Pivotal

Generating distributed plan for PostgreSQL

Richard Guo / Pivotal

2019/05/31, PGCon 2019

Who Am I?

- Richard Guo | 峰(Feng) 郭(Guo)
- From Beijing, China
- Working at Pivotal on Greenplum
  - Merge
  - MPP
  - Support

Agenda

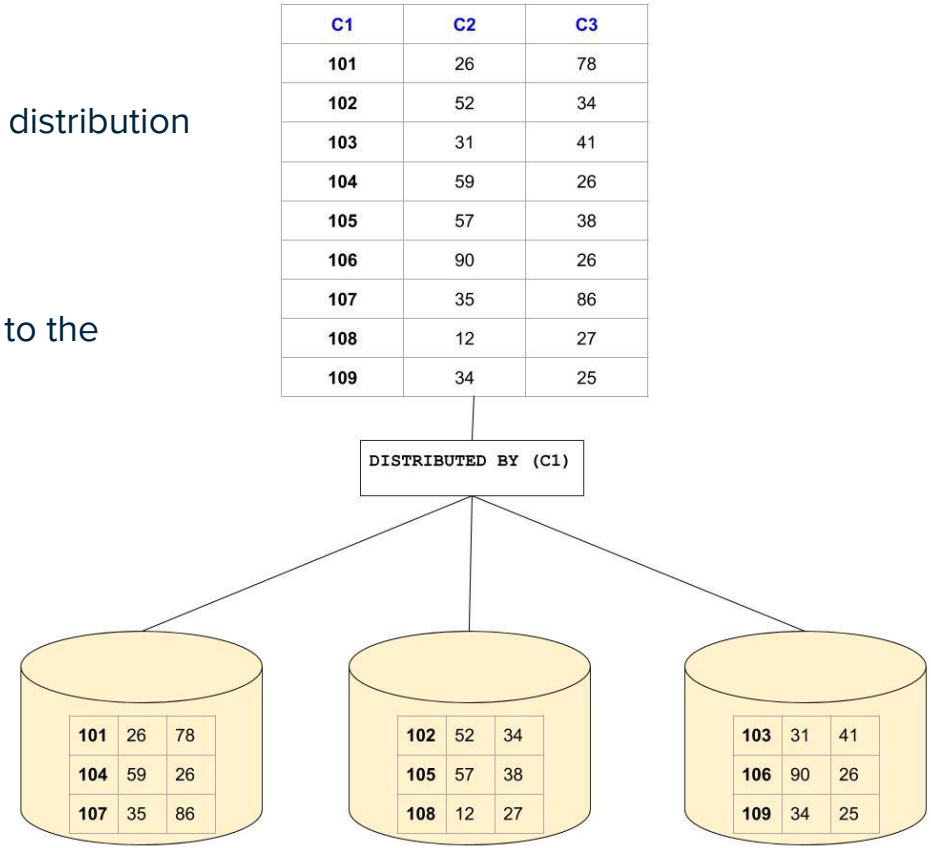
- When PostgreSQL meets MPP
- Distributed Plan
  - Scan
  - Join
  - Aggregation
- Comparison with Parallel Query in PostgreSQL

When PostgreSQL meets MPP

- MPP (massively parallel processing) refers to systems with two or more processors that cooperate to carry out an operation, each processor with its own memory, operating system and disks.
- Greenplum(master) = PostgreSQL(9.4) + MPP
  - Several PostgreSQL instances to contain user-defined tables and their indexes, called segments, and each segment contains a distinct portion of user data.
  - One PostgreSQL instance to generate plan, coordinate its work with segments, called master, and master contains no user data.

When PostgreSQL meets MPP

- Hash distribution
  - Tuples are distributed based on values of a distribution key (one or more columns).
- Random distribution
  - Tuples are distributed in a round-robin way to the segments.



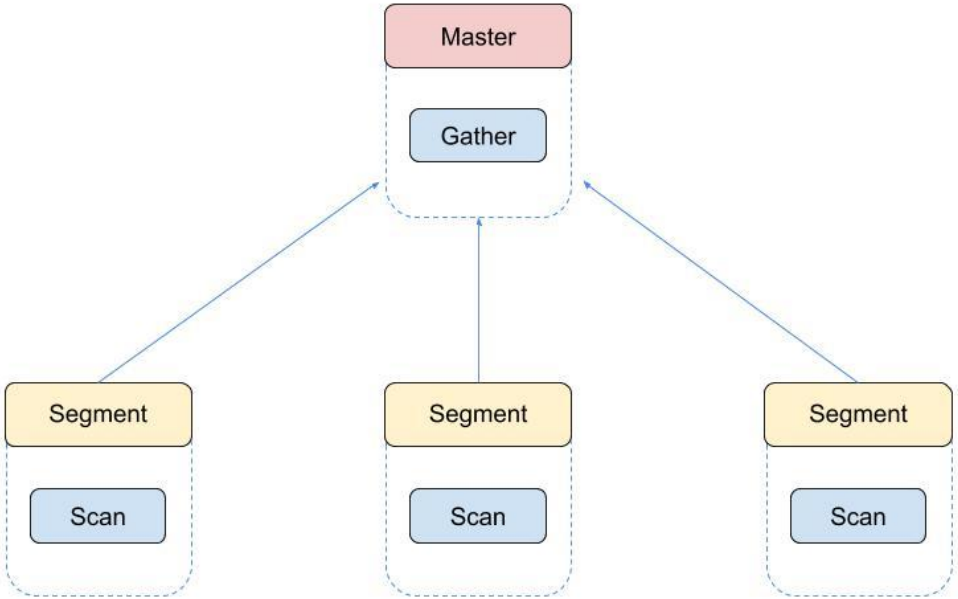
When PostgreSQL meets MPP

Tables used as an example:

```
CREATE TABLE t1 (c1 int, c2 int, c3 int) DISTRIBUTED BY (c1);
CREATE TABLE t2 (c1 int, c2 int, c3 int) DISTRIBUTED BY (c1);
```

Distributed Plan: Scan

1. Each segment scans on its local data.
2. Master runs gather and outputs the complete results.



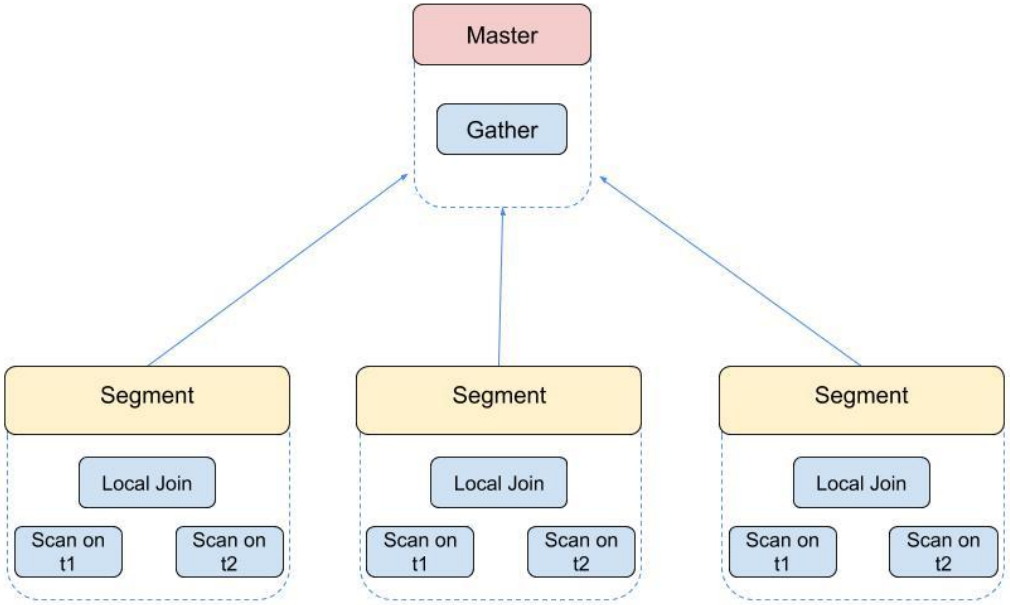

```
SELECT * from t1 where t1.c2 = 1;
        QUERY PLAN
--------------------------
 Gather Motion 3:1
   ->  Seq Scan on t1
         Filter: (c2 = 1)
(3 rows)
```

Distributed Plan: Join

1. To perform a join, matching rows must be located together on the same segment. If not, data needs to be routed among segments.
    - Redistribute: each segment rehashes the data and sends the rows to the appropriate segments according to hash key.
    - Broadcast: each segment sends its own, individual rows to all other segments so that every segment instance has a complete local copy of the table.
2. Each segment then performs the join locally, in parallel.
3. Master runs gather and outputs the complete results.

Distributed Plan: Join

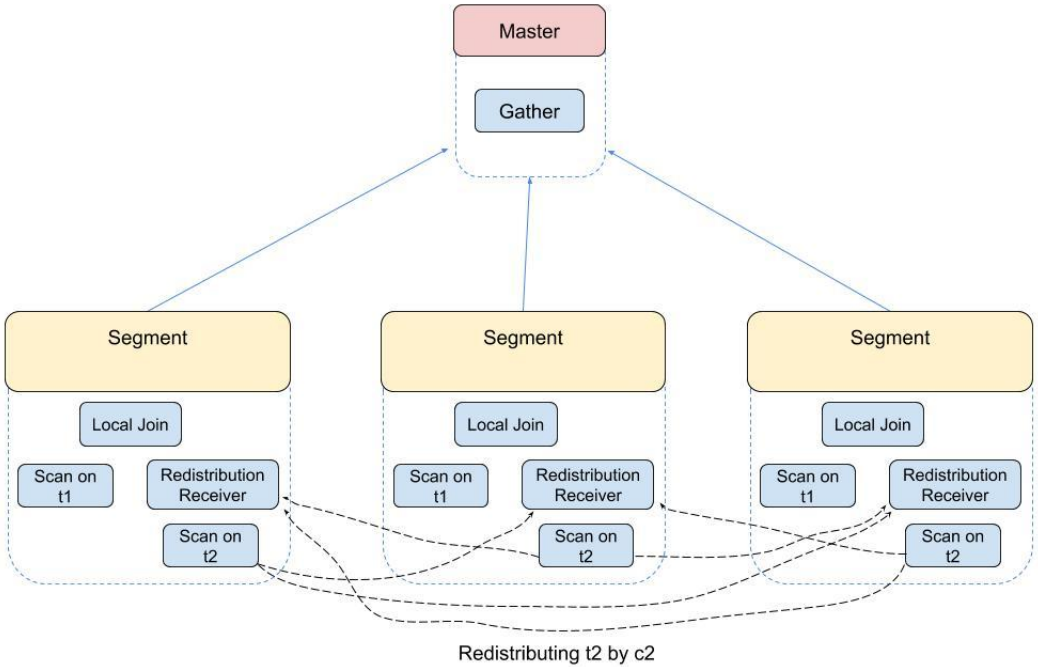
If we are performing equi-join on distribution keys, we do not need route data among segments.




```
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)
```

Distributed Plan: Join

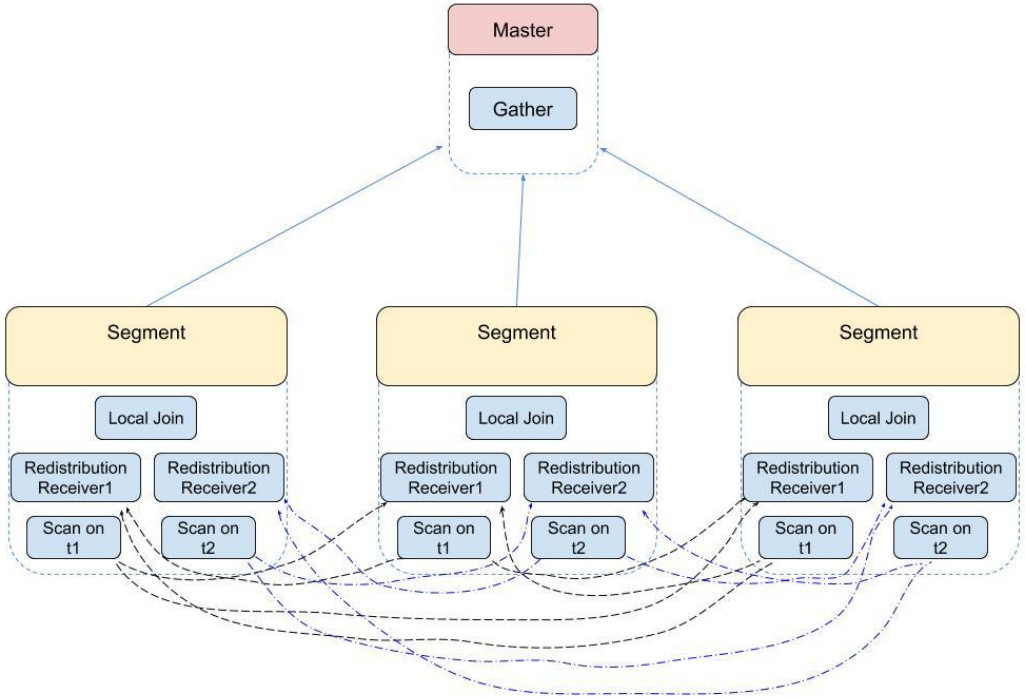
Else we may need to redistribute one table,




```
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)
```

Distributed Plan: Join

Or two tables,

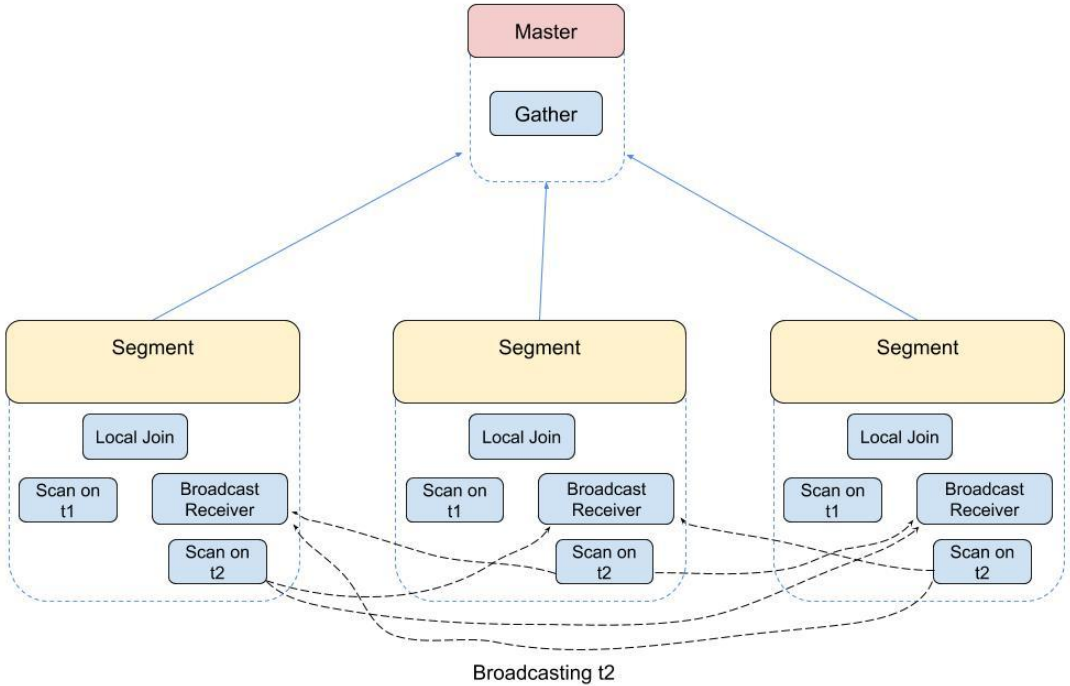


Redistributing t1 and t2 by c2

```
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)
```

Distributed Plan: Join

Or instead broadcast one table, depending on costs.




```
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)
```

Distributed Plan: Join

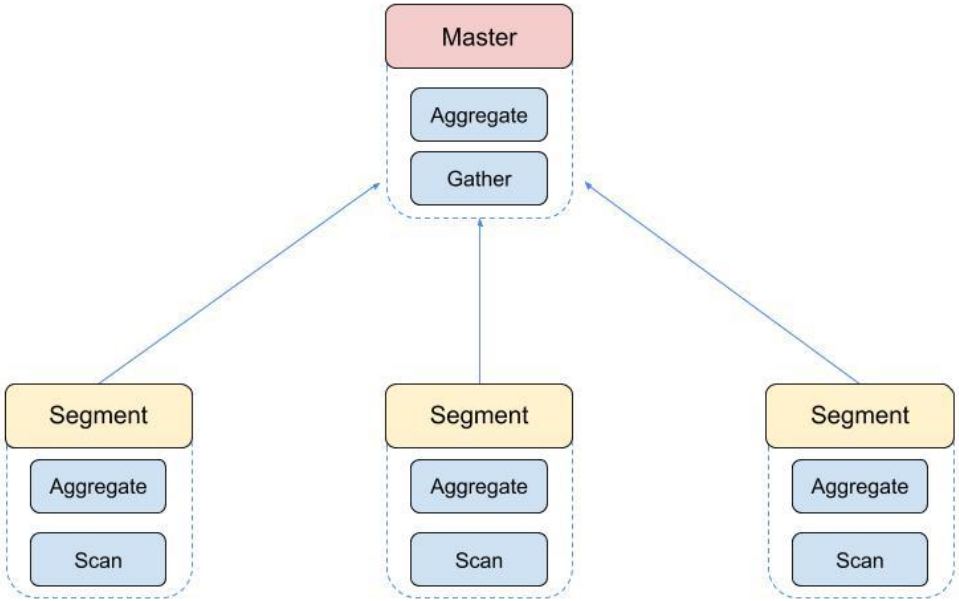
- For non equi-join, we cannot perform redistribution. We can only perform broadcast.
- But be careful about broadcast. In the case of outer joins, broadcasting non-nullable side would cause problems. It may cause the outer join to emit null-extended rows that should not have been formed. So always broadcast the nullable side.

Distributed Plan: Aggregate without Group

1. To perform aggregation with DISTINCT, tuples with the same values of the DISTINCT columns must be located on the same segment in order to perform deduplication. If not, data needs to be redistributed among segments.
2. Each segment then performs the aggregation locally, in parallel.
    a. aggtransfn and NULL aggfinalfn
3. Master runs gather and performs a final aggregation.
    a. aggcombinefn and aggfinalfn

Distributed Plan: Aggregate without Group

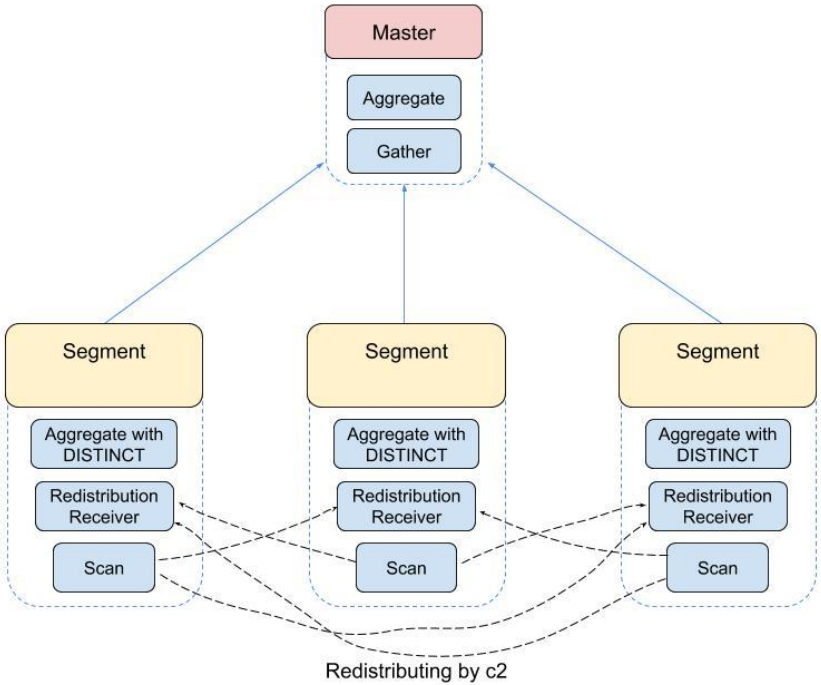
No need to redistribute data if there is no DISTINCT, or DISTINCT only on distribution key.




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

Distributed Plan: Aggregate without Group

Need to redistribute data if there is DISTINCT on non-distribution key.




```
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)
```

Distributed Plan: Aggregate with Group

1. Gather tuples belonging to the same group to the same segment for aggregation.
2. Distribute different groups to different segments for parallelism.

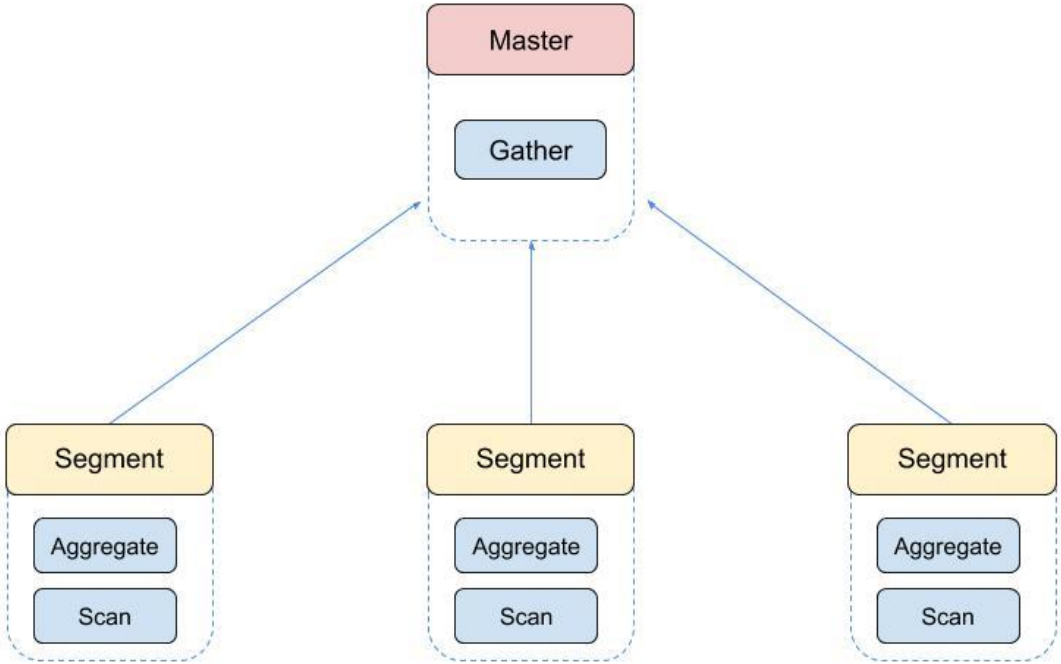
Group by Distribution Key?	**DISTINCT?**	
YES	*DON'T CARE*	→ **CASE 1**
NO	NO	→ **CASE 2**
NO	YES	→ **CASE 3**

Distributed Plan: Aggregate with Group

CASE 1: Group by distribution key

1. Each segment performs the aggregation locally, in parallel.
2. Master runs gather and outputs the complete results.

Distributed Plan: Aggregate with Group




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

Distributed Plan: Aggregate with Group

CASE 2: Group by non-distribution key, and no DISTINCT.

1. Each segment redistributes tuples by group key.
2. Each segment performs the aggregation locally, in parallel.
3. Master runs gather and outputs the complete results.

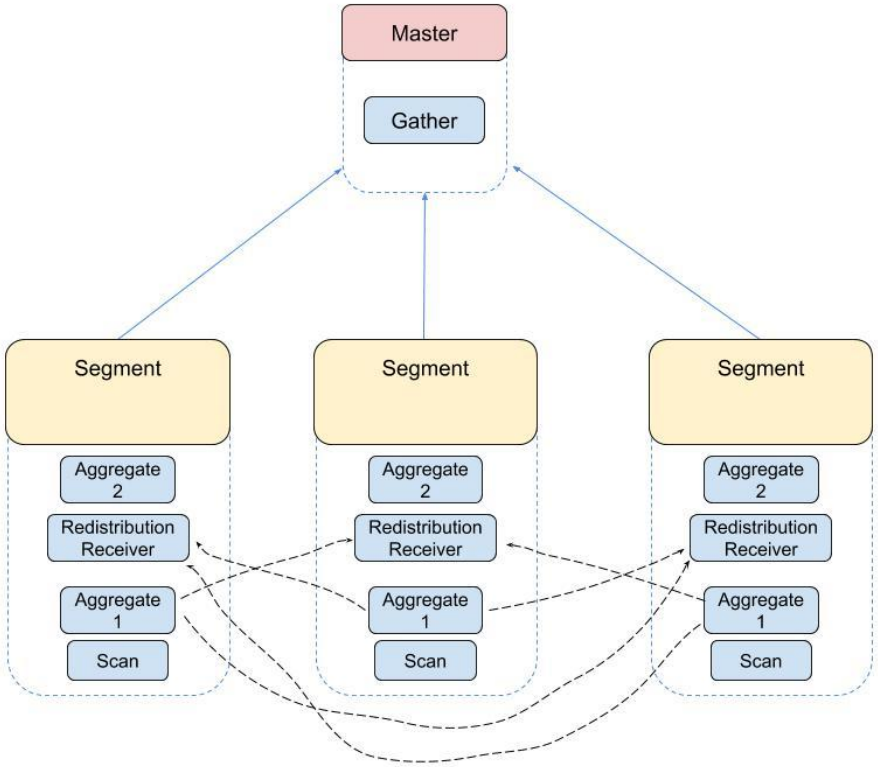
CASE 1

Distributed Plan: Aggregate with Group

CASE 2: Group by non-distribution key, and no DISTINCT.

1. Each segment performs the aggregation locally for the first time, in parallel.
    a. aggtransfn and NULL aggfinalfn.
2. Each segment redistributes data (group key + transvalue) by group key.
3. Each segment performs the aggregation locally for the second time, in parallel.
    a. aggcombinefn and aggfinalfn.
4. Master runs gather and outputs the complete results.

Distributed Plan: Aggregate with Group

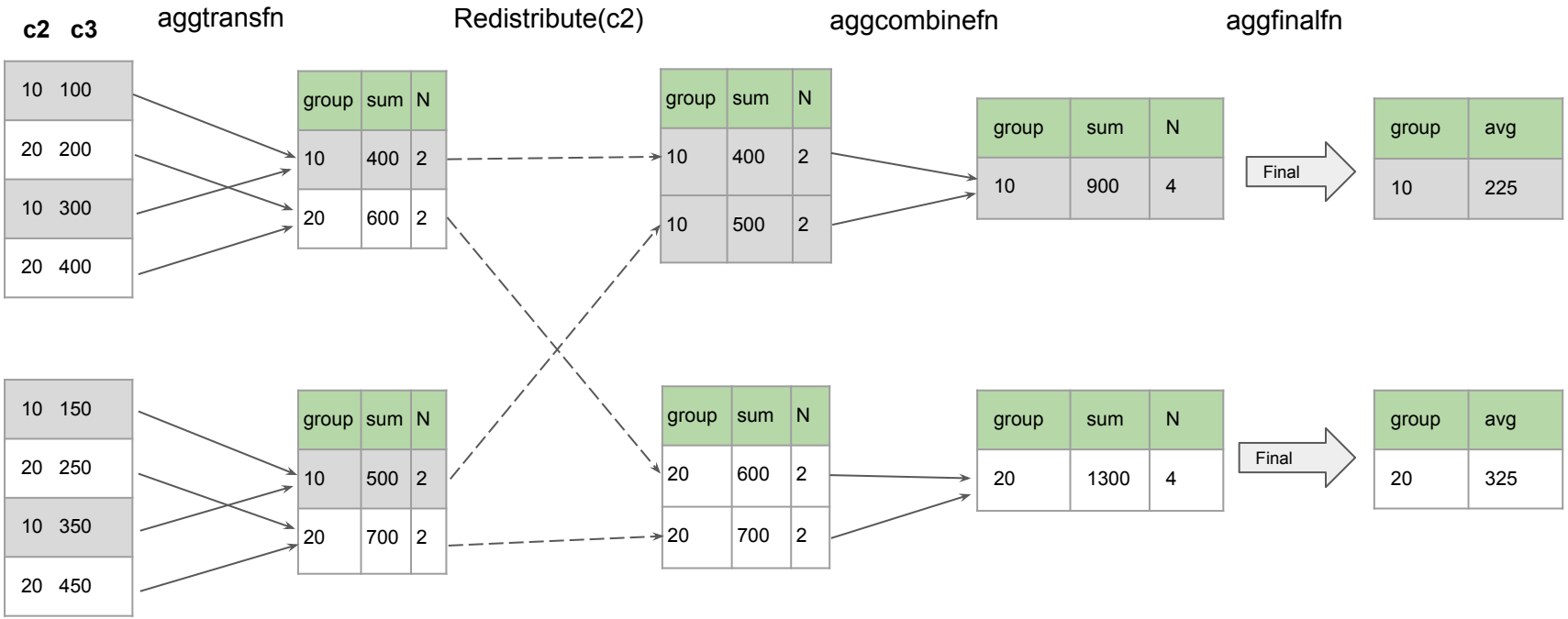



```
SELECT avg(c3) from t1 group by c2;
                QUERY PLAN
---------------------------------------
 Gather Motion 3:1
   ->  HashAggregate
         Group Key: t1.c2
         ->  Redistribute Motion 3:3
               Hash Key: t1.c2
               ->  HashAggregate
                     Group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

Pivotal

Distributed Plan: Aggregate with Group

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



Distributed Plan: Aggregate with Group

CASE 3: Group by non-distribution key, aggregate with DISTINCT.

1. Each segment redistributes tuples by group key.
2. Each segment performs the aggregation locally, in parallel.
3. Master runs gather and outputs the complete results.

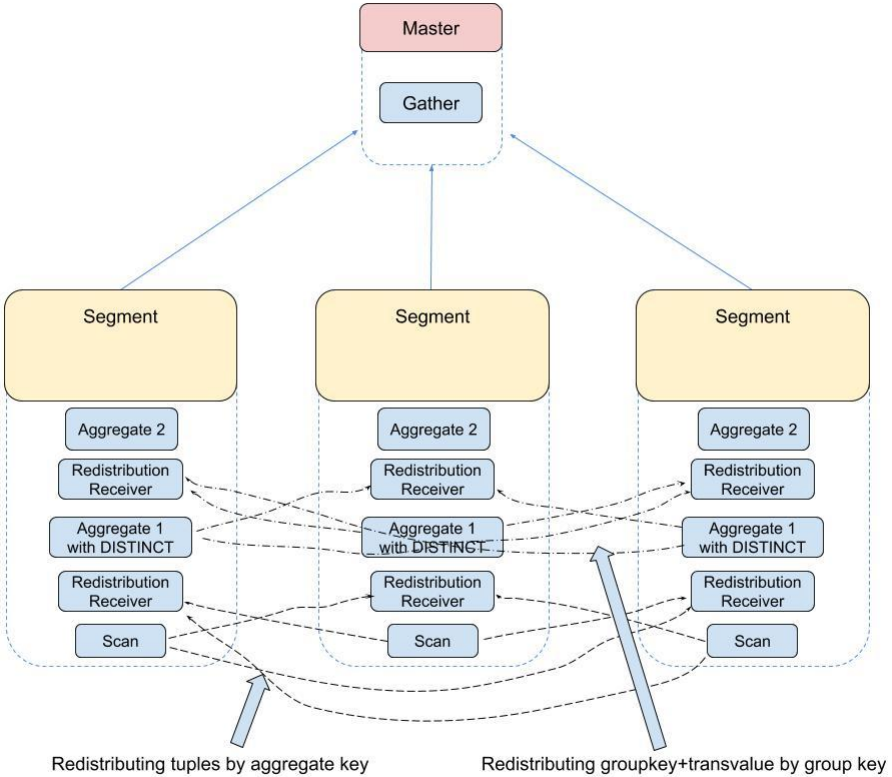
CASE 1

Distributed Plan: Aggregate with Group

CASE 3: Group by non-distribution key, aggregate with DISTINCT.

1. Each segment redistributes tuples by aggregate key.
2. Each segment performs the aggregation locally for the first time, in parallel.
    a. aggtransfn and NULL aggfinalfn, applying DISTINCT.
3. Each segment redistributes data (group key + transvalue) by group key.
4. Each segment performs the aggregation locally for the second time, in parallel.
    a. aggcombinefn and aggfinalfn.
5. Master runs gather and outputs the complete results.

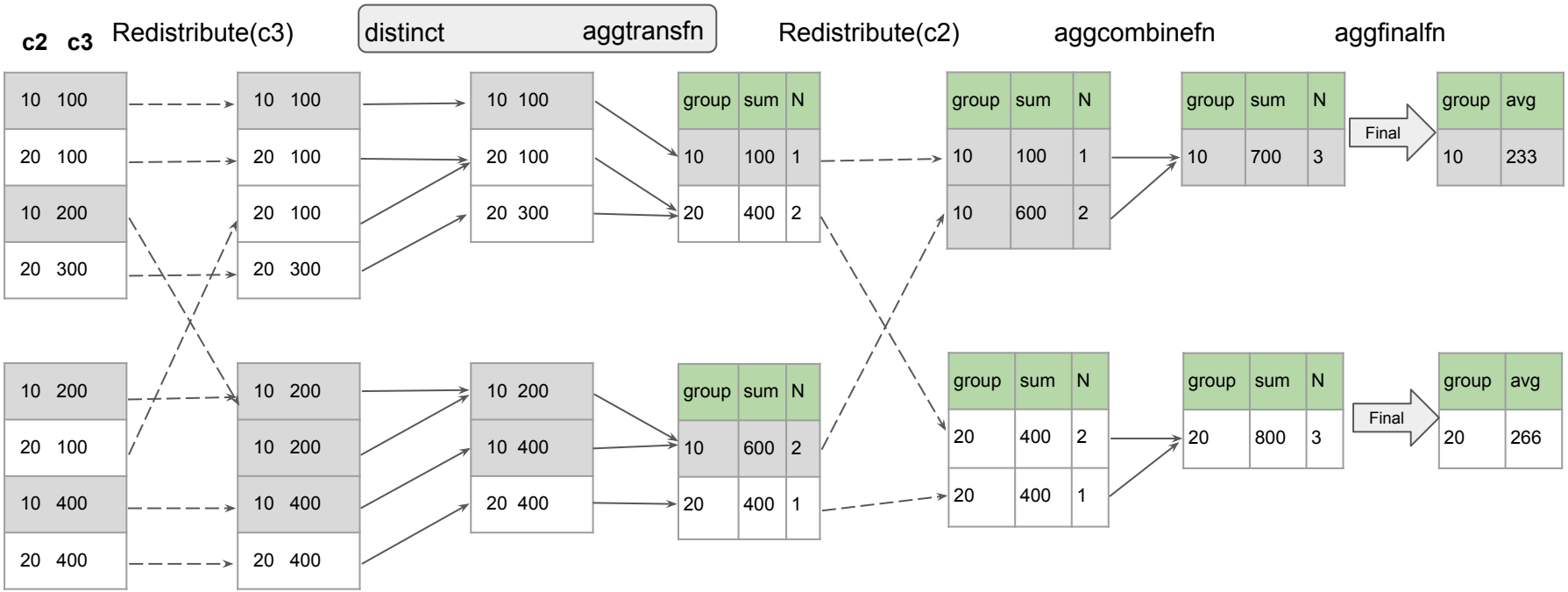
Distributed Plan: Aggregate with Group




```
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)
```

Distributed Plan: Aggregate with Group

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

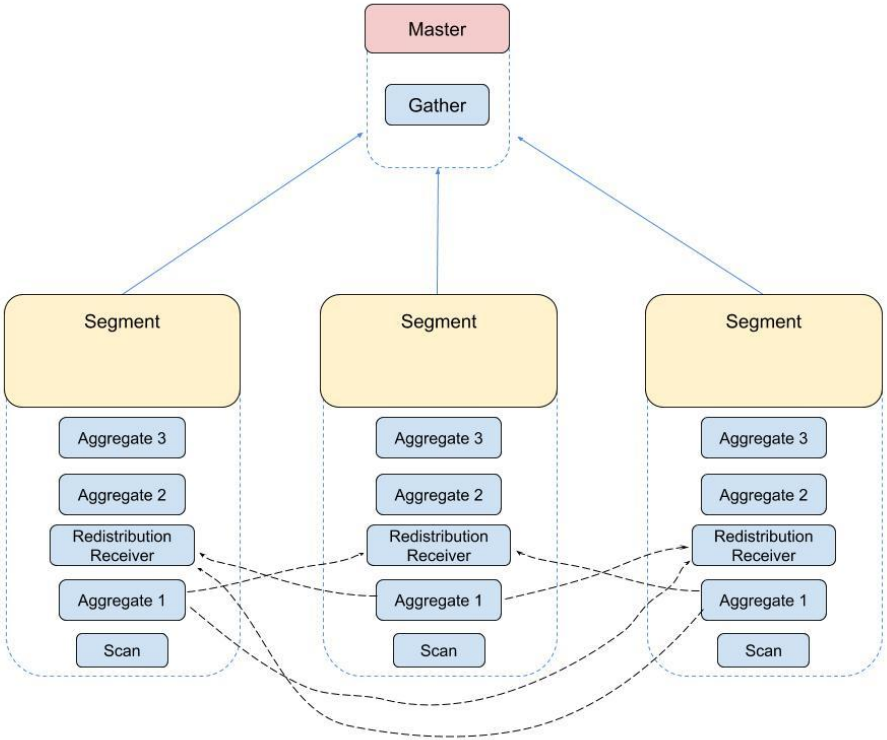


Distributed Plan: Aggregate with Group

CASE 3: Group by non-distribution key, aggregate with DISTINCT.

1. Each segment performs local deduplication via hash aggregation by (group key + aggregate key).
    a. Then the aggregate key is distinct within a group locally.
2. Each segment redistributes data (group key + aggregate key) by group key.
3. Each segment performs another deduplication via another hash aggregation by (group key + aggregate key).
    a. Then the aggregate key is distinct within a group globally.
4. Each segment performs the aggregation locally, in parallel.
    a. aggtransfn and aggfinalfn
5. Master runs gather and outputs the complete results.

Distributed Plan: Aggregate with Group

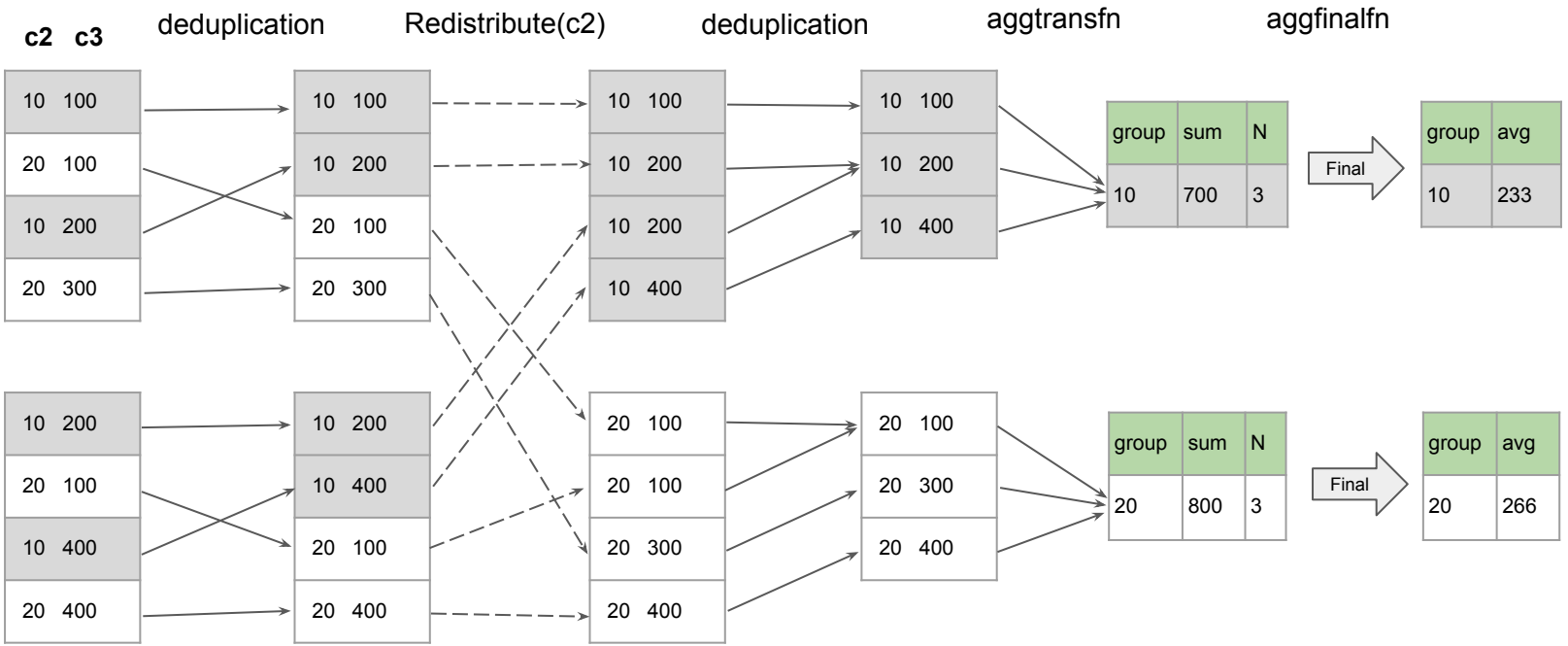



```
SELECT avg(distinct c3) from t1 group by c2;
                    QUERY PLAN
-------------------------------------------------
 Gather Motion 3:1
   ->  HashAggregate
         Group Key: t1.c2
         ->  HashAggregate
               Group Key: t1.c2, t1.c3
               ->  Redistribute Motion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

Pivotal

Distributed Plan: Aggregate with Group

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



Pivotal

Comparison with Parallel Query in PostgreSQL

1. Tuples need to be distributed in a pre-defined policy.
2. Exchange operators
    - Redistribution, Broadcast, Gather
3. Tuples can be routed among segments via redistribution or broadcast. And tuples can flow between segments and master via gather.
    - Final aggregation can be done by segments in parallel.
    - Parallel DISTINCT aggregation is supported.

Greenplum

1. Tuples do not need to be distributed well in advance.
2. Exchange operators
    - Gather
3. Tuples can only flow between workers and gather.
    - Final aggregation can be done by gather only.
    - Parallel DISTINCT aggregation is not supported.

Parallel Query in PostgreSQL

Case-study: Parallel Aggregation

1. Each segment performs an aggregation step, producing a partial result for each group that exists on that segment.
    - First stage, performed in parallel.
2. The partial results are redistributed among segments by group key.
3. Each segment re-aggregates the results, producing the final result for each group it received.
    - Second stage, performed in parallel.
4. The final results are transferred to master for outputs via Gather.

Greenplum

1. Each worker performs an aggregation step, producing a partial result for each group of which that worker is aware.
    - First stage, performed in parallel.
2. The partial results are transferred to the leader via Gather or Gather Merge.
3. The leader re-aggregates the results across all workers in order to produce the final result.
    - Second stage, performed by leader only.

Parallel Query in PostgreSQL

Thank You

Output: Thank You

```
Gather Motion 3:1  (slice1; segments: 3)
   ->  Index Scan using Common_phrases_idx on Common_phrases
         Index Cond: (value = 'Thank You')
         Filter: (Language = 'English')
```