GREENPLUM DATABASE®

Greenplum 分布式数据库内核揭秘

李正龙

Greenplum内核开发工程师

2022-03-16

vmware®

GREENPLUM DATABASE®

Agenda

- Greenplum 分布式数据库简介
- Greenplum 集群化概述
- 分布式数据存储与多态存储
- 分布式查询优化器与执行器
- Greenplum 中文社区

GREENPLUM DATABASE®

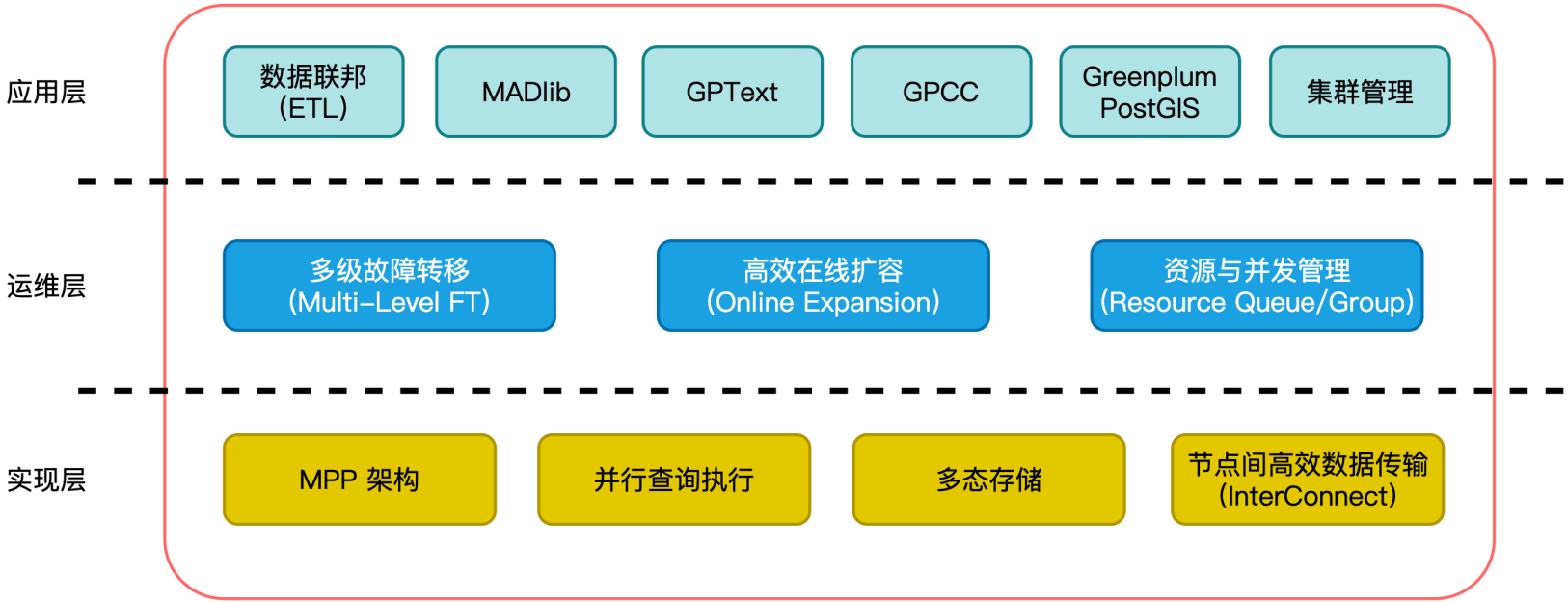
Greenplum 分布式数据库简介

Features

GREENPLUM DATABASE

Greenplum 分布式数据库简介

Greenplum 是基于 PostgreSQL 所实现的大规模并行处理(MPP)开源数据平台，具有良好的弹性和线性拓展能力，内置并行存储、并行通信、并行计算和并行优化功能，兼容 SQL 标准。拥有独特的高效的 ORCA 优化器，具有强大、高效的 PB 级数据存储、处理和实时分析能力，同时支持 OLTP 型业务的混合负载。



GREENPLUM DATABASE®

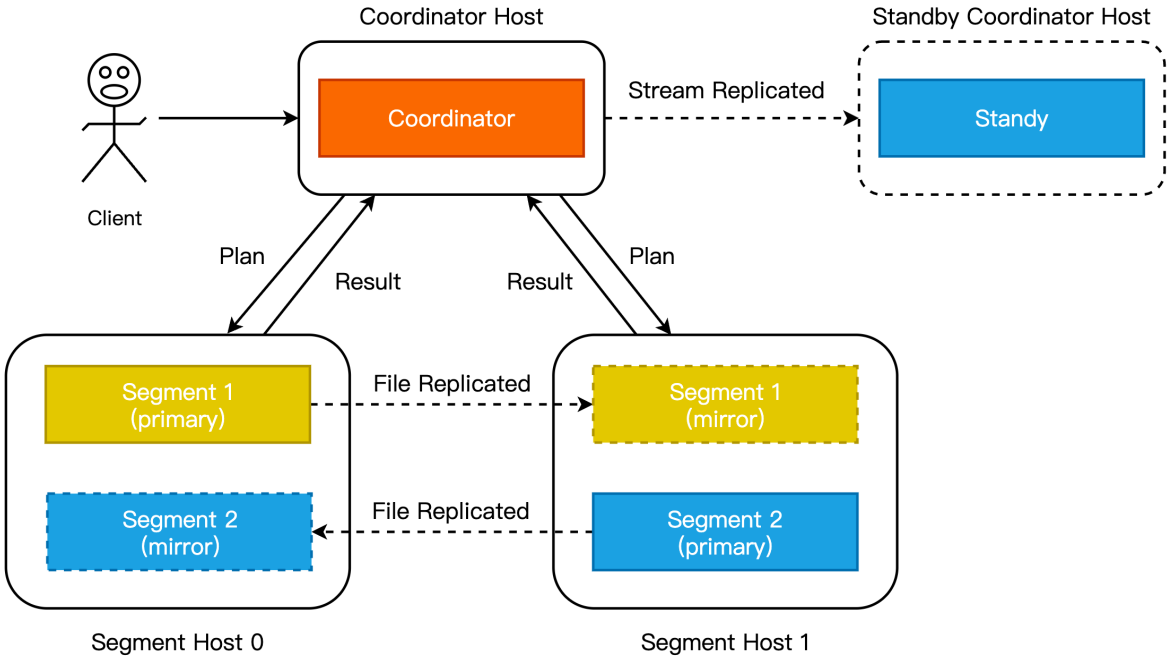
Greenplum 集群化概述

Coordinator/Segment, Primary/Mirror

GREENPLUM DATABASE

Greenplum 集群化概述

数据库的组成



- Coordinator/Segment 架构
- Greenplum 集群通常由一个 Coordinator 节点、一个 Standby Coordinator 节点以及多个 Segment 节点组成
- Coordinator 是整个数据库的入口，客户端只会连接至 Coordinator 节点，并执行相关的查询操作
- Standby 节点为 Coordinator 提供高可用支持
- Mirror 则为 Segment 提供高可用支持

GREENPLUM DATABASE®

Greenplum 分布式数据存储与多态存储

Hash/Randomly/Replicated

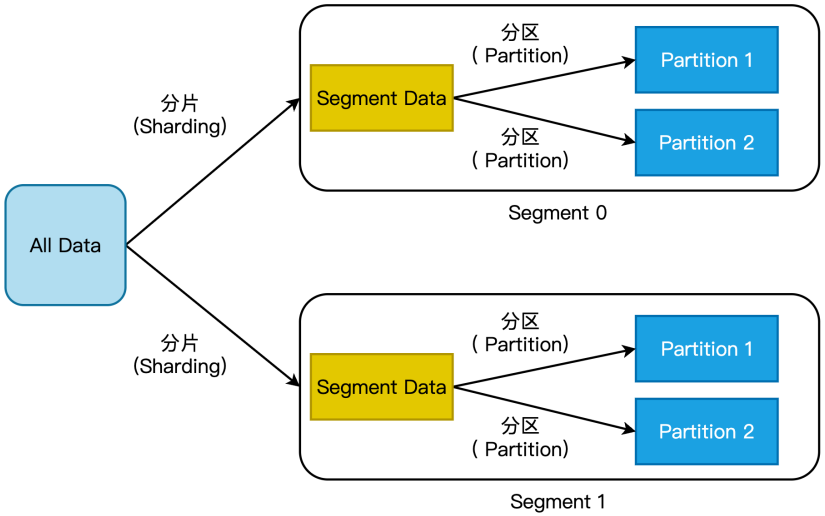
GREENPLUM DATABASE®

分布式数据存储

数据存储分布化是分布式数据库要解决的第一个问题。

通过将海量数据分散到多个节点上，一方面大大降低了单个节点处理的数据量，另一方面也为处理并行化奠定了基础，两者结合起来可以极大的提高系统的性能。譬如在 100 个节点的集群上，每个节点仅保存总数据量的 1/100，100 个节点同时并行处理，性能会是单个配置更强节点的几十倍。

Greenplum 不仅仅实现了基本的分布式数据存储，还提供了更高级更灵活的特性，譬如**多种分布策略**、**多级分区**以及**多态存储**。



GREENPLUM DATABASE®

数据分布策略

Greenplum 6 提供了以下 3 种数据分布策略:

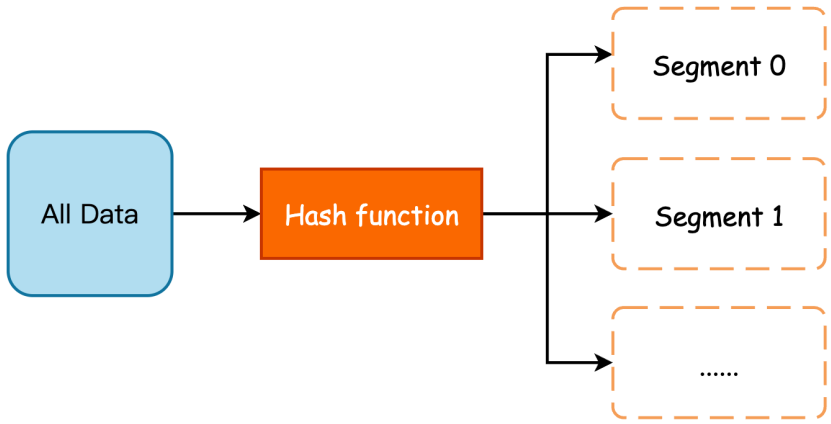
- 哈希分布 (Hash Distribution)
- 随机分布 (Randomly Distribution)
- 复制分布 (Replicated Distribution)

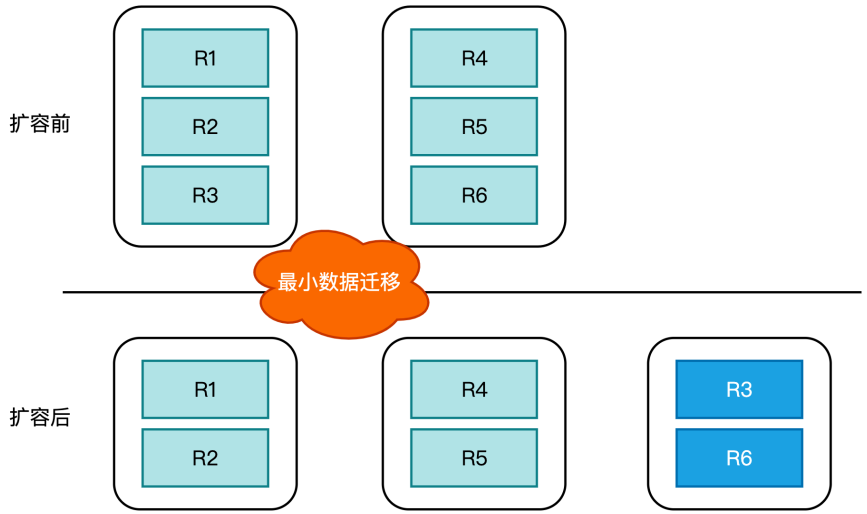
GREENPLUM DATABASE

哈希分布

哈希分布是分布式数据库最为常用的数据分布方式。根据用户自定义的分布键计算哈希值，然后将哈希结果映射到某个 Segment 上。在 Greenplum 6 中，默认采用一致性哈希(Jump Consistent Hash)分布策略。

当增加一个新的节点时，需要对原有数据进行重新映射。一致性哈希则保证了在重新映射的过程追中，**tuple 要么保留在原有节点中，要么迁移至新的节点中**，从而实现最小数据迁移。



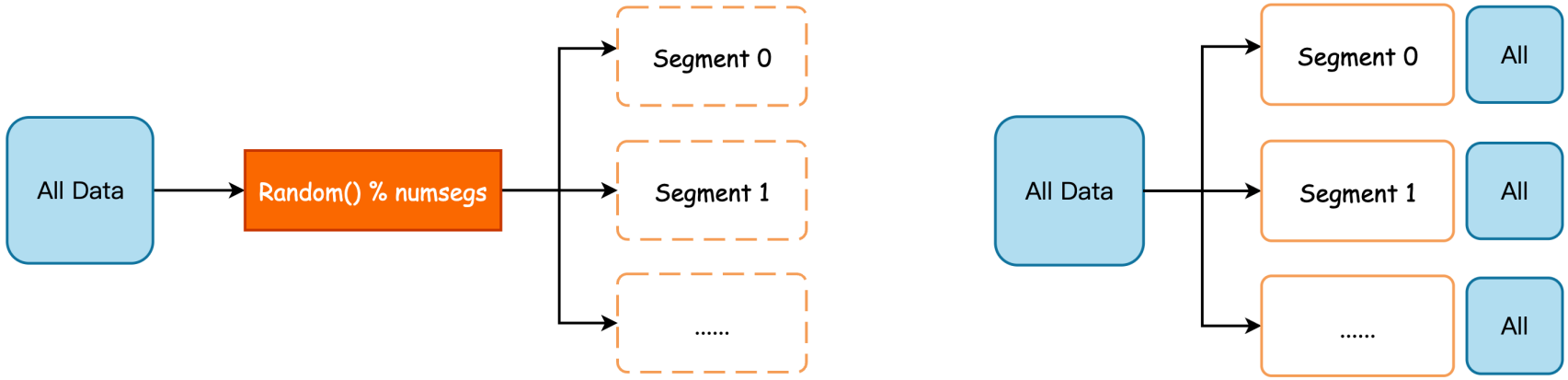


GREENPLUM DATABASE®

随机分布与复制分布

随机分布则采用随机的方式将数据存储到不同的节点。当不确定一张表的哈希分布键，或者是不存在合理的避免数据倾斜的分布键时，即可采用随机分布的方式。

复制分布则表示整张表在每个节点上都有一份完整的拷贝，假设我们有 100 个节点，复制表则会将数据保存 100 份。复制表可避免生成分布式查询计划，而是生成本地计划，从而避免数据在集群的不同节点间移动。



GREENPLUM DATABASE®

分区表

除了支持数据在不同的 segment 节点上水平分布以外，还支持在单个节点按照不同的标准进行分区，将单个节点上一个逻辑上的大表分割成物理上的几块，且支持多级分区。

Greenplum 目前支持的分区方法有：

- 范围分区：根据某个列的时间范围或者数值范围对数据进行分区。譬如以下 SQL 将创建一个按天分区的分区表，将 2021-01-01 到 2022-01-01 这一年的数据分成 366 个分区：

```
CREATE TABLE sales (id int, created date, amount int) DISTRIBUTED BY (id)
PARTITION BY RANGE(created)
(START (date '2021-01-01') INCLUSIVE
 END (date '2022-01-01') EXCLUSIVE
 EVERY (INTERVAL '1 day'));
```

- 列表分区：按照某个列的数值列表，将数据分到不同的分区。譬如以下 SQL 将根据性别创建一个分区表，共有 3 个分区：一个分区存储男士数据，一个分区存储女士数据。对于其它值譬如 NULL，在存储在默认分区 others 中：

```
CREATE TABLE rank (id int, rank int, gender char(1)) DISTRIBUTED BY (id)
PARTITION BY LIST(gender)
(PARTITION girls VALUES ('F'),
 PARTITION boys VALUES ('M'),
 DEFAULT PARTITION others);
```

GREENPLUM DATABASE®

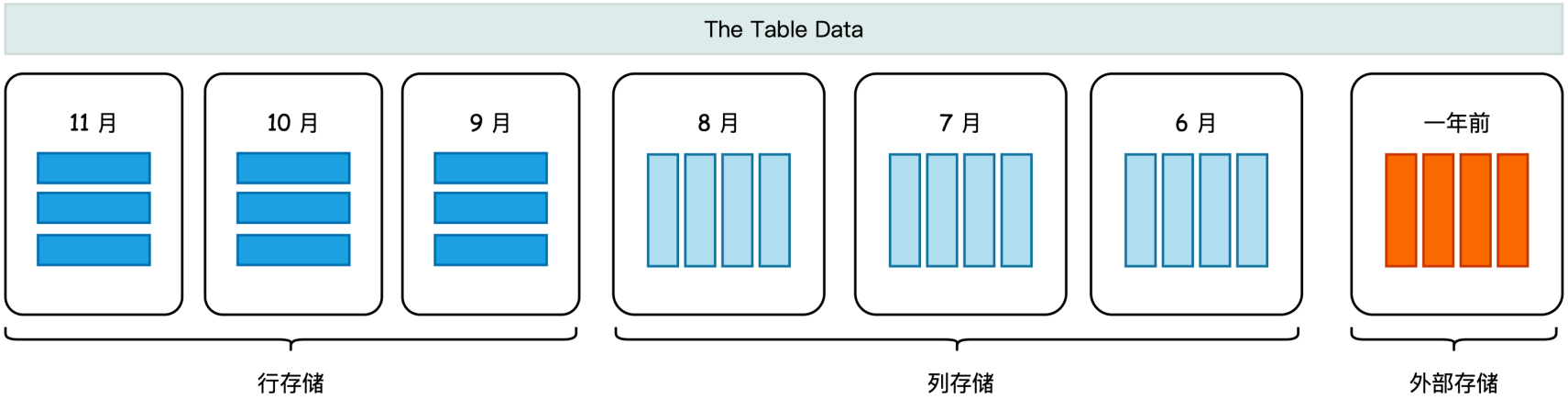
多态存储

Greenplum 支持多态存储，即单张用户表，可以根据访问模式的不同而使用不同的存储方式存储不同的分区。例如根据数据的新、旧程度决定将数据存储至本地硬盘还是以外部表的方式存储在 HDFS 或者是 S3 中。Greenplum 提供以下存储方式:

- 堆表 (Heap Table)：默认存储方式，同时也是 PostgreSQL 的默认存储方式。支持高效的更新和删除操作，通常用于 OLTP 。
- Append-Optimized 表：以追加的方式写入数据，有着极高的写入性能，通常用于存储数据仓库中的事实数据，不适合做频繁的更新、删除操作。
- Append-Optimized, Column Oriented 表：即 AOCO 表，在 Append-Optimized 的基础之上按列进行存储，可对其使用不同的压缩算法进行压缩，对聚合查询有着天然的优势。
- 外部表：外部表的数据存储在外部，Greenplum 仅管理其元数据，支持多种外部数据源，例如 S3、HDFS、文件、Gemfire，以及多种数据格式譬如 Text、CSV、Avro、Parquet 等。

GREENPLUM DATABASE®

多态存储



如上所示，可以根据数据访问频率以及数据量这两个维度来选择不同的存储方式，并且在逻辑上仍然是同一张表。

GREENPLUM DATABASE®

Greenplum 分布式查询优化器

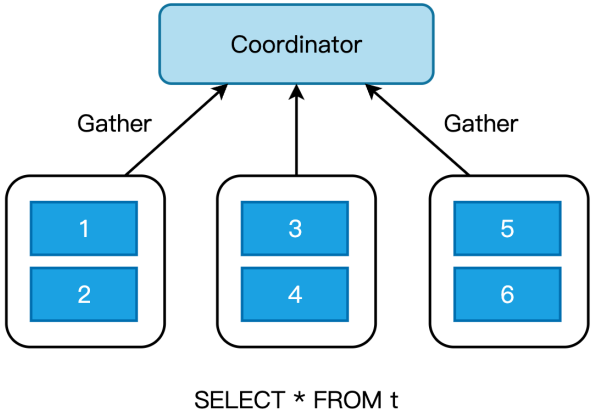
Motion

GREENPLUM DATABASE

分布式查询优化器

当我们插入数据时，Coordinator 将会根据分布键以及分布策略将数据分布到不同的节点中去。那么在查询时，就需要各个节点将数据处理完毕后发送至 Coordinator 节点并返回给客户端用户。

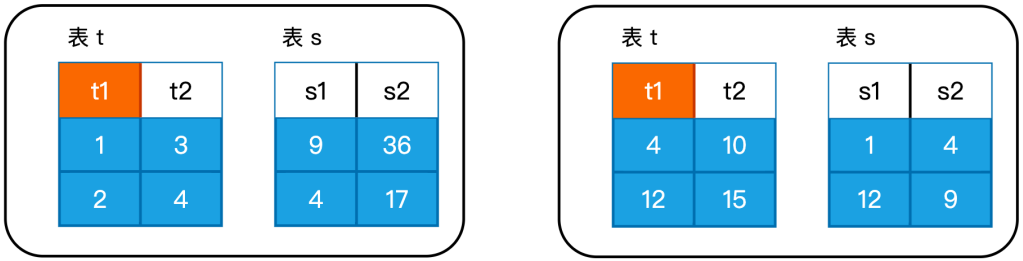
- 对于普通查询，只需要将 Segment 上的数据汇总即可，如果有 filter，则在 segment 上执行条件过滤



SELECT * FROM t

- 对于 JOIN，我们需要考虑两张表的分布键以及分布策略。若分布键和分布策略不同，就需要对数据进行节点间移动

CREATE TABLE t (t1 int, t2 int) DISTRIBUTED BY (t1);
CREATE TABLE s (s1 int, s2 int) DISTRIBUTED RANDOMLY;



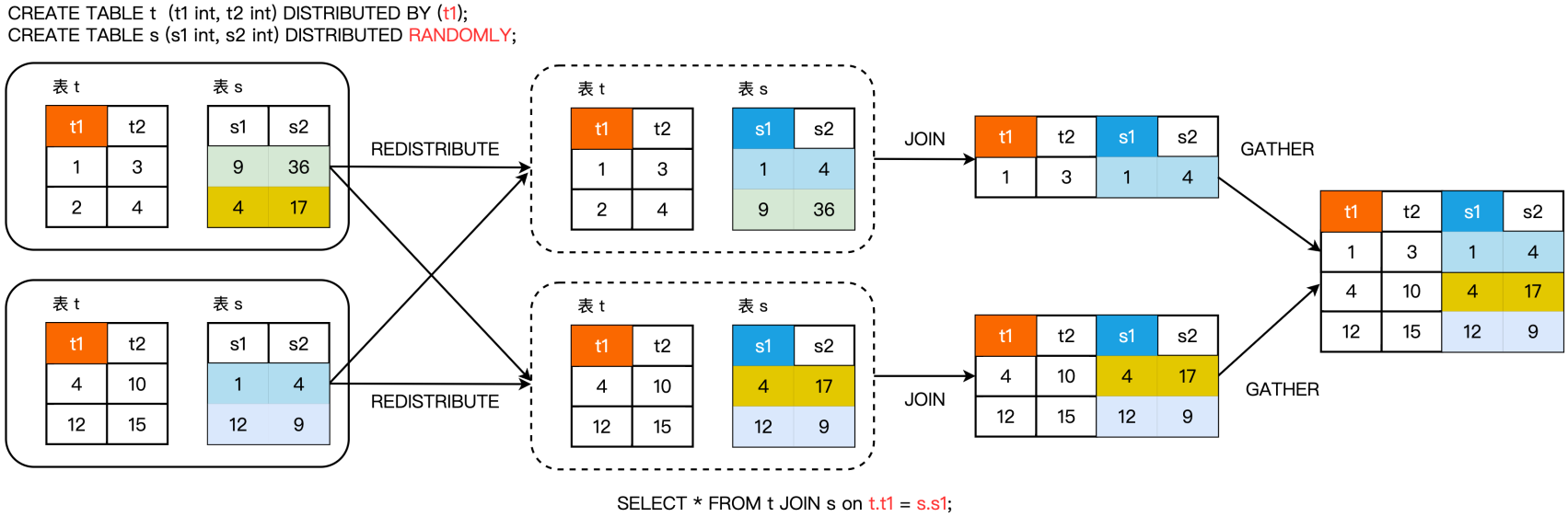
SELECT * FROM t JOIN s on t.t1 = s.s1;

GREENPLUM DATABASE®

Motion

由于数据是根据某种分布策略分散的存储在集群中的各个节点，那么进行查询时，就必然存在数据在各个节点间的移动，这既可能发生在 Coordinator 和 Segment 之间，也可能发生在 Segment 和 Segment 节点之间。

因此，Greenplum 引入了 Motion 算子，来实现数据在不同节点间的传输，为其它算子（如 Hash Join、Sort）隐藏 MPP 架构，使得绝大多数算子不用关心是在集群上执行还是在单机上执行。



GREENPLUM DATABASE®

Motion

- Gather Motion
- Redistribute Motion
- Broadcast Motion

Experiment Environment

create table t (t1 int, t2 int) distributed by (t1);

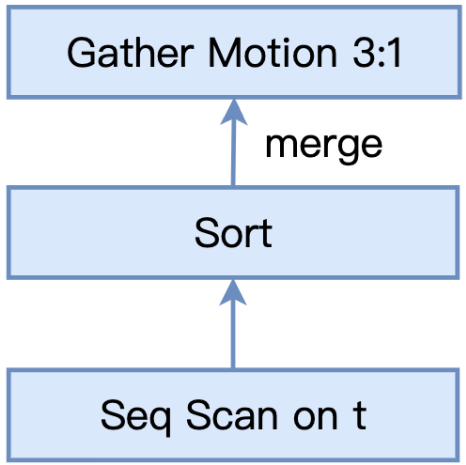
create table s (s1 int, s2 int) distributed randomly;

GREENPLUM DATABASE

Gather Motion

Gather Motion 用于收集 Segments 所发送的所有数据，不一定是最终结果数据，Coordinator 可能需要对收集的数据进行进一步地加工和处理，例如对 Segments 所发来的有序 Tuples 进行 Merge 操作。

```
postgres=# explain (costs off) select * from t order by t1;
                QUERY PLAN
-----------------------------------------
 Gather Motion 3:1  (slice1; segments: 3)
   Merge Key: t1
   ->  Sort
         Sort Key: t1
         ->  Seq Scan on t
 Optimizer: Postgres query optimizer
(6 rows)
```

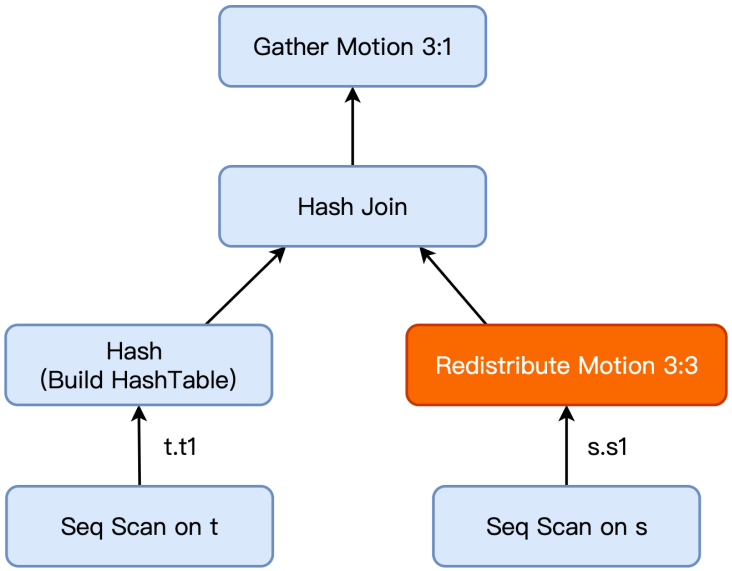


GREENPLUM DATABASE

Redistribute Motion

Redistribute Motion 将数据根据某一个或多个字段对数据进行哈希重分布，目的在于完成诸如连接(JOIN)、聚合(Agg) 等操作。

```
postgres=# explain (costs off) select * from t join s on t.t1 = s.s1;
                        QUERY PLAN
-----------------------------------------------------------
 Gather Motion 3:1  (slice2; segments: 3)
   ->  Hash Join
         Hash Cond: (s.s1 = t.t1)
         ->  Redistribute Motion 3:3  (slice1; segments: 3)
               Hash Key: s.s1
               ->  Seq Scan on s
         ->  Hash
               ->  Seq Scan on t
 Optimizer: Postgres query optimizer
(9 rows)
```

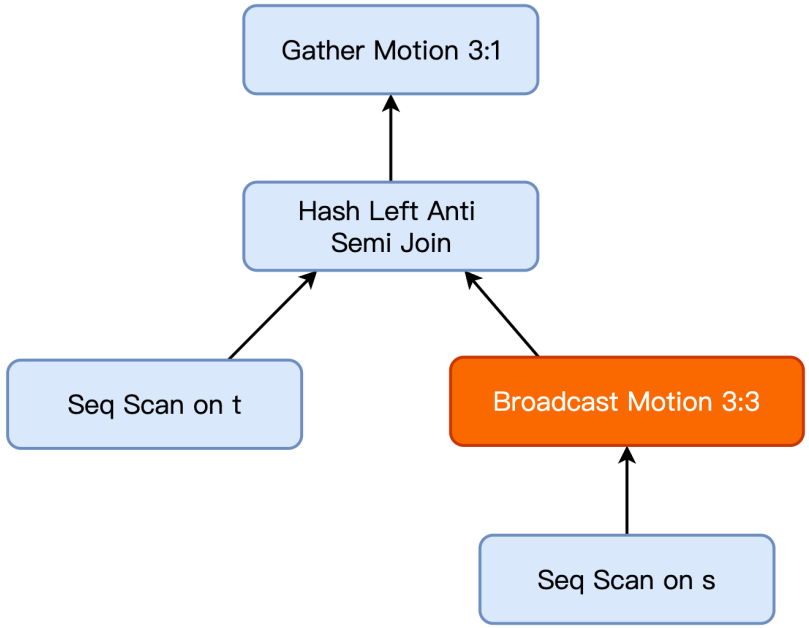


GREENPLUM DATABASE

Broadcast Motion

所有的 Segments 将自身数据发送给其他所有 Segments，这样每一个 Segment 实例都有表的一份完整拷贝。

```
postgres=# explain (costs off) select * from t where t1 not in
(select s1 from s);
                        QUERY PLAN
-----------------------------------------------------------
 Gather Motion 3:1  (slice2; segments: 3)
   ->  Hash Left Anti Semi (Not-In) Join
       Hash Cond: (t.t1 = s.s1)
       ->  Seq Scan on t
       ->  Hash
             ->  Broadcast Motion 3:3  (slice1; segments: 3)
                   ->  Seq Scan on s
 Optimizer: Postgres query optimizer
(8 rows)
```



GREENPLUM DATABASE

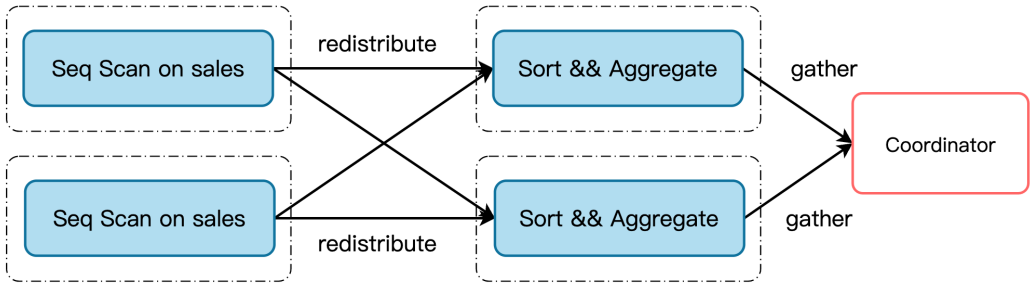
Multi-Stage Aggregate

create table sales (order_id int, brand int, quantity int) distributed by(order_id);

select brand, avg(quantity) from sales group by brand;

一阶段聚集

```
postgres=# explain (costs off) select brand, avg(quantity) from sales group by brand;
                        QUERY PLAN
------------------------------------------------------------------------
 Gather Motion 3:1  (slice2; segments: 3)
   ->  GroupAggregate
         Group Key: brand
         ->  Sort
               Sort Key: brand
               ->  Redistribute Motion 3:3  (slice1; segments: 3)
                     Hash Key: brand
                     ->  Seq Scan on sales
```



Warning

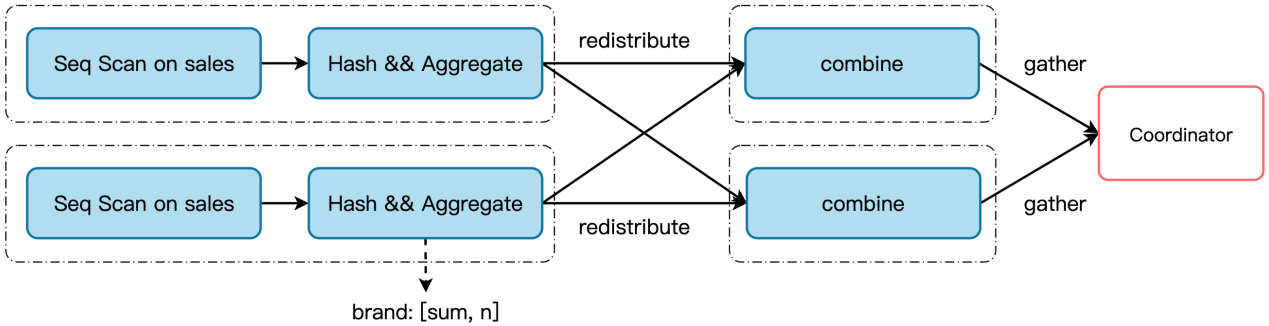
- 我们需要对所有数据进行重分布，网络开销昂贵
- 若分组数量远小于集群节点数量，则会造成严重的计算倾斜

GREENPLUM DATABASE®

Multi-Stage Aggregate

二阶段聚集

```
postgres=# explain (costs off) select brand, avg(quantity) from sales group by brand;
                        QUERY PLAN
------------------------------------------------------------
 Gather Motion 3:1  (slice2; segments: 3)
   ->  HashAggregate
         Group Key: sales.brand
         ->  Redistribute Motion 3:3
               Hash Key: sales.brand
               ->  HashAggregate
                     Group Key: sales.brand
                     ->  Seq Scan on sales
```



- 此时，只需要对在各个节点聚合后的数据进行重分布
- 但需要额外实现 combine() 方法

GREENPLUM DATABASE®

Greenplum 分布式执行器

QD/QE/火山模型/Gang

GREENPLUM DATABASE®

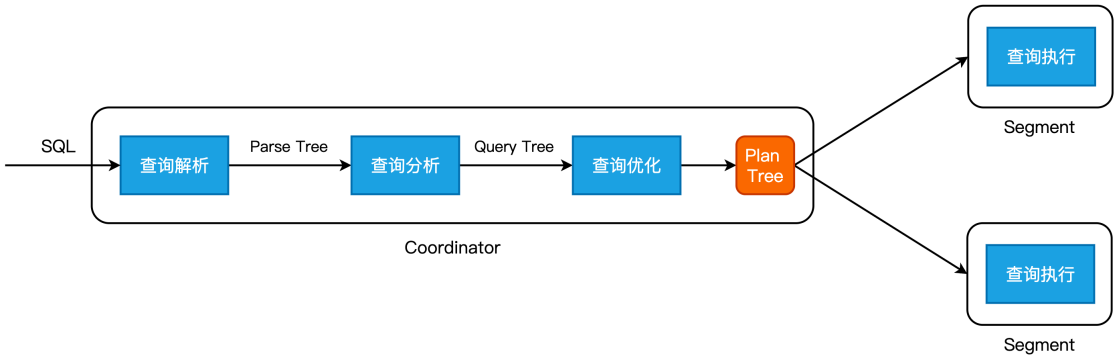
QD && QE

Greenplum，或者说 PostgreSQL 是进程模型，而不是类似于 MySQL 的线程模型。

主进程 postmaster 是整个数据库实例的总控进程，负责启动和关闭数据库实例。当客户端和 Coordinator 建立连接时，postmaster 会 fork 出一个子进程来为该连接提供服务。

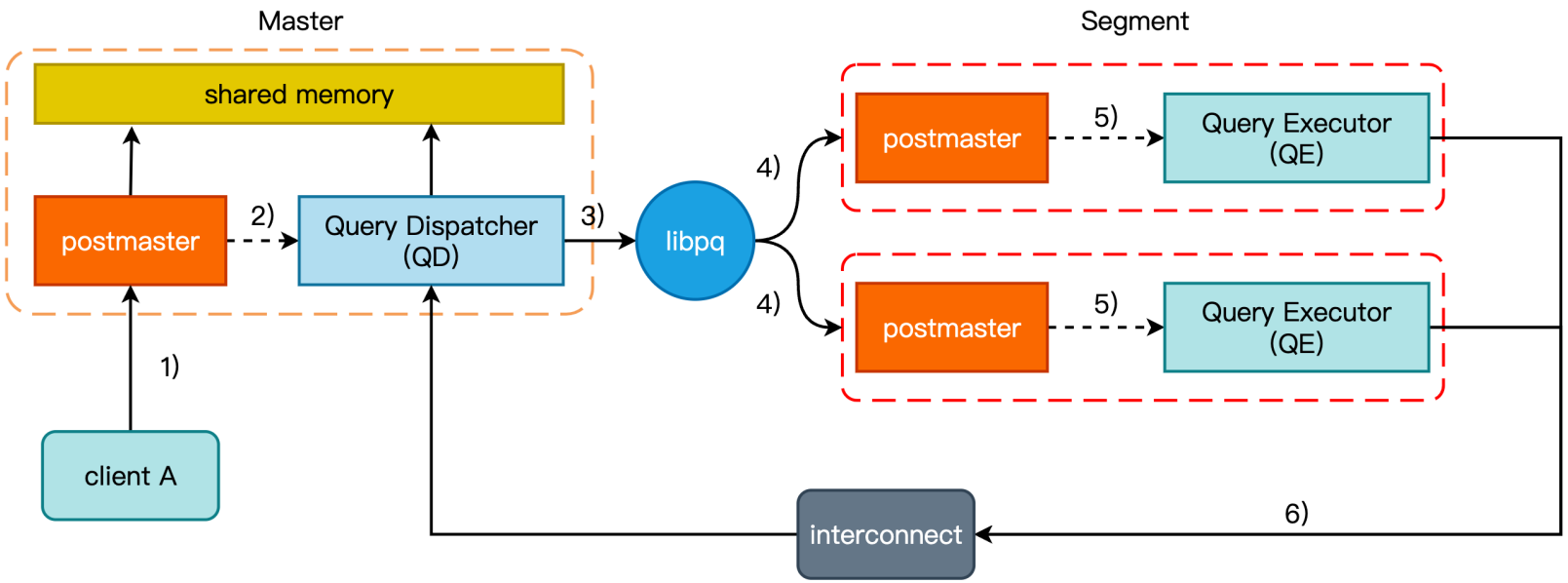
Coordinator 节点上负责处理用户查询请求的进程称为 QD (Query Dispatcher) 进程。当 QD 进程收到客户的 SQL 时，就会对其进行解析、重写和优化，并将分布式查询计划发送给 Segment 节点进行执行，并将最终结果返回给客户端。

Segment 节点上负责执行 QD 分发来的查询任务的进程称为 QE (Query Executor) 进程，递归遍历 QD 发来的计划树，对每一个节点按照拉模型 (火山模型) 进行执行。



GREENPLUM DATABASE®

QD && QE



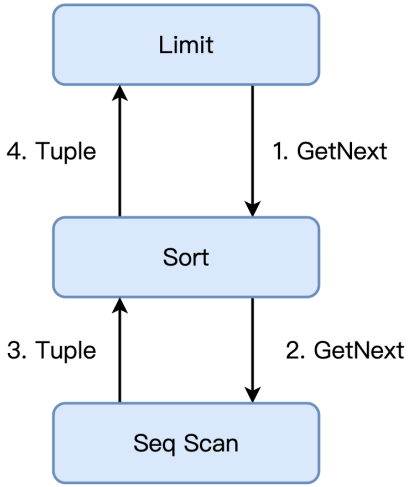
GREENPLUM DATABASE

火山模型

火山模型，或者说拉模型，是指从最顶层的输出节点开始，不断从下层节点拉取数据，一种自顶向下的执行方式。最常见的拉模型是 Tuple-At-A-Time，即每次从下层拉取一个元组进行处理。Greenplum、PostgreSQL、MySQL 以及 Oracle 等主流数据库均采用拉模型。

拉模型的每个算子都实现了从下层节点获取一条元组的 GetNext 函数，每次调用该函数都会从下层节点返回一条元组或者 EOF 的 NULL 指针。上层节点不断地调用 GetNext 函数从下层节点获取数据，直至数据全部获取完毕。

```
postgres=# explain select * from t order by t1 limit 1;
                        QUERY PLAN
-----------------------------------------------------------------
 Limit  (cost=42.10..42.10 rows=1 width=8)
   ->  Sort  (cost=42.10..47.45 rows=2140 width=8)
         Sort Key: t1
         ->  Seq Scan on t  (cost=0.00..31.40 rows=2140 width=8)
(4 rows)
```



GREENPLUM DATABASE®

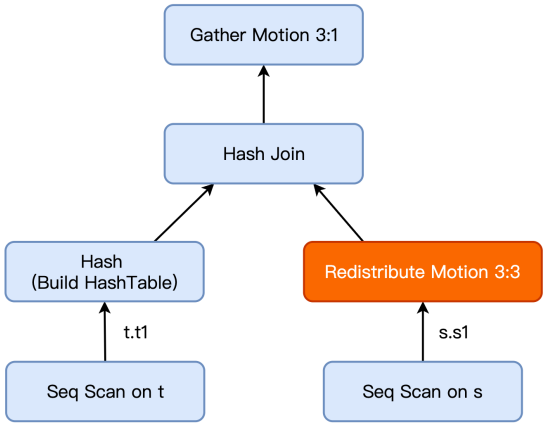
Slice && Gang

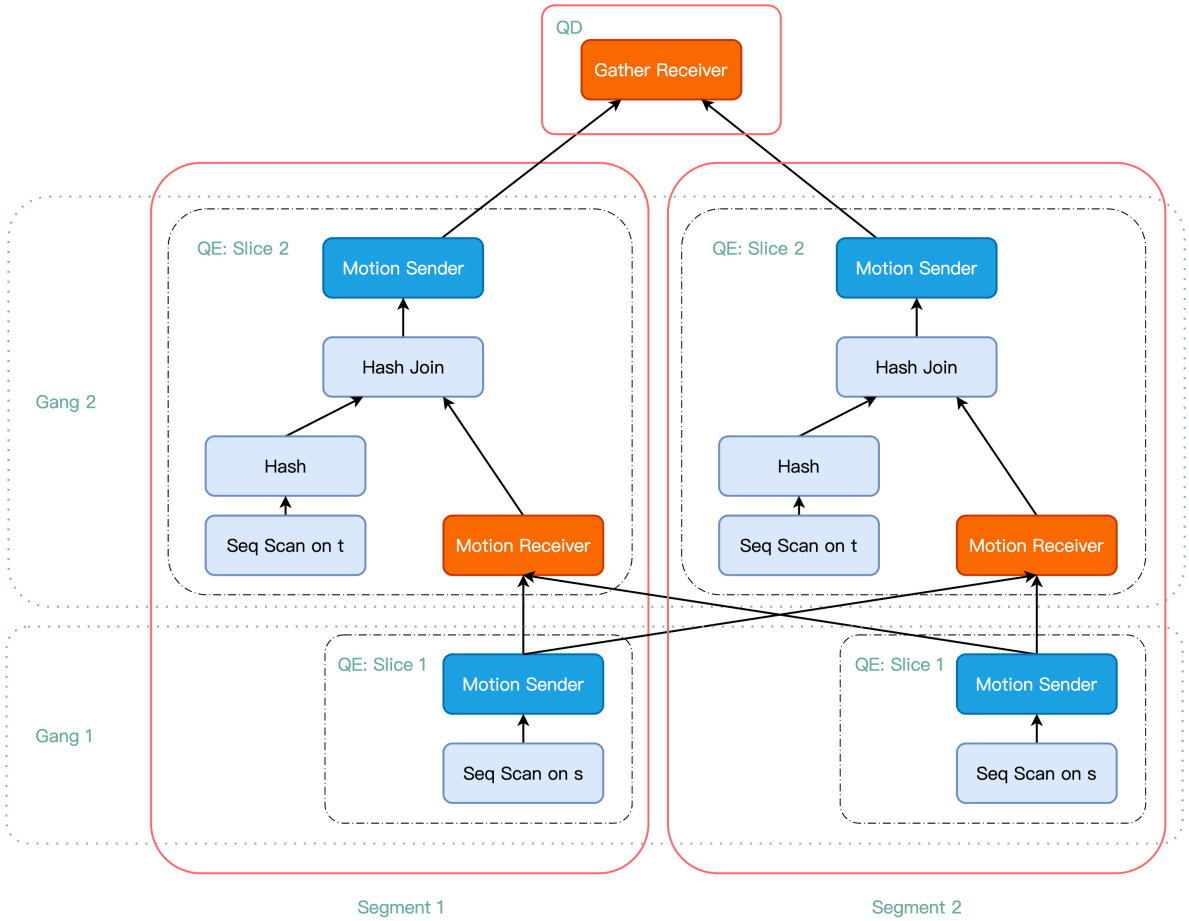
Slice：为了提高查询执行并行度和效率，Greenplum 把一个完整的分布式执行计划分割成多个 Slice，每个 Slice 负责查询计划的一部分。划分 Slice 的边界为 Motion，每遇到一个 Motion 则一刀将 Motion 切成发送方和接收方。同时，每个 Slice 都由一个 QE 进程进行处理。通常来说，Slice 的数据量为 Motion 的数据量再加 1。

Gang：在不同 Segment 上执行同一个 Slice 的所有 QEs 进程称为 Gang，Gang 用来表示一组进程

GREENPLUM DATABASE®

Slice && Gang





GREENPLUM DATABASE®

微信技术讨论群
微信搜索添加“gp_assistant”
加入技术讨论

微信公众号
搜索添加“Greenplum中文社区”
技术干货、行业热点、活动预告

Thank You