从查询计划入手优化数据库

郭峰
Pivotal 中国研发中心

© Copyright 2018 Pivotal Software, Inc. All rights Reserved. Version 1.0

Agenda

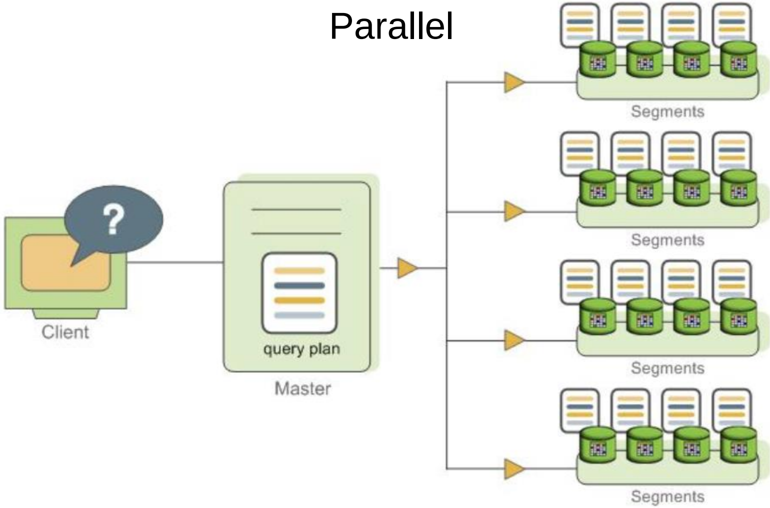
- Greenplum 查询处理过程
- Greenplum 查询计划解读
- Greenplum **分布式**查询计划生成
- Greenplum 查询调优
- Q+A

Greenplum 查询处理过程

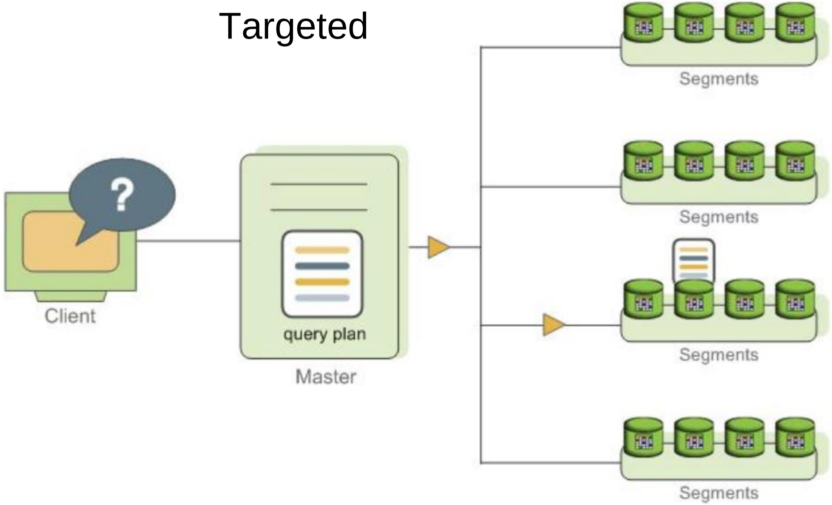
Greenplum 查询处理过程

Greenplum 对查询的处理过程可以分成四个步骤:

1. Master **接收**查询语句并生成查询计划。
2. Master **把**查询计划分发到Segments.



Parallel



Targeted

Greenplum 查询处理过程

Greenplum 对查询的处理过程可以分成四个步骤:

1. Master **接收**查询语句并生成查询计划。

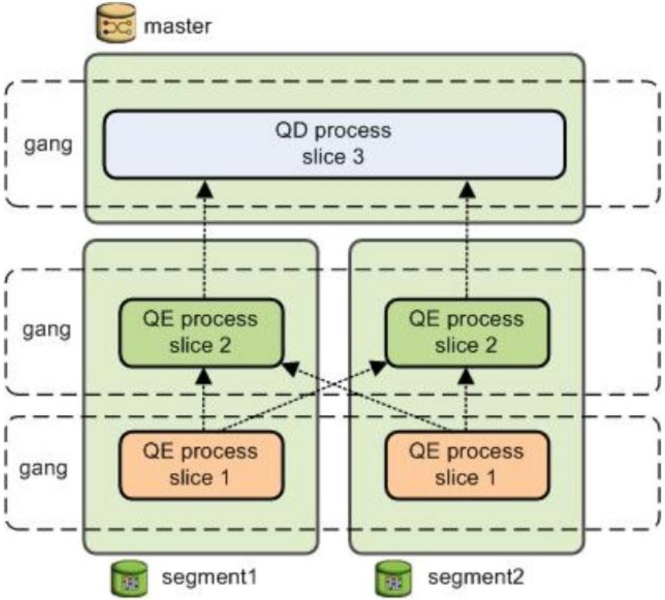
2. Master **把**查询计划分发到Segments.

3. Segments **并**发在各自本地的数据集上执行计划。

   a. Slice: A portion of the plan that segments can work on independently.

   a. Gang: Related processes that are working on the same slice of the query plan but on different segments.

1. Master **收集**结果并返回给客户端。



Greenplum 查询计划解读

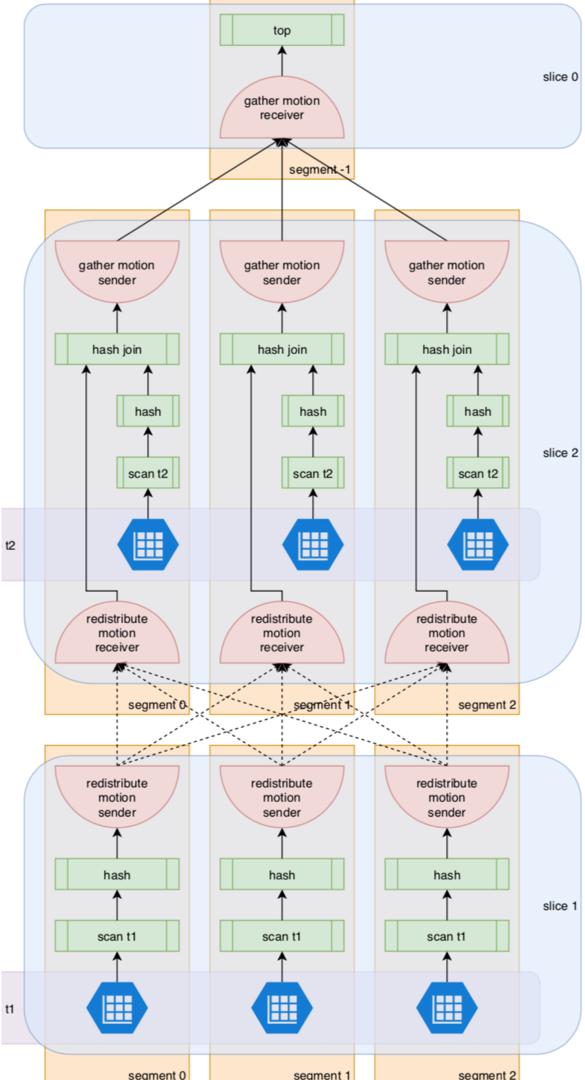
Greenplum 查询计划解读

```
gpadmin=# EXPLAIN select * from t2 join t1 on t2.c1 = t1.c2;

                                             QUERY PLAN
------------------------------------------------------------------------------------
-----------------
 Gather Motion 3:1  (slice2; segments: 3)  (cost=2037.25..97600.62 rows=7413210
width=16)
   ->  Hash Join  (cost=2037.25..97600.62 rows=2471070 width=16)
         Hash Cond: t1.c2 = t2.c1
         ->  Redistribute Motion 3:3  (slice1; segments: 3)  (cost=0.00..2683.00
rows=28700 width=8)
               Hash Key: t1.c2
               ->  Seq Scan on t1  (cost=0.00..961.00 rows=28700 width=8)
         ->  Hash  (cost=961.00..961.00 rows=28700 width=8)
               ->  Seq Scan on t2  (cost=0.00..961.00 rows=28700 width=8)
 Optimizer status: legacy query optimizer
(9 rows)
```

6个节点，其中2个Motion节点。

```
select * from t2 join
t1 on t2.c1 = t1.c2
```



Motion: 在Segments之间传输数据

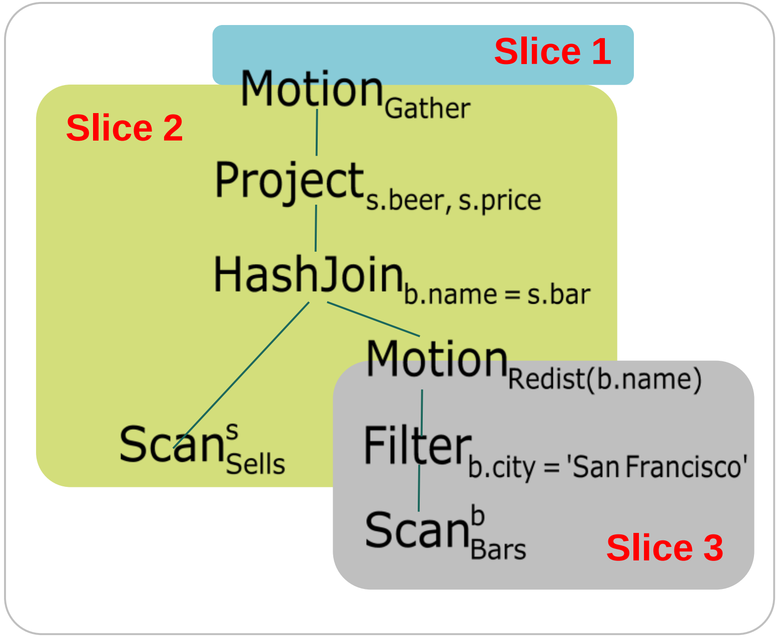
Slice: 根据Motion切割slice, motion的两边各有一个slice.

Greenplum 查询计划解读

```
SELECT s.beer, s.price
FROM Bars b, Sells s
WHERE b.name = s.bar
AND b.city = 'San Francisco'
```

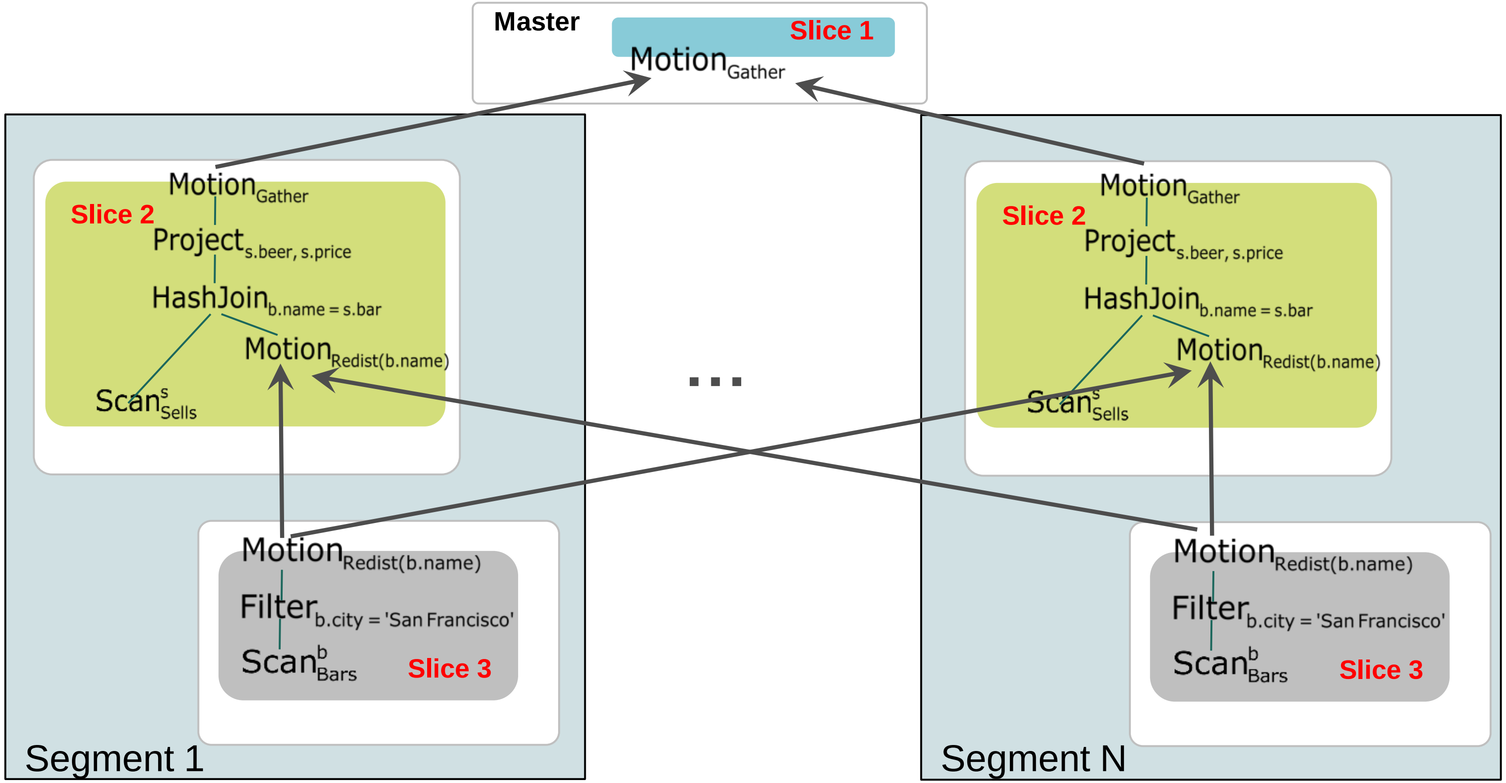
Bars is distributed randomly

Sells is distributed by bar



Pivotal

Greenplum 查询计划解读



Pivotal

Greenplum 查询计划解读

每个节点含有三个估计值：cost, rows, width.

- Cost: Greenplum采用基于代价的优化器来评估执行查询的不同策略，并选择代价最低的方法。
    - 启动代价: 得到第一个tuple的代价。
    - 总代价： 处理完所有tuple的代价。
- Rows: 查询节点输出的tuple的个数 （估计值）。
- Width: 查询节点输出的tuple的长度（估计值），单位是字节。

Pivotal

Greenplum 查询计划解读

EXPLAIN ANALYZE

- 执行查询的总时间（以毫秒为单位）
- **内存使用量**；
- 查询节点需要的工作进程个数
- **每个操作中**处理最多行的Segment处理的最大行数以及Segment**的序号**
- **从**处理最多行的Segment**上**获得第一行花费的时间（单位是毫秒）及从该Segment获**得所有行花**费的时间。

*EXPLAIN ANALYZE **除了**显示查询计划外还会执行查询，若需对DML语句使用EXPLAIN ANALYZE **而不影响数据，　可置** EXPLAIN ANALYZE **于事**务中：*

```
BEGIN;
EXPLAIN ANALYZE ...;
ROLLBACK;
```

```
select * from t2 join t1 on
t2.c1 = t1.c2
```

```
 Gather Motion 3:1  (slice2; segments: 3)  (cost=2360.80..7000.90 rows=99992 width=16)
   Rows out:  33333 rows at destination with 26 ms to first row, 93 ms to end, start offset by 0.530 ms.
   ->  Hash Join  (cost=2360.80..7000.90 rows=33331 width=16)
         Hash Cond: t1.c2 = t2.c1
         Rows out:  Avg 11111.0 rows x 3 workers.  Max 11246 rows (seg1) with 24 ms to first row, 83 ms to end, start offset by 0.821 ms.
         Executor memory:  1042K bytes avg, 1043K bytes max (seg0).
         Work_mem used:  1042K bytes avg, 1043K bytes max (seg0). Workfile: (0 spilling)
         (seg1)   Hash chain length 2.3 avg, 9 max, using 14237 of 16384 buckets.
         ->  Redistribute Motion 3:3  (slice1; segments: 3)  (cost=0.00..3137.97 rows=33633 width=8)
               Hash Key: t1.c2
               Rows out:  Avg 33333.3 rows x 3 workers at destination.  Max 33468 rows (seg0) with 0.433 ms to first row, 40 ms to end, start o
ffset by 25 ms.
               ->  Seq Scan on t1  (cost=0.00..1119.99 rows=33633 width=8)
                     Rows out:  Avg 33333.3 rows x 3 workers.  Max 33348 rows (seg0) with 0.048 ms to first row, 4.813 ms to end, start offset
by 0.991 ms.
         ->  Hash  (cost=1110.91..1110.91 rows=33331 width=8)
               Rows in:  Avg 33333.3 rows x 3 workers.  Max 33348 rows (seg0) with 24 ms to end, start offset by 0.904 ms.
               ->  Seq Scan on t2  (cost=0.00..1110.91 rows=33331 width=8)
                     Rows out:  Avg 33333.3 rows x 3 workers.  Max 33348 rows (seg0) with 0.045 ms to first row, 4.892 ms to end, start offset
by 0.905 ms.
 Slice statistics:
   (slice0)    Executor memory: 386K bytes.
   (slice1)    Executor memory: 212K bytes avg x 3 workers, 212K bytes max (seg0).
   (slice2)    Executor memory: 4767K bytes avg x 3 workers, 4767K bytes max (seg0).  Work_mem: 1043K bytes max.
 Statement statistics:
   Memory used: 4096K bytes
 Optimizer status: legacy query optimizer
 Total runtime: 96.875 ms
(25 rows)
```

Greenplum 查询计划解读

with T1 ms to first row, T2 ms to end, start offset by T3 ms

- T1: 这个查询节点上得到第一个tuple的执行时间
- T2: 这个查询节点上得到所有tuple的总执行时间
- T3: 这个查询节点执行开始时间 **减去** 这个查询的开始时间
  - Interconnect的建立
  - 分发查询计划
  - 创建需要的gang

Greenplum 查询计划解读

Work_mem used

- 这个查询节点执行时用到的内存大小。
- **如果分配**给这个查询节点的内存量不足够用来执行操作，部分数据将溢出的文件中。

```
 Work_mem used:  68K bytes avg, 69K bytes max (seg0). Workfile: (3 spilling)
 Work_mem wanted: 1042K bytes avg, 1043K bytes max (seg0) to lessen workfile I/O affecting 3
workers.
 (seg0)   Initial batch 0:
 (seg0)     Wrote 480K bytes to inner workfile.
 (seg0)     Wrote 480K bytes to outer workfile.
 (seg0)   Initial batches 1..15:
 (seg0)     Read 611K bytes from inner workfile: 41K avg x 15 nonempty batches, 43K max.
 (seg0)     Read 613K bytes from outer workfile: 41K avg x 15 nonempty batches, 43K max.
 (seg0)   Hash chain length 2.3 avg, 10 max, using 14298 of 16384 buckets.
```

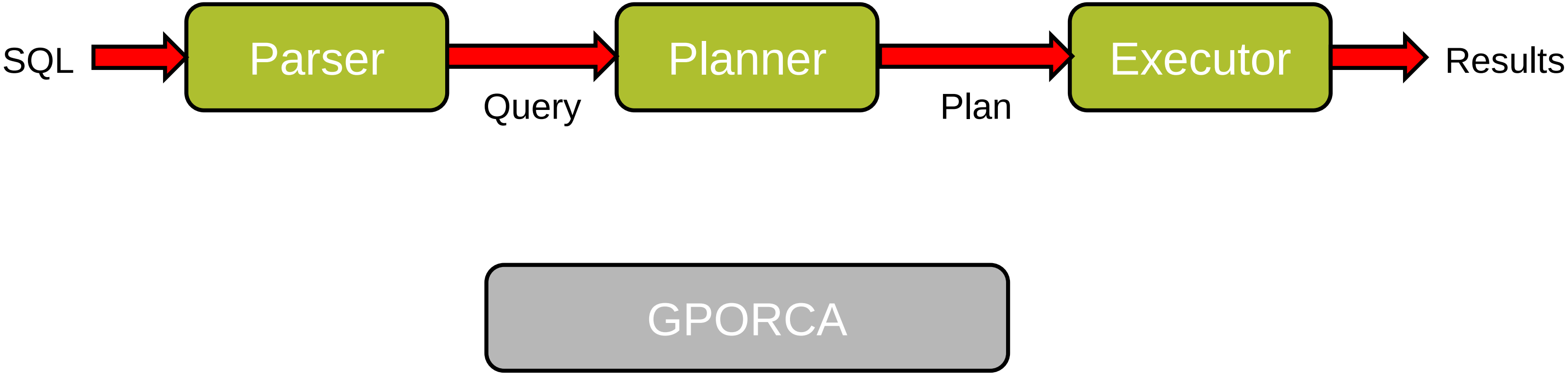
Greenplum 分布式查询计划生成

Pivotal

Greenplum 分布式查询计划生成

Greenplum支持两种查询优化器：Planner 和 GPORCA

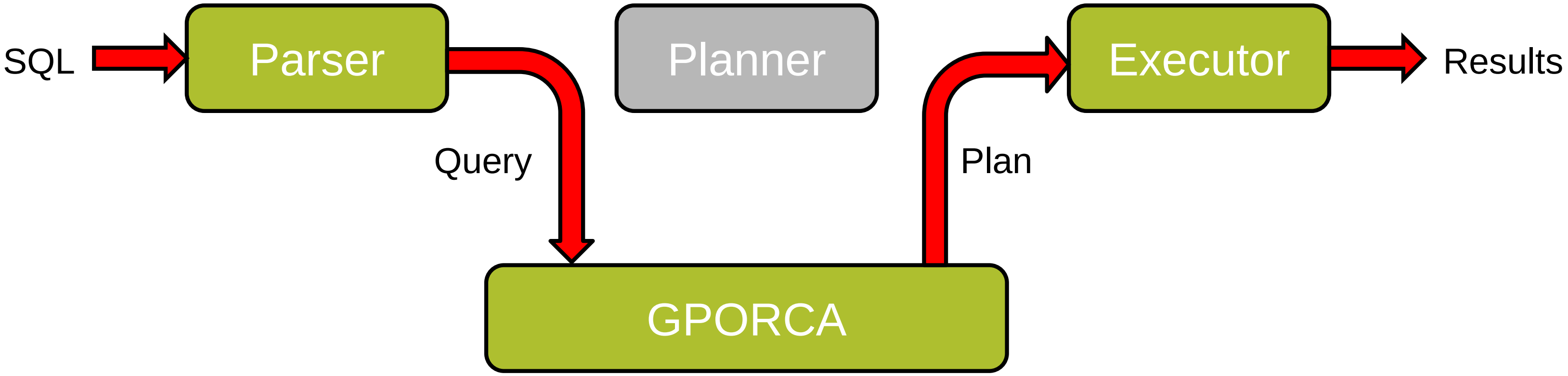
使用Planner:



Greenplum 分布式查询计划生成

Greenplum支持两种查询优化器：Planner 和 GPORCA

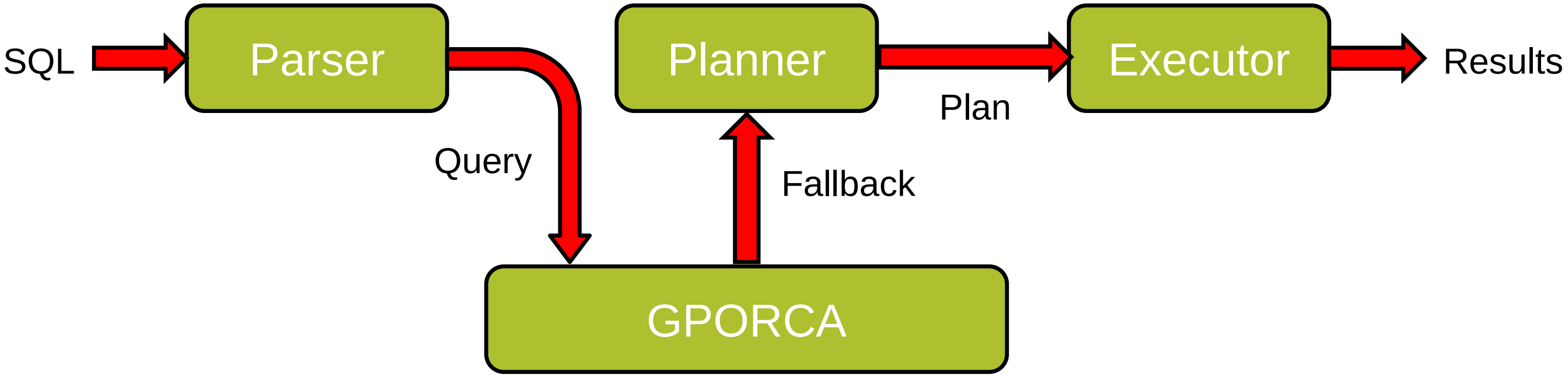
使用GPORCA:



Greenplum 分布式查询计划生成

Greenplum支持两种查询优化器：Planner 和 GPORCA

对于GPORCA不支持的特性，GPORCA会自动fall back到Planner:



GPORCA or Planner?

```
bootcamp=# explain select * from products;
                                  QUERY PLAN
-----------------------------------------------------------------------------
 Gather Motion 2:1  (slice1; segments: 2)  (cost=0.00..431.00 rows=1 width=11)
   ->  Table Scan on products  (cost=0.00..431.00 rows=1 width=11)
 Settings:  optimizer=on
 Optimizer status: PQO version 1.628
(4 rows)
```

PQO: Pivotal Query Optimizer
+
Optimizer Version #

GPORCA or Planner?

```
bootcamp=# explain select * from products;
                                QUERY PLAN
---------------------------------------------------------------------------
 Gather Motion 2:1  (slice1; segments: 2)  (cost=0.00..1.01 rows=1 width=11)
   ->  Seq Scan on products  (cost=0.00..1.01 rows=1 width=11)
 Optimizer status: legacy query optimizer
(3 rows)
```

Planner Plan

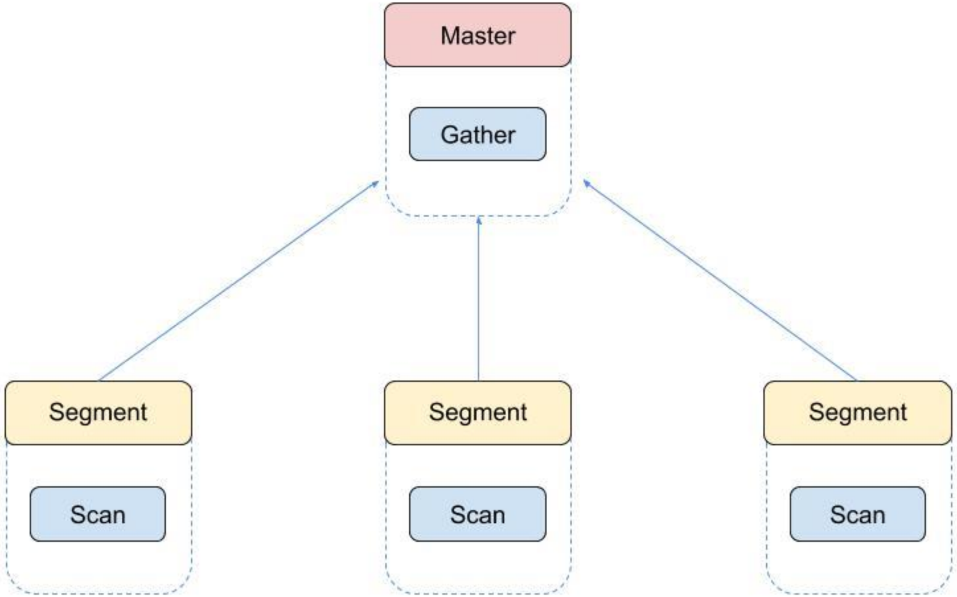
Greenplum 分布式查询计划生成

```
create table t1(c1 int, c2 int, c3 int) distributed by (c1);

create table t2(c1 int, c2 int, c3 int) distributed by (c1);
```

Greenplum 分布式查询计划生成: Scan

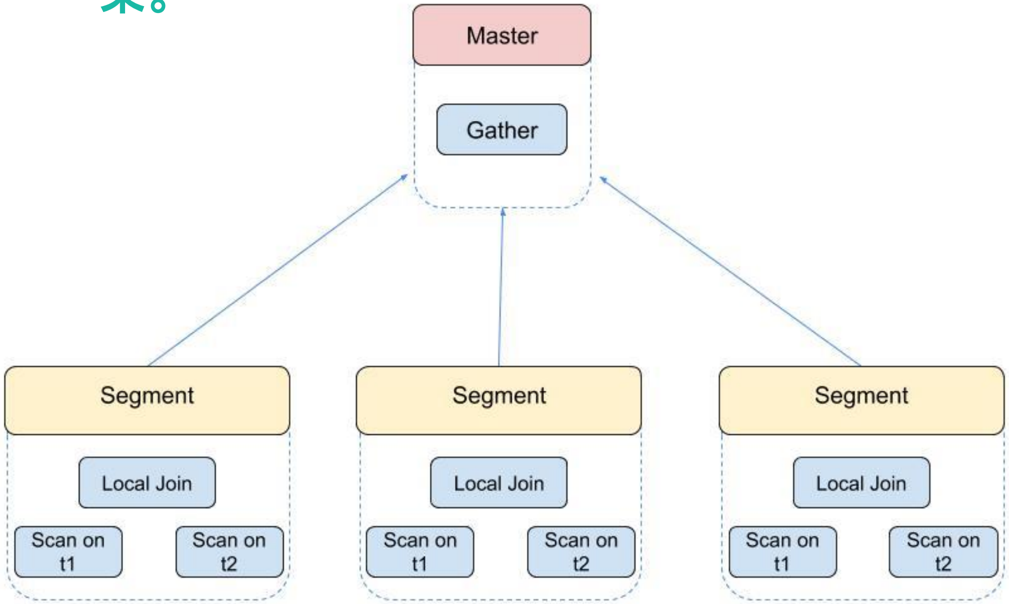
对于Scan节点，每个segment扫描其本地数据，最后master通过Gather Motion来收集结果。



```
SELECT * from t1 where t1.c2 = 1;
        QUERY PLAN
--------------------------
 Gather Motion 3:1
   ->  Seq Scan on t1
         Filter: (c2 = 1)
(3 rows)
```

Greenplum 分布式查询计划生成: Join

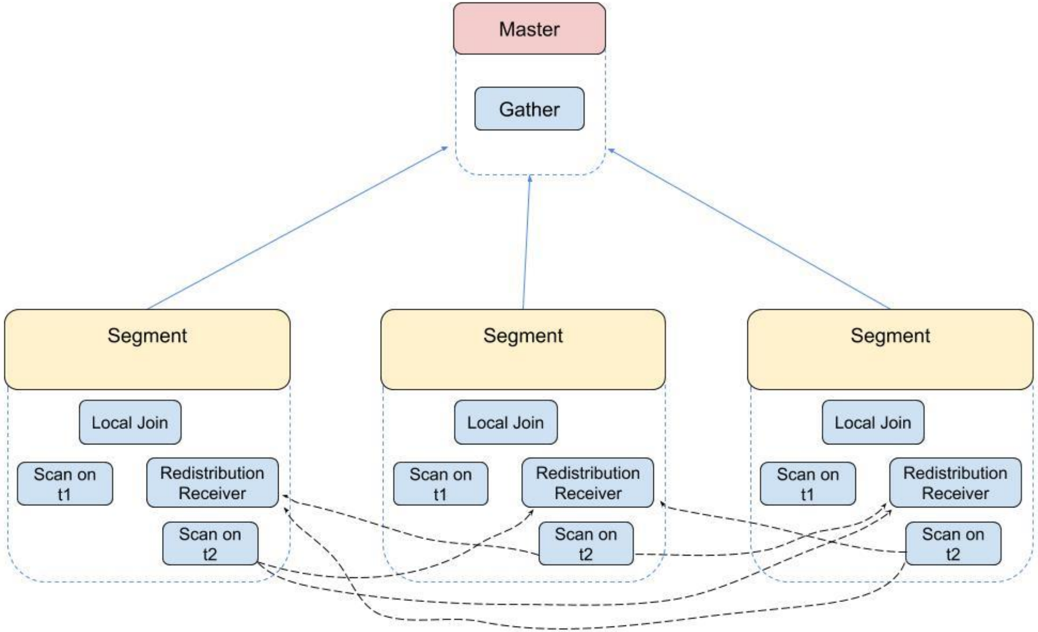
对于Join节点，如果我们执行的是基于分布键的等值连接，那么每个segment可以执行本地连接，最后master通过Gather Motion来收集结果。



```
SELECT * from t1, t2 where t1.c1 = t2.c1;
             QUERY PLAN
------------------------------------
 Gather Motion 3:1
   ->  Hash Join
         Hash Cond: (t1.c1 = t2.c1)
         ->  Seq Scan on t1
         ->  Hash
               ->  Seq Scan on t2
(6 rows)
```

Greenplum 分布式查询计划生成: Join

否则，我们可能会需要重分布其中一个表。

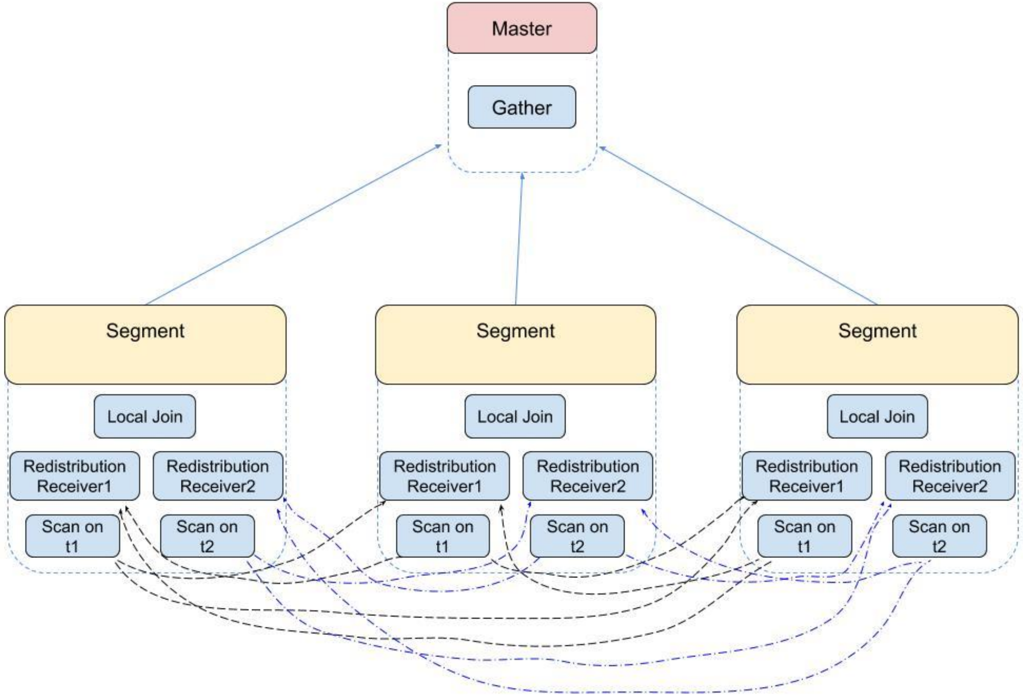


Redistributing t2 by c2

```
SELECT * from t1, t2 where t1.c1 = t2.c2;
                  QUERY PLAN
-------------------------------------------
 Gather Motion 3:1
   ->  Hash Join
         Hash Cond: (t1.c1 = t2.c2)
         ->  Seq Scan on t1
         ->  Hash
               ->  Redistribute Motion 3:3
                     Hash Key: t2.c2
                     ->  Seq Scan on t2
(8 rows)
```

Greenplum 分布式查询计划生成: Join

或重分布两个表。



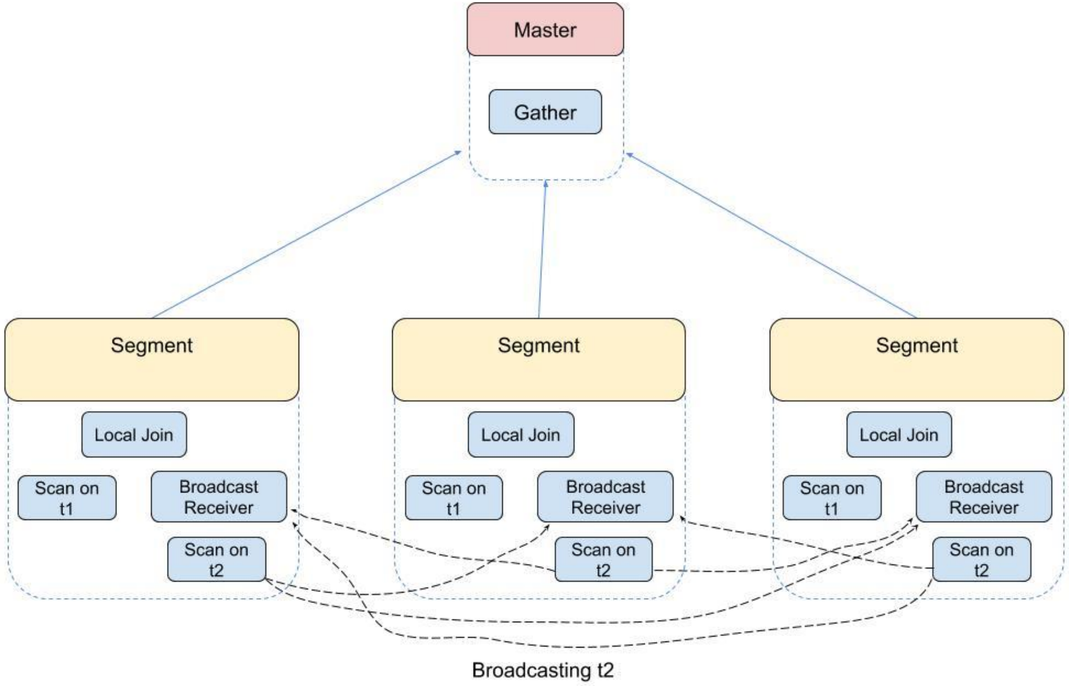
Redistributing t1 and t2 by c2

```
SELECT * from t1, t2 where t1.c2 = t2.c2;
                QUERY PLAN
-------------------------------------------
 Gather Motion 3:1
   ->  Hash Join
         Hash Cond: (t1.c2 = t2.c2)
         ->  Redistribute Motion 3:3
               Hash Key: t1.c2
               ->  Seq Scan on t1
         ->  Hash
               ->  Redistribute Motion 3:3
                     Hash Key: t2.c2
                     ->  Seq Scan on t2
(10 rows)
```

Pivotal

Greenplum 分布式查询计划生成: Join

或者广播其中一个表。



```
SELECT * from t1, t2 where t1.c2 = t2.c2;
                QUERY PLAN
----------------------------------------
 Gather Motion 3:1
   ->  Hash Join
         Hash Cond: (t1.c2 = t2.c2)
         ->  Seq Scan on t1
         ->  Hash
               ->  Broadcast Motion 3:3
                     ->  Seq Scan on t2
(7 rows)
```

Pivotal

Greenplum 分布式查询计划生成: Aggregate without Group

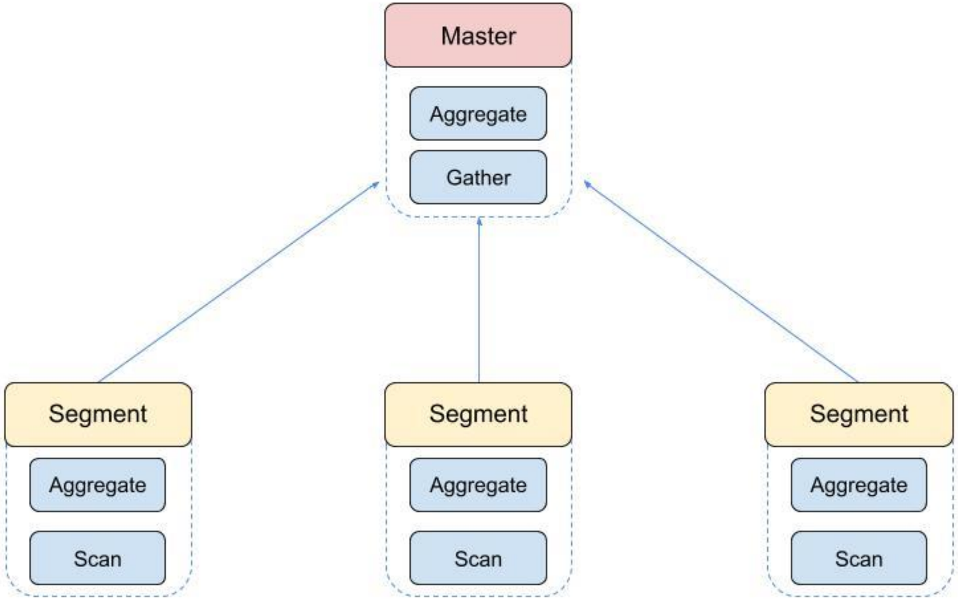
对于没有分组的聚集，Greenplum是通过两阶段聚集来完成的。

第一阶段：每个segment执行一次本地聚集。

第二阶段：Master通过Gather Motion收集第一阶段聚集的结果，然后执行第二阶段聚集。

Greenplum 分布式查询计划生成: Aggregate without Group

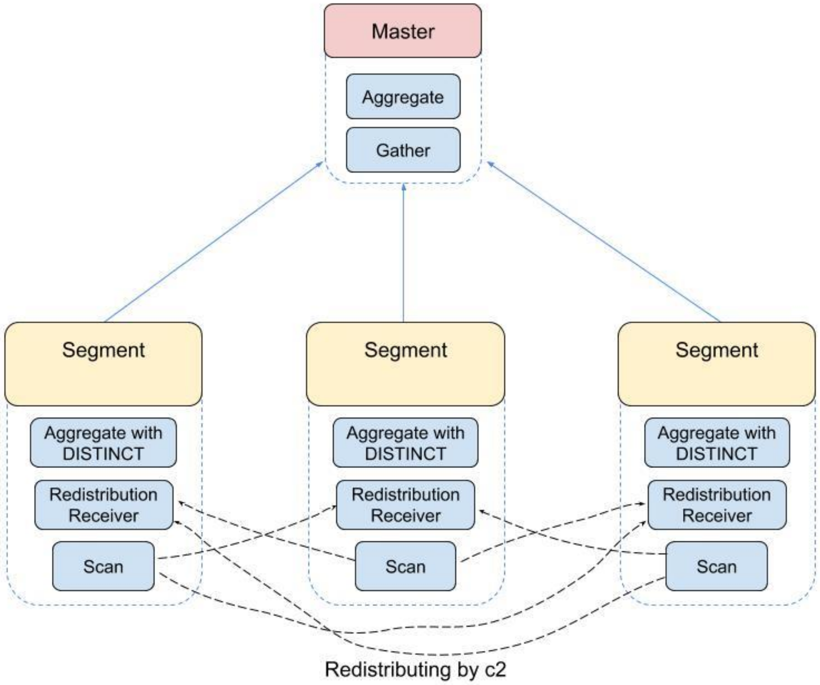
如果没有DISTINCT，或者DISTINCT键是分布键，那么可以直接执行两阶段聚集



```
SELECT avg(c1) from t1;
            QUERY PLAN
----------------------------------
 Aggregate
   ->  Gather Motion 3:1
         ->  Aggregate
               ->  Seq Scan on t1
(4 rows)
```

Greenplum 分布式查询计划生成: Aggregate without Group

否则，需要先重分布数据，然后再执行两阶段聚集。



```
SELECT avg(distinct c2) from t1;
                QUERY PLAN
-------------------------------------------
 Aggregate
   ->  Gather Motion 3:1
         ->  Aggregate
               ->  Redistribute Motion 3:3
                 Hash Key: t1.c2
                     ->  Seq Scan on t1
(6 rows)
```

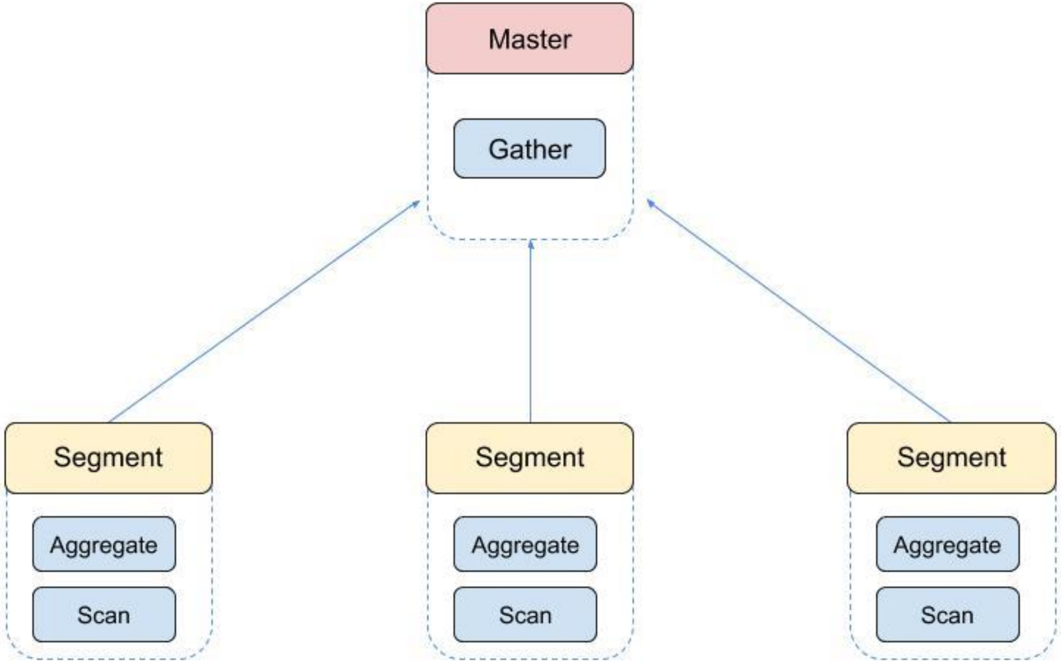
Greenplum 分布式查询计划生成: Aggregate with Group

对于有分组的聚集，**Greenplum的两个基本原**则是：

1. **把属于同一**组的数据重分布到同一**segment上做聚集操作。**
2. **尽量把不同的**组分到不同的**segment上从而提高并**发度。

Greenplum 分布式查询计划生成: Aggregate with Group

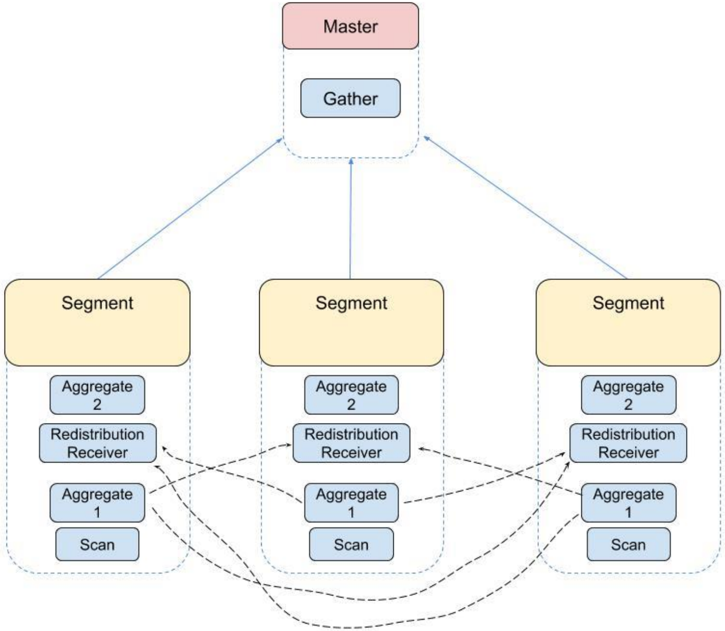
如果是根据分布键分组，那么每个**segment**执行一阶段聚集。



```
SELECT avg(c2) from t1 group by c1;
         QUERY PLAN
----------------------------
 Gather Motion 3:1
   ->  HashAggregate
         Group Key: c1
         ->  Seq Scan on t1
(4 rows)
```

Greenplum 分布式查询计划生成: Aggregate with Group

如果是根据非分布键分组，同时没有DISTINCT操作， 那么每个segment执行两阶段聚集。

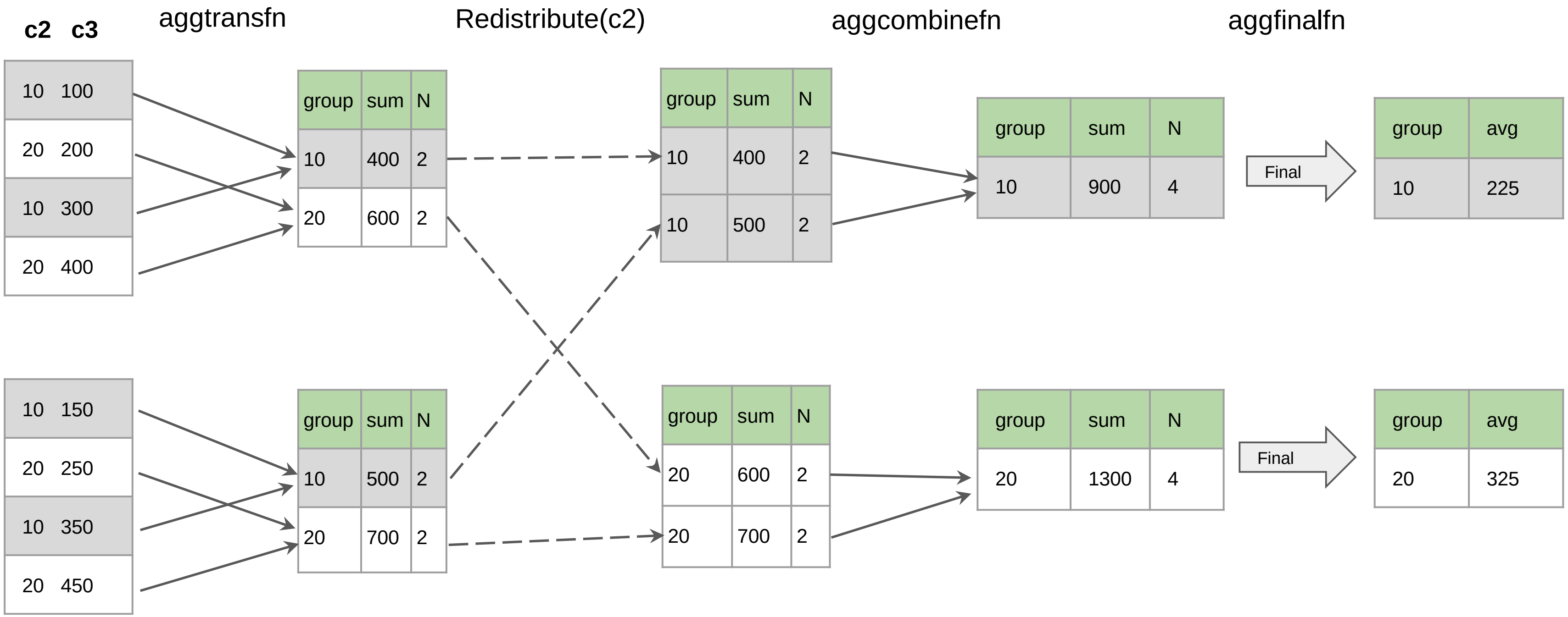


Redistributing (group key+transvalue) by group key

```
SELECT avg(c3) from t1 group by c2;
               QUERY PLAN
----------------------------------------
 Gather Motion 3:1
   ->  HashAggregate
         Group Key: t1.c2
         ->  Redistribute Motion 3:3
               Hash Key: t1.c2
               ->  HashAggregate
                     Group Key: t1.c2
                     ->  Seq Scan on t1
(8 rows)
```

Greenplum 分布式查询计划生成: Aggregate with Group

```
SELECT avg(c3) from t1 group by c2;
```



c2	c3
10	100
20	200
10	300
20	400

c2	c3
10	150
20	250
10	350
20	450

group	sum	N
10	400	2
20	600	2

group	sum	N
10	500	2
20	700	2

group	sum	N
10	400	2
10	500	2

group	sum	N
20	600	2
20	700	2

group	sum	N
10	900	4

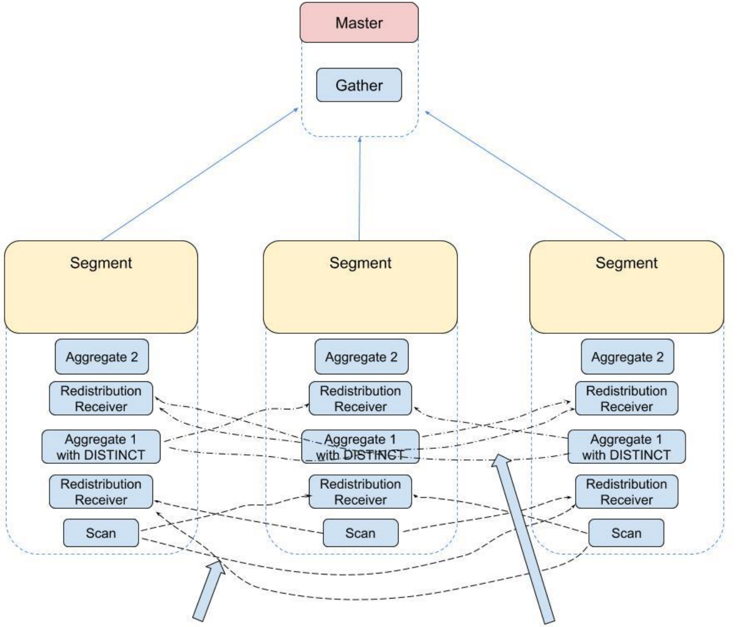
group	sum	N
20	1300	4

group	avg
10	225

group	avg
20	325

Greenplum 分布式查询计划生成: Aggregate with Group

如果是根据非分布键分组，且有DISTINCT操作，那么每个segment先要重分布数据，然后再执行两阶段聚集。



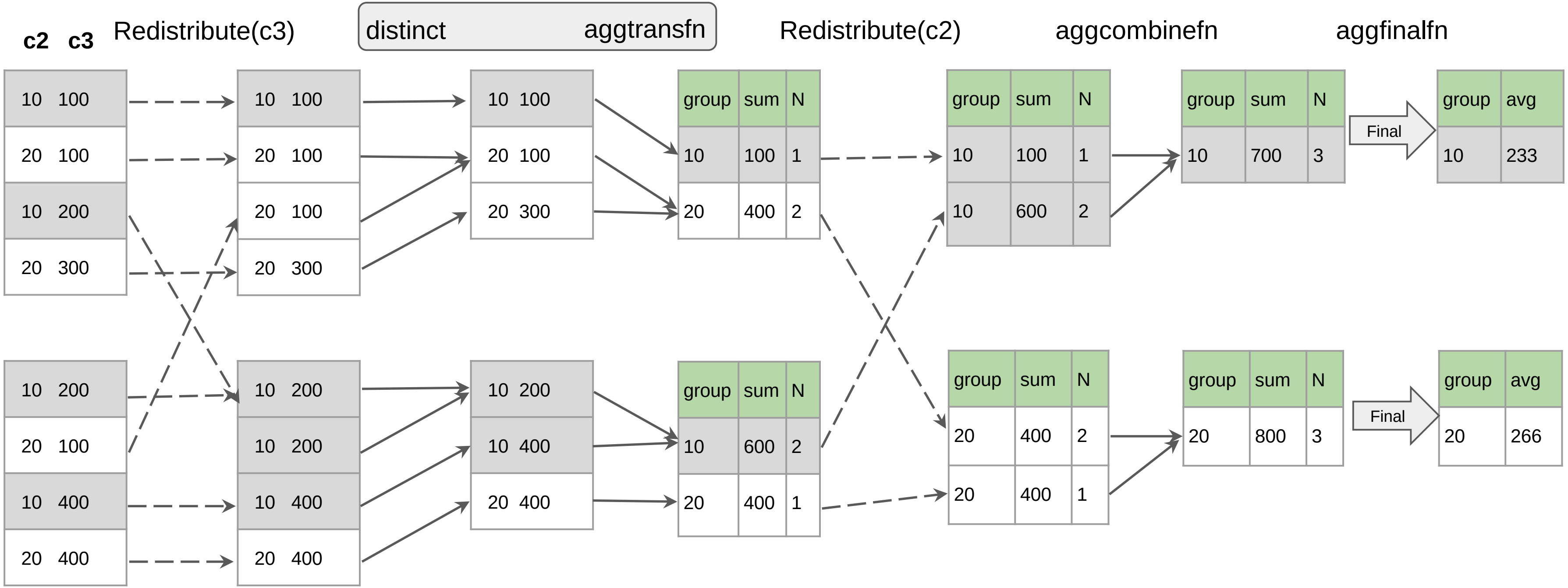
Redistributing tuples by aggregate key

Redistributing groupkey+transvalue by group key

```
SELECT avg(distinct c3) from t1 group by c2;
                         QUERY PLAN
-------------------------------------------------------------
 Gather Motion 3:1
   ->  GroupAggregate
         Group Key: t1.c2
         ->  Sort
               Sort Key: t1.c2
               ->  Redistribute Motion 3:3
                     Hash Key: t1.c2
                     ->  GroupAggregate
                           Group Key: t1.c2
                           ->  Sort
                                 Sort Key: t1.c2
                                 ->  Redistribute Motion 3:3
                                       Hash Key: t1.c3
                                       ->  Seq Scan on t1
(14 rows)
```

Greenplum 分布式查询计划生成: Aggregate with Group

```
SELECT avg(distinct c3) from t1 group by c2;
```



c2	c3
10	100
20	100
10	200
20	300

c2	c3
10	200
20	100
10	400
20	400

Redistribute(c3)

10	100
20	100
20	100
20	300

10	200
10	200
10	400
20	400

distinct

10	100
20	100
20	300

10	200
10	400
20	400

aggtransfn

group	sum	N
10	100	1
20	400	2

group	sum	N
10	600	2
20	400	1

Redistribute(c2)

group	sum	N
10	100	1
10	600	2

group	sum	N
20	400	2
20	400	1

aggcombinefn

group	sum	N
10	700	3

group	sum	N
20	800	3

aggfinalfn

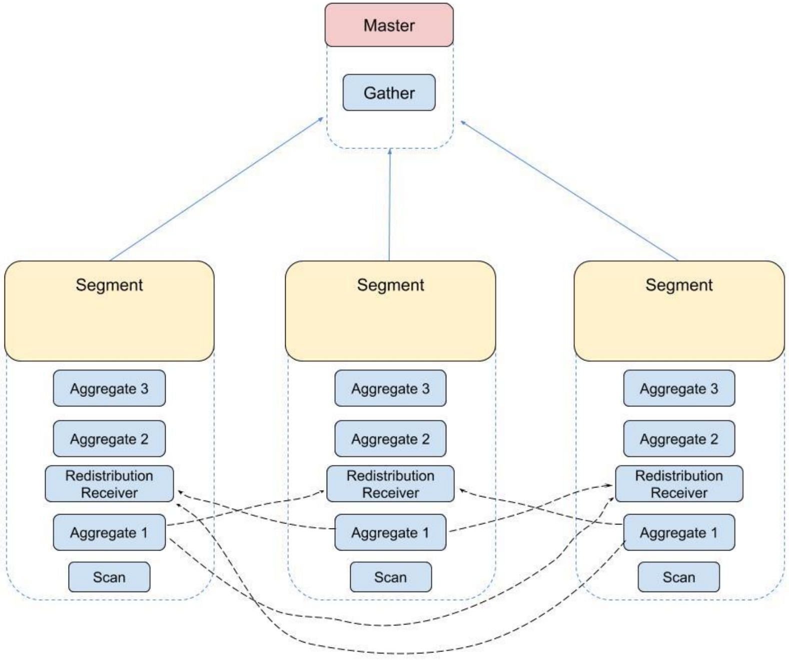
group	avg
10	233

group	avg
20	266

Pivotal

Greenplum 分布式查询计划生成: Aggregate with Group

或者是每个segment执行三阶段聚集。

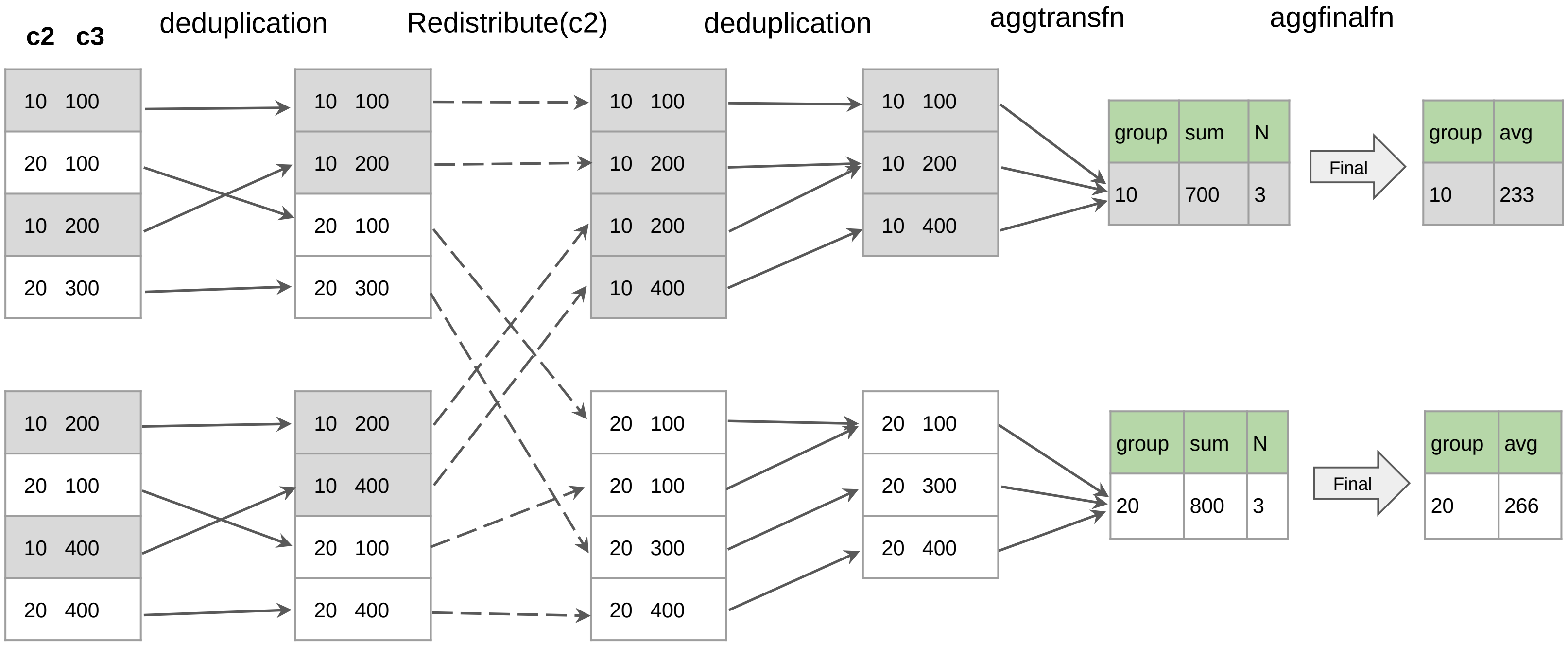


Redistributing (group key + aggregate key) by group key

```
SELECT avg(distinct c3) from t1 group by c2;
                    QUERY PLAN
---------------------------------------------------
 Gather Motion 3:1
   ->  HashAggregate
         Group Key: t1.c2
         ->  HashAggregate
               Group Key: t1.c2, t1.c3
               ->  Redistribute Motion 3:3
                     Hash Key: t1.c2
                     ->  HashAggregate
                           Group Key: t1.c2, t1.c3
                           ->  Seq Scan on t1
(10 rows)
```

Greenplum 分布式查询计划生成: Aggregate with Group

```
SELECT avg(distinct c3) from t1 group by c2;
```



c2	c3
10	100
20	100
10	200
20	300

10	200
20	100
10	400
20	400

deduplication

10	100
10	200
20	100
20	300

10	200
10	400
20	100
20	400

Redistribute(c2)

10	100
10	200
10	200
10	400

20	100
20	100
20	300
20	400

deduplication

10	100
10	200
10	400

20	100
20	300
20	400

aggtransfn

group	sum	N
10	700	3

group	sum	N
20	800	3

aggfinalfn

group	avg
10	233

group	avg
20	266

Greenplum 查询调优

Greenplum 查询调优: Example 1/2

```
# explain analyze select * from t1 join t2 on t1.c1 = t2.c2;
                                                               QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
-----
 Gather Motion 3:1  (slice2; segments: 3)  (cost=1192.55..5169.33 rows=50380 width=16)
   Rows out:  51500 rows at destination with 88 ms to first row, 187 ms to end.
   ->  Hash Join  (cost=1192.55..5169.33 rows=16794 width=16)
         Hash Cond: t2.c2 = t1.c1
         Rows out:  Avg 17166.7 rows x 3 workers.  Max 50000 rows (seg0) with 132 ms to first row, 166 ms to end.
         Executor memory:  524K bytes avg, 1563K bytes max (seg0).
         Work_mem used:  524K bytes avg, 1563K bytes max (seg0). Workfile: (3 spilling)
         Work_mem wanted: 1563K bytes avg, 1563K bytes max (seg0) to lessen workfile I/O affecting 1 workers.
         (seg0)   Initial batch 0:
         (seg0)   Initial batches 1..7:
         (seg0)   Secondary Overflow batches 8..15:
         (seg0)   Hash chain length 50000.0 avg, 50000 max, using 1 of 1024 buckets.  Skipped 15 empty batches.
         ->  Redistribute Motion 3:3  (slice1; segments: 3)  (cost=0.00..3098.10 rows=33190 width=8)
               Hash Key: t2.c2
               Rows out:  Avg 33333.3 rows x 3 workers at destination.  Max 33348 rows (seg0) with 0.429 ms to first row, 38 ms to
end.
               ->  Seq Scan on t2  (cost=0.00..1106.70 rows=33190 width=8)
                     Rows out:  Avg 33333.3 rows x 3 workers.  Max 33348 rows (seg0) with 0.049 ms to first row, 4.865 ms to end.
         ->  Hash  (cost=562.80..562.80 rows=16794 width=8)
               Rows in:  Avg 99.5 rows x 2 workers.  Max 125 rows (seg1) with 1.178 ms to end, start offset by 0.929 ms.
               ->  Seq Scan on t1  (cost=0.00..562.80 rows=16794 width=8)
                     Rows out:  Avg 17166.7 rows x 3 workers.  Max 50000 rows (seg0) with 0.048 ms to first row, 7.870 ms to end.
 Optimizer status: legacy query optimizer
(38 rows)
```

Pivotal

Greenplum 查询调优: Example 1/2

1. 数据倾斜

    ALTER TABLE SET DISTRIBUTED BY

1. 数据溢出

    SET statement_mem TO XXX

Greenplum 查询调优: Example 2/2

```
# EXPLAIN ANALYZE select avg(distinct c2) from t1 group by c2;
                                                                   QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------------------
------------
--
 Gather Motion 3:1  (slice2; segments: 3)  (cost=144826.86..164922.45 rows=946623 width=36)
   Rows out:  1000000 rows at destination with 829 ms to first row, 6591 ms to end.
   ->  GroupAggregate  (cost=144826.86..164922.45 rows=315541 width=36)
         Group By: c2
         Rows out:  Avg 333333.3 rows x 3 workers.  Max 333385 rows (seg2) with 828 ms to first row, 6623 ms to end.
         Executor memory:  10692717K bytes avg, 10694374K bytes max (seg2).
         Work_mem used:  33K bytes avg, 33K bytes max (seg0).
         ->  Sort  (cost=144826.86..147581.13 rows=367236 width=4)
               Sort Key: c2
               Sort Method:  quicksort  Max Memory: 36857KB  Avg Memory: 36857KB (3 segments)
               Rows out:  Avg 366666.7 rows x 3 workers.  Max 366704 rows (seg0) with 823 ms to first row, 892 ms to end.
               Executor memory:  36857K bytes avg, 36857K bytes max (seg0).
               Work_mem used:  36857K bytes avg, 36857K bytes max (seg0). Workfile: (0 spilling)
               ->  Redistribute Motion 3:3  (slice1; segments: 3)  (cost=0.00..34263.24 rows=367236 width=4)
                     Hash Key: c2
                     Rows out:  Avg 366666.7 rows x 3 workers at destination.  Max 366704 rows (seg0) with 1.061 ms to first row,
422 ms to end
.
                     ->  Seq Scan on t1  (cost=0.00..12229.08 rows=367236 width=4)
                           Rows out:  Avg 366666.7 rows x 3 workers.  Max 366704 rows (seg0) with 0.052 ms to first row, 122 ms to
end.
 Optimizer status: legacy query optimizer
(27 rows)
```

Greenplum 查询调优: Example 2/2

多阶段聚集

SET gp_enable_multiphase_agg TO on;

SET gp_enable_agg_distinct TO on; ← 2-phase DQA

SET gp_enable_agg_distinct_pruning TO on; ← 3-phase DQA

Pivotal®

Transforming How The World Builds Software

© Copyright 2018 Pivotal Software, Inc. All rights Reserved.