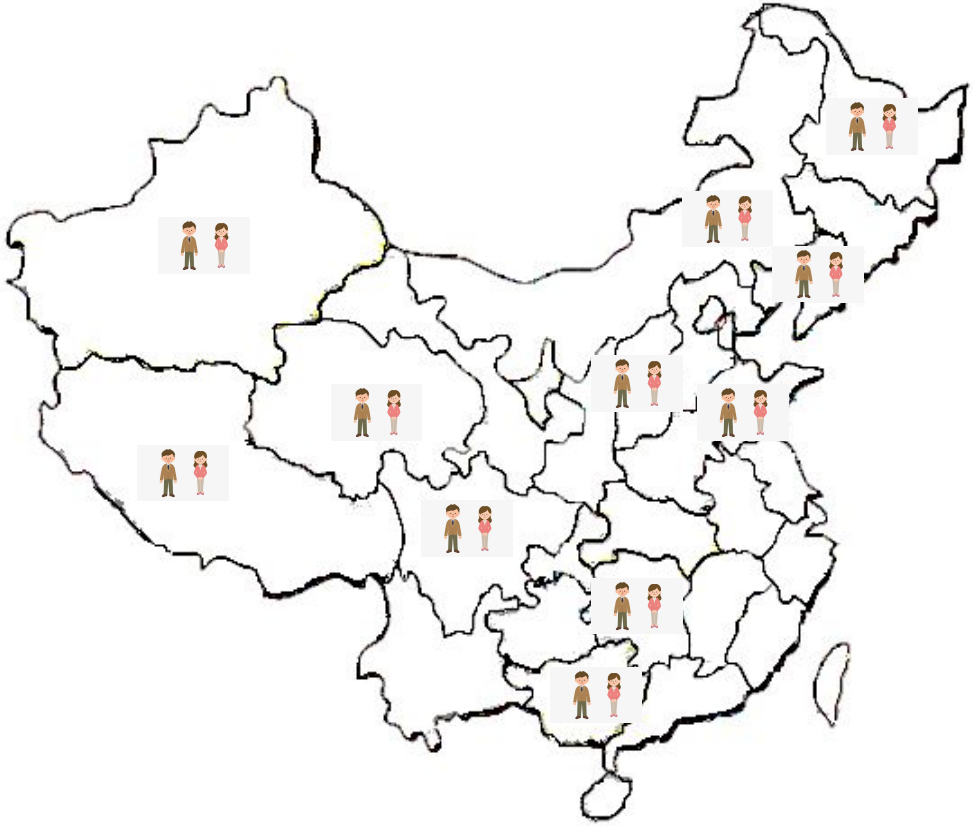
要求:
生成一个可以被并行化 执行的计划树。

现实生活该怎么办？

GREENPLUM DATABASE®

相亲策划：同户籍的适龄单身男女进行配对





Case 1: 单身男女都居住在户籍所在地

GREENPLUM DATABASE®

策略:

各省的分部独自举办相亲会，将本省的适龄单身男女组织到相亲场地进行配对，并将结果返回总部

Case 2：单身女居住在户籍所在地，单身男居住在全国

GREENPLUM DATABASE®

策略1：

各省的分部独自 举办相亲会：

- 首先每个省的单身男青年找出来，**并将它们通过火车派送回原户籍所在地。**
- 然后每个省接待 这些男青年，并在本省找出女 单身青年，对他们进行相亲配对。

策略2：(若单身女生很少)

各省的分部独自 举办相亲会：

- 首先找到本省所有适龄单身女青年，**并为其买好34个省份的车票**，每个省份都去一趟。
- 然后每个省接待这些单身女青年，并安排其与生活在本省的男青年相亲，找出户籍一致的配对。

Case 3：单身男女随机分布在全国各地

GREENPLUM DATABASE®

策略1：

在总部举办相亲会，各省把单身男女通过火车派送回总部，总部接待并安排相亲配对。

策略2：

各分部举办相亲会：

- 首先各省找出居住在本省的适龄单身男，并按户籍派送到相应的省。
- 然后各省找出居住在本省的适龄单身女，并按户籍派送到相应的省。
- 最后各省接待全国归来的男女，进行相亲配对。

相亲策划思路

人多力量大的原则，尽量利有各省的分部

首先分析当前男女青年的地域分布

必要时使用交通工具来打破地域的限制

GPDB采用类似的思想

GREENPLUM DATABASE®

每一张普通表都有数据分布信息：

1. 键值分布
2. 随机分布
3. 复制分布

Locus 信息

GREENPLUM DATABASE

GPDB数据分布的内部表示

CdbLocusType_Entry (访问系统表等)

CdbLocusType_SingleQE (limit等)

CdbLocusType_General (generate_series(1, 10)等)

CdbLocusType_SegmentGeneral (复制表)

CdbLocusType_Hashed (键值表等)

CdbLocusType_Strewn (随即表等)

数据集合都有数据分布状 态, hashjoin后的数据集合也需要有数据分布信息

并行化PLAN

GREENPLUM DATABASE®

通过Motion来打破物理上的隔离：

- Redistribute Motion
- Gather/Gather Merge Motion
- Broadcast Motion
- Explict Redistribute Motion

并行化PLAN

GREENPLUM DATABASE

GPDB优化器会在一些特定的点 进行MOTION评估。

评估点：

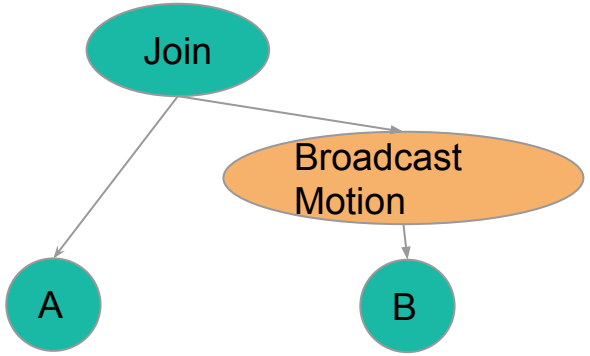
1. 生成新的数据集合 时
   a. join path生成时，参考cdbpath_motion_for_join函数
   b. group path生成时，参考create_grouping_paths函数
   c. subplan的mpp化时，参考cdbllize_decorate_subplans_with_motions函数
   d. ....
2. 对已有数据集合 进行INSERT/UPDATE/DELETE操作时
   参考create_modifytable_path->adjust_modifytable_subpaths

分布式Join

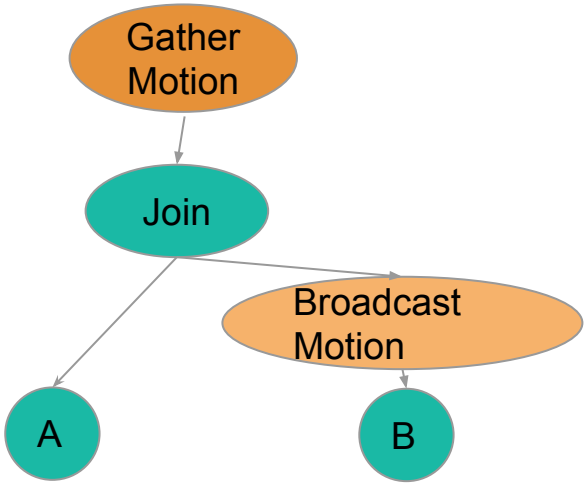
select * from A inner join B;

A：键值表　CdbLocusType_Hashed

B：键值表　CdbLocusType_Hashed



A inner join B: CdbLocusType_Hashed

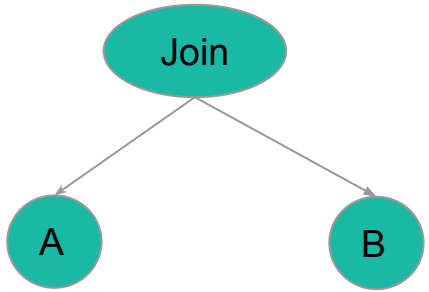


分布式Join

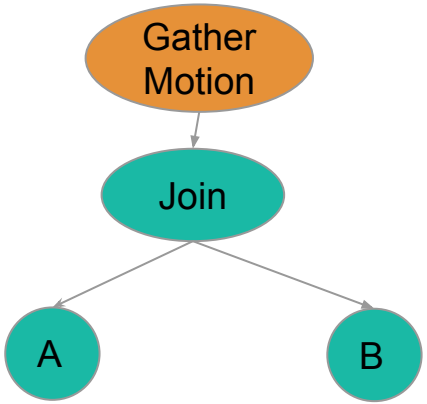
select * from A inner join B;

A：键值表　CdbLocusType_Hashed

B：复制表　CdbLocusType_SegmentGeneral



A inner join B: CdbLocusType_Hashed

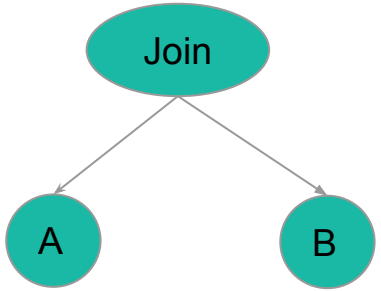


分布式Join

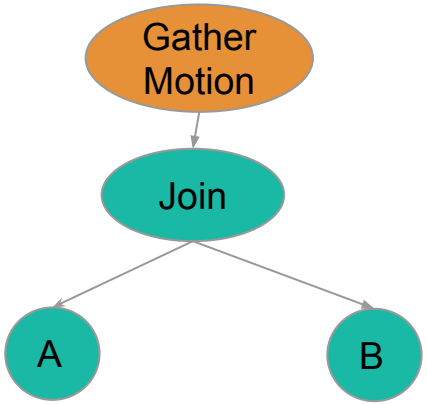
select * from A inner join B;

A：键值表　CdbLocusType_Hashed

B：generate_series(1, 10)　CdbLocusType_General



A inner join B: CdbLocusType_Hashed

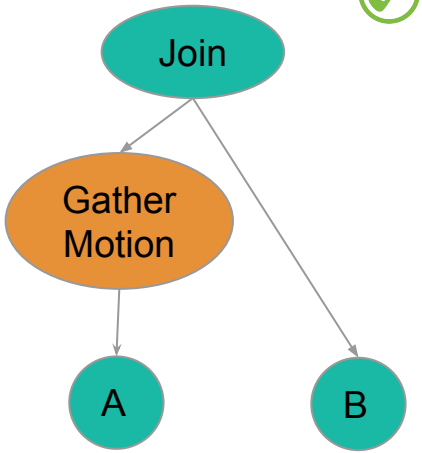


分布式Join

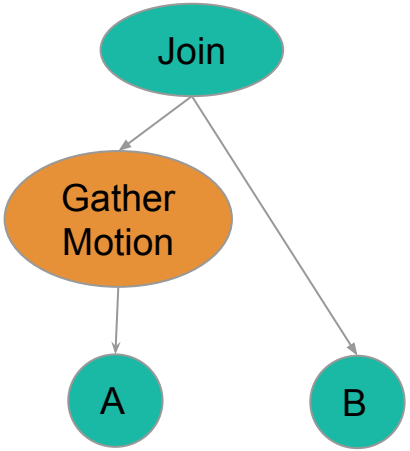
select * from A inner join B;

A：键值表　CdbLocusType_Hashed

B：(select * from A limit 1)　CdbLocusType_SingleQE



A inner join B: CdbLocusType_SingleQE



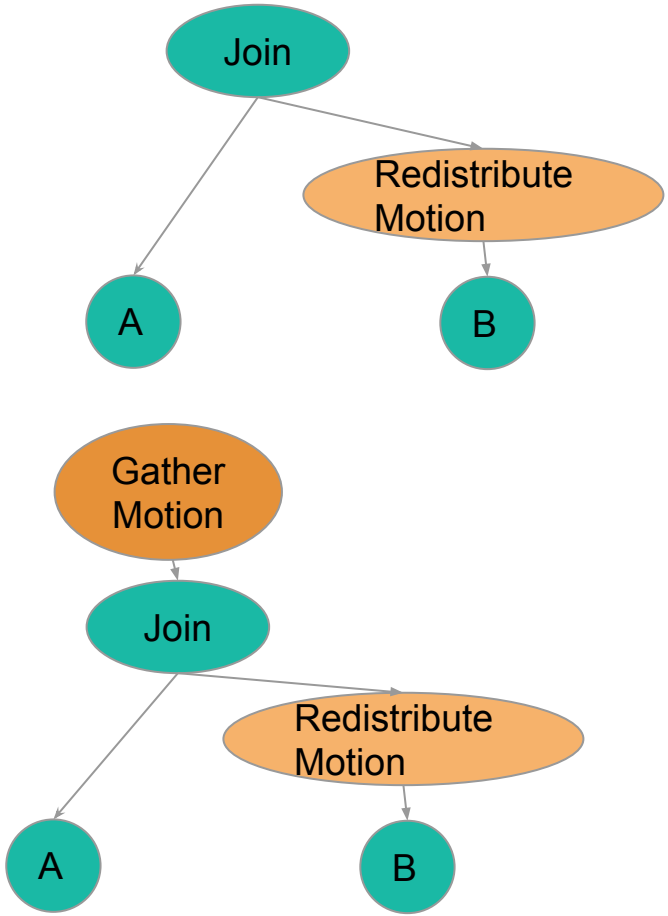
分布式Join

select * from A inner join B where A.c1 = B.c2;

A：键值表　CdbLocusType_Hashed on c1

B：键值表　CdbLocusType_Hashed on c1

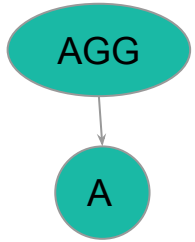
A inner join B: CdbLocusType_Hashed



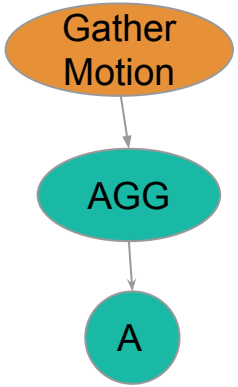
分布式AGG

select count(*) from A group by c1;

A：键值表　CdbLocusType_Hashed on c1



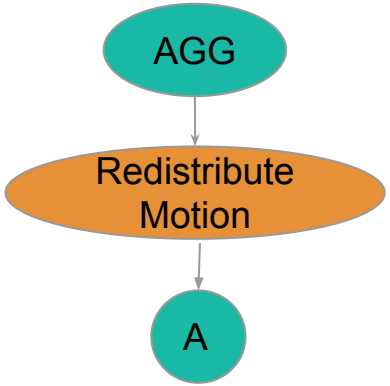
Agg(A): CdbLocusType_Hashed



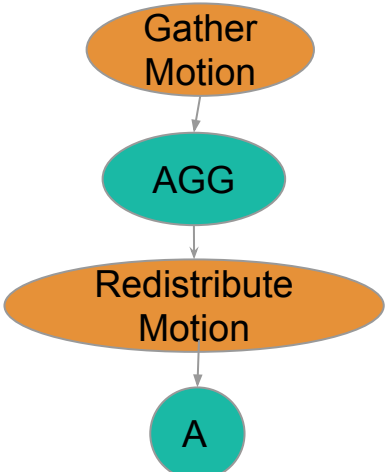
分布式AGG (非两阶段)

select count(*) from A group by c1;

A：键值表　CdbLocusType_Hashed on c2



Agg(A): CdbLocusType_Hashed




Dispatcher

现实生活相亲策划方案怎么实施

GREENPLUM DATABASE®

策略

各省的分部独自举办相亲会:

- 首先每个省的单身男青年找出来，**并将它们通过火车派送回原户籍所在地。**
- 然后每个省接待这些男青年，并在本省找出女单身青年，对他们进行相亲配对。

具体实施/执行步骤

GREENPLUM DATABASE®

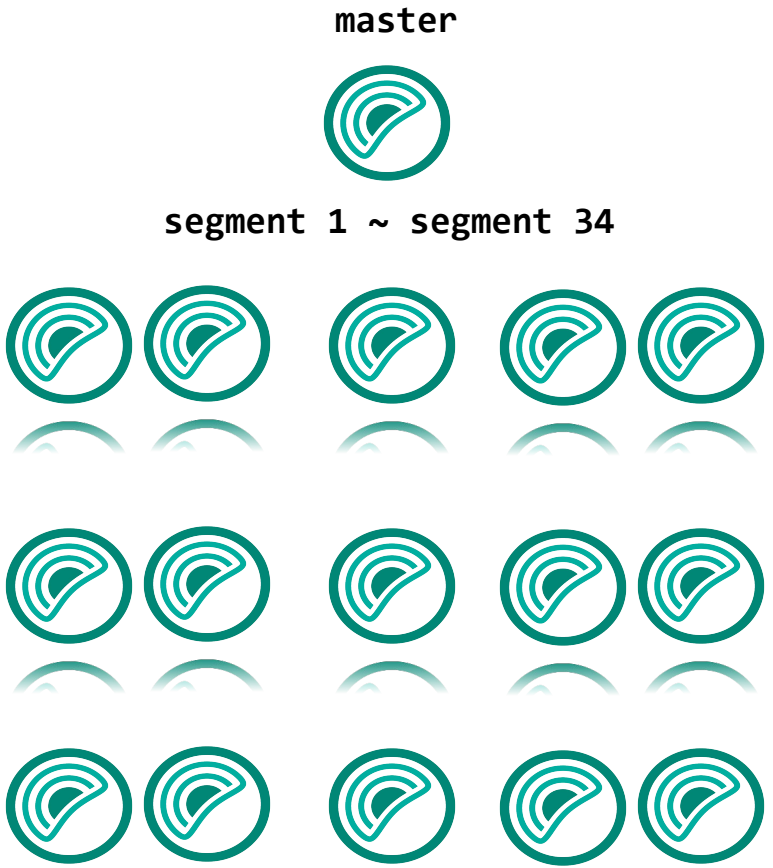
1. 联系各省分部，每省选出两名员工，并上报总部的姓名，联系方式以及接待的火车站。
2. 将上报的所有员工分组：
    a. 组一的人负责找到单身男青年，并买火车票派送到相应的火车站。
    b. 组二的人负责找出单身女青年，并接待全国男青年，举行相亲配对活动并上报结果。
3. 将组一，组二的人员分配以及任务分配邮件发送给所有各省的执行人。
4. 各省执行人按照人员分配及任务分配分别执行任务。
5. 总部收到各省上报的配对结果，相亲会圆满结束。

GPDB执行器面对的问题

有了分布式PLAN，一堆计算资源怎么分配调度和执行起来？

QD：master 提供的计算资源

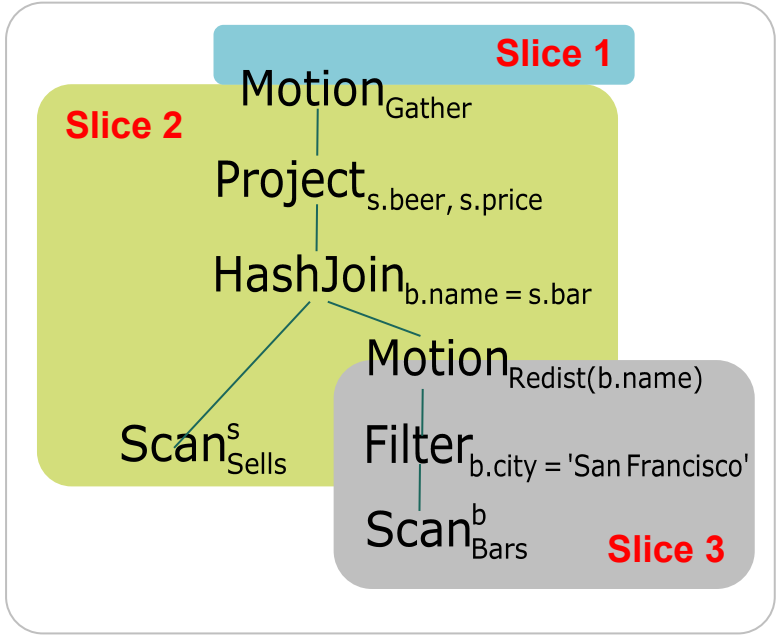
QE：segment提供的计算资源



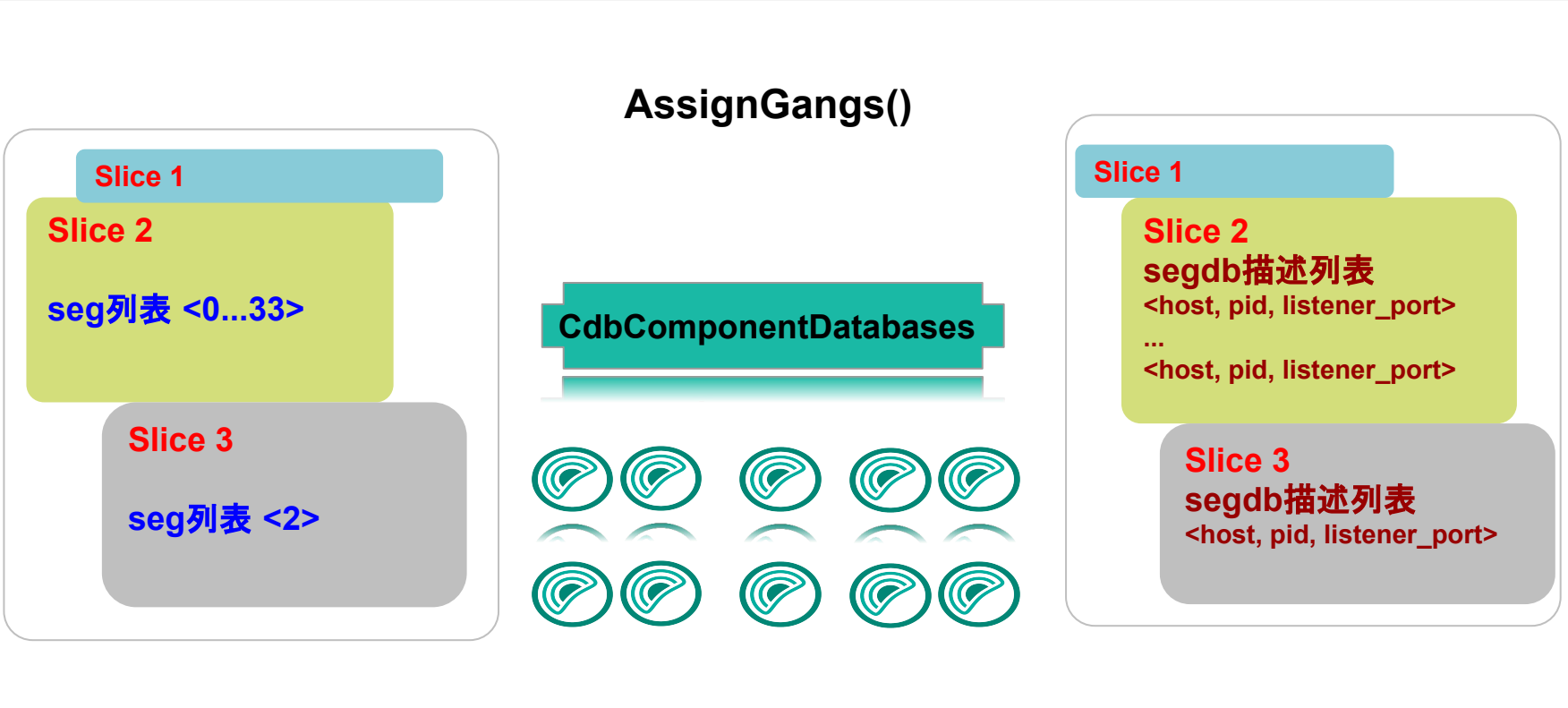
Dispatcher：分配QE资源

GREENPLUM DATABASE

SliceTable



Dispatcher功能一：分配QE资源



Dispatcher：分配QE资源

GREENPLUM DATABASE

`AllocateGang()`

GANG大小分配灵活

最小一个

一般为segment的个数

甚至可以大于segment的个数

(一个segment为一个gang分配多于一个的QE资源)

QE资源闲置以后可以被后续查询重用

(或者闲置一段时间后被清除)

Dispatcher功能: 分发任务

CdbDispatchPlan Plan + SliceTable

CdbDispatchCommand

CdbDispatchDtxProtocolCommand

CdbDispatchUtilityStatement

Dispatcher功能: 协调控制

cdbdisp_checkDispatchResult(等待模式)

等待模式:

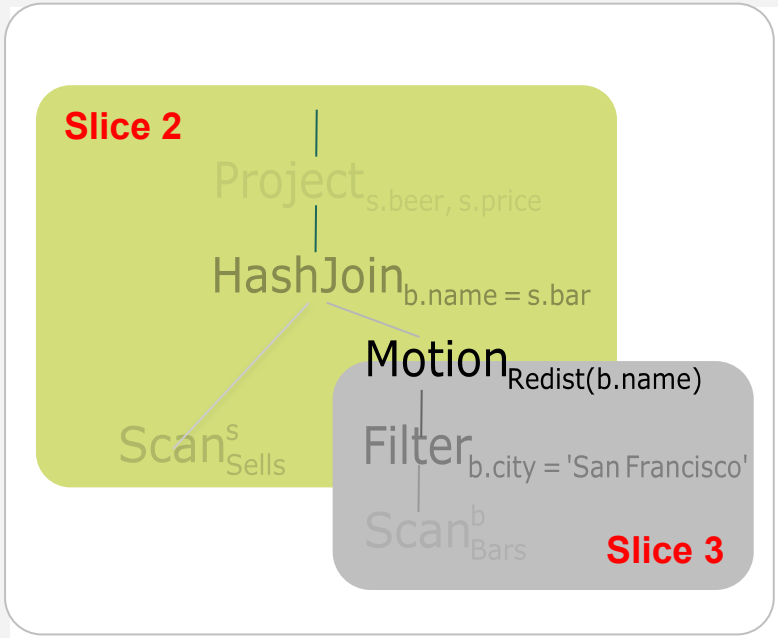
- **Blocking** 阻塞等待所有的QEs完成执行或者出现异常
- **Non-blocking** 检查所有QEs的状态，若QEs有异常则报错，否则立即返回
- **Finish** 给所有活动的QEs发送QueryFinish消息提前结束任务，QE不报错。
- **Cancel** 给所有活动的QEs发送QueryCancel消息，终止执行。

Dispatcher：典型的Dispatcher程序

```
ds = cdbdisp_makeDispatcherState

primaryGang = AllocateGang(ds, GANGTYPE_PRIMARY_WRITER, segments);

cdbdisp_dispatchToGang(ds, primaryGang, -1);

cdbdisp_waitDispatchFinish(ds);

cdbdisp_checkDispatchResult(ds, DISPATCH_WAIT_NONE);

cdbdisp_getDispatchResults(ds, &qeError);

cdbdisp_destroyDispatcherState(ds);
```


Interconnect

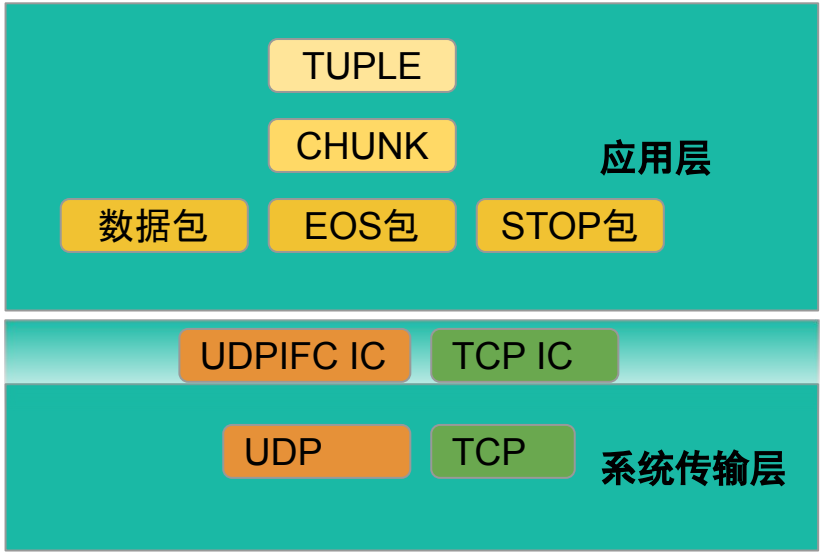
Interconnect



Motion的内部实现是Interconnect

sender和receiver之间通过网络在QE之间移动数据，在GPDB中，该网络模块叫做**Interconnect**

Interconnect Layout



UDPIFC

GPDB自己实现的一种RUDP(Reliable User Datagram Protocol)协议

基于UDP协议，为了支持传输可靠性，实现了重传，乱序处理，不匹配处理，流量控制等功能。

GPDB当初引入UDPIFC主要为了解决复杂OLAP查询在大集群中使用连接资源过多的问题

UDPIFC：线程模型

为什么使用线程模型？

UDPIFC在应用层保证传输的可靠性，需要单独的线程来保证传输可靠协议。

QE在fork的时候会启动一个udpifc线程，该线程将服务该session所有将要可能执行的查询。

udpifc线程接受所有发送给该QE的数据包并通过共享内存移交给主进程。

线程细节可参考rxThreadFunc函数

接收端逻辑可参考RecvTupleChunkFrom*函数

发送端逻辑可参考SendChunkUDPIFC函数

可能会有新的Interconnect类型

QUIC协议

Proxy协议

GREENPLUM DATABASE®

扫码加入Greenplum技术讨论群

微信群：gp_assistant　　钉钉群：https://dwz.cn/23XPHVOD　　QQ群：99194625

欢迎访问Greenplum问答平台 ask.greenplum.cn


Q&A