阿里巴巴 Alibaba.com.cn

Greenplum介绍

唐成 - 2011.02.17

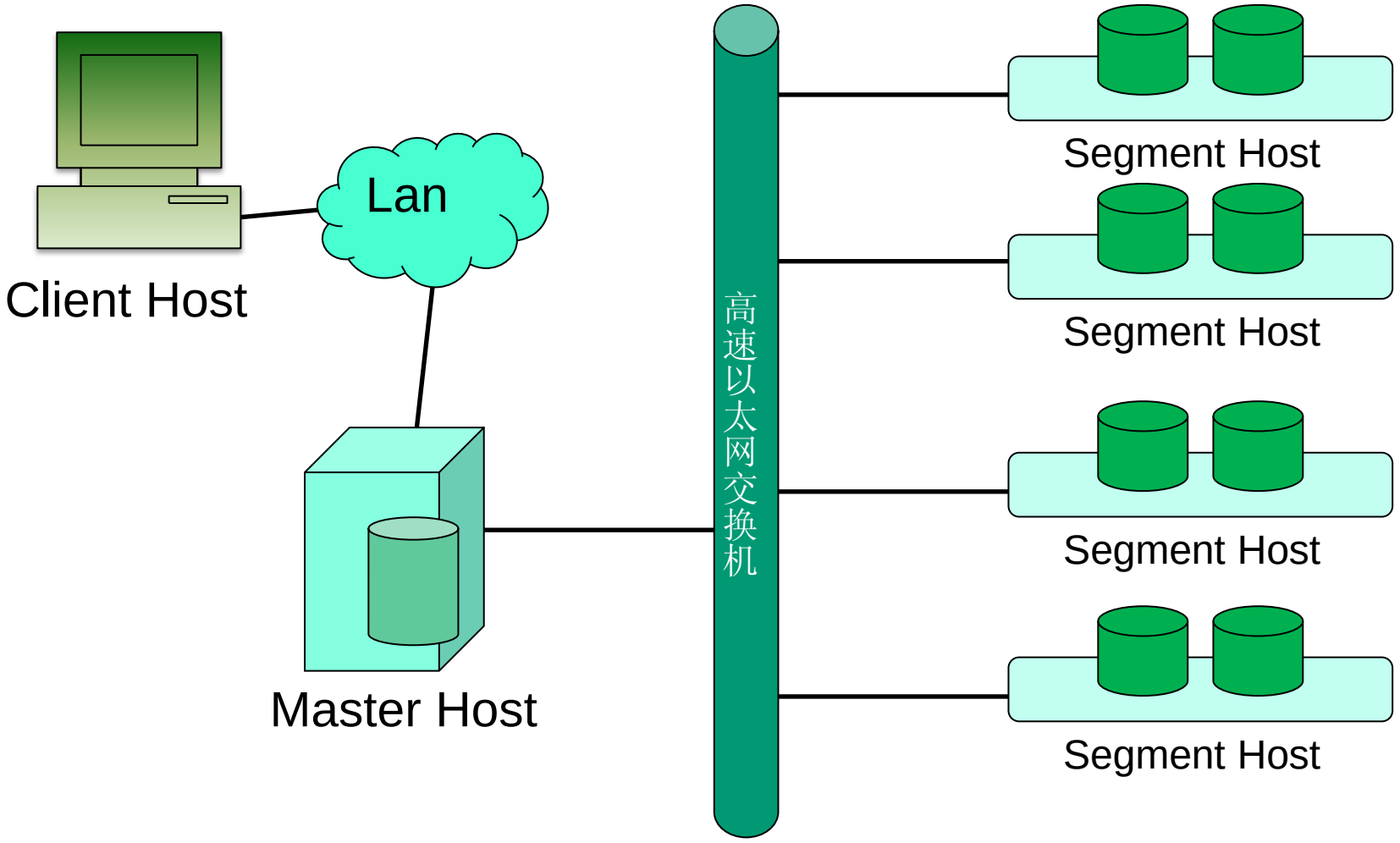
汇报提纲

- Greenplum VS hadoop
- Greenplum架构
- Greenplum的高可用方案
- GP分布式数据库功能介绍
- 理解GP的查询处理

Greenplum VS hadoop

比较项	Greenplum	Hadoop+hive
软件性质	商业软件	开源
集群规模	一般在100台以下	可以到上万台。
性能	在100台以下时，性能比hadoop好。 单个SQL可以做到秒级别	集群规模越大，总体性能越好。 单个SQL最少也有数十秒。
SQL的支持程度	支持完善，几乎所有PostgreSQL支持的SQL，gp都支持。	支持有限的SQL，查询支持子查询，但不支持窗口函数。大部分dml都不支持，只支持append。
稳定性	有较多的bug。	比较稳定。

Greenplum架构图



Greenplum架构: Master介绍

Master服务器是外面用户访问greenplum的入口。用户都是连接master服务器的，对于外部用户来说，他并不与segment host服务器发生任何关系，外部用户的网络只需要与master服务器连通就可以了，不需要访问segment host服务器。
所有的用户连接都是直接连接到master服务器上的。

Greenplum数据库是基于PostgreSQL数据库的，所以可以用PostgreSQL数据库的工具来连接Greenplum数据库，如java程序可以使用PostgreSQL的jdbc驱动来访问Greenplum数据库，也可以使用psql工具或pgadminII来管理Greenplum。

Greenplum架构: Master介绍

Greenplum的Master数据库也是一个被改造过的PostgreSQL数据库，它包含了整个分布 式数据库中的所有元数据，如表结构定义、索引等等。但其并不存储实际的数据，实际的数据是存储在segment 数据库的。

master服务器接受从用户发来的连接，并做用户验证，接收用户发来的sql，生成分布式执行计划，再把分布执行计划分发到segment上执行，接收segment返回的数据，最后返回给用户。

Greenplum架构: Segment介绍

Segment是数据的实际存储的地方，也是一个经过改造过的PostgreSQL数据库。它做实际的数据处理工作。Greenplum建议在Segment host上建多个Segment数据库，数量等于实际的CPU的core数。

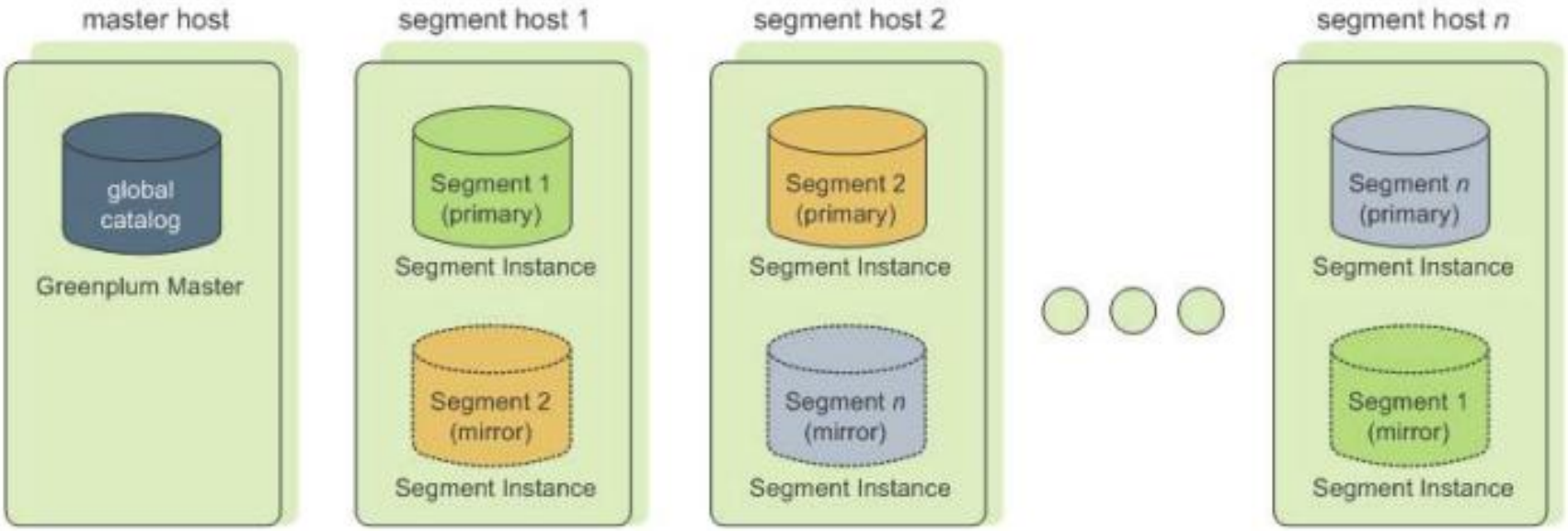
Greenplum架构: 内部网络

Segment host与master是通过greenplum的内部网络互联起来的，外部用户不需要访问这个内部网络的。
Segment 与Segment之间是有网络连接的，所以Segment之间可以直接交互数据的。

Greenplum默认使用UDP协议，不过我们发现UDP有时不稳定，我们一般都使用TCP协议。使用TCP协议，greenplum最多1000个segment。

Greenplum中的高可用方案

Segment的mirror



当配置了segment mirror，当segment primary不能写的时候，greenplum会自动切换到mirror。当master不能连接到一个segment instance时，会把这个instance标记为invalid。

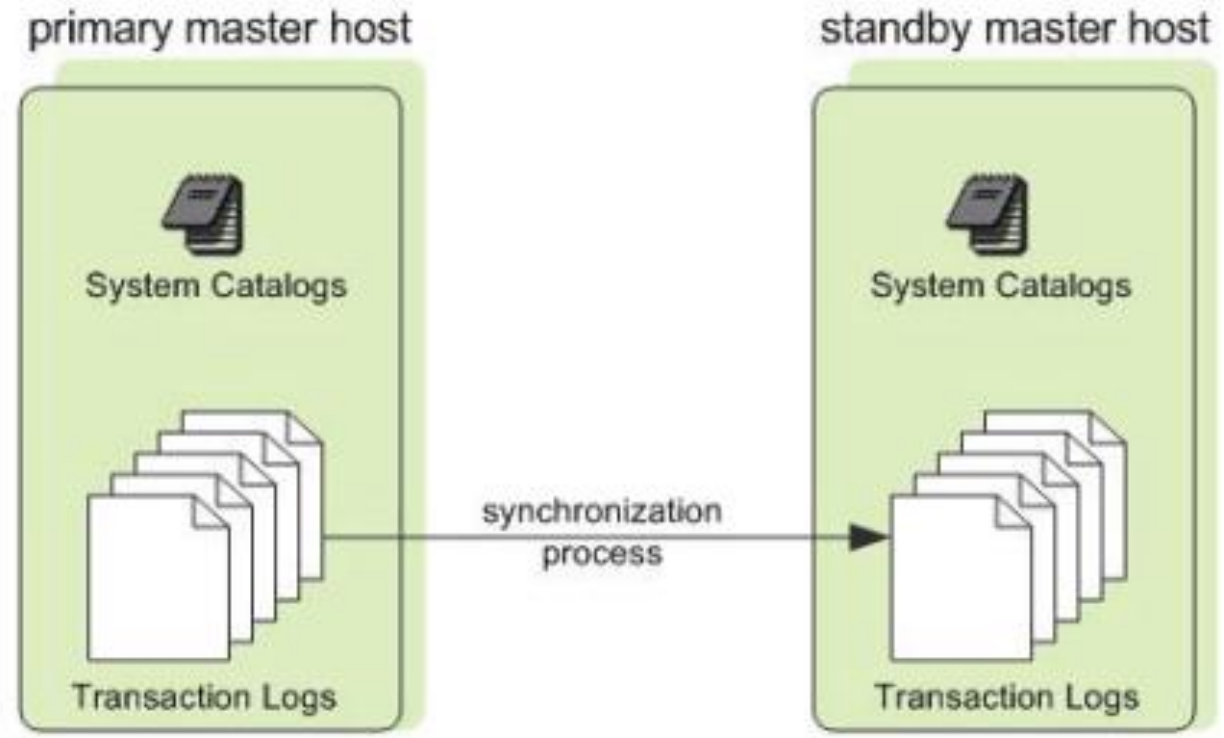
Greenplum中的高可用方案

默认情况下，greenplum的失败操作模式是“read-only"模式，也就是说如果一个segment坏了，整个greenplum会变成只读，不能写了。如果模式是“continue”模式时，一个segment坏了的时候，数据库仍然可以继续工作。但由于segment的primary与mirror端的数据不同步了，所以恢复的时候需要花比较长的时间。对于Greenplum 3.X的版本，恢复时，需要把好的节点上的所有数据都copy到坏的机器上。而Greenplum4.0版本增加了功能，当备份节点坏的时候，主节点可以把增量数据记下来，这样当备份节点的主机恢复时，只需要恢复增量数据就可以了。
要让原先已offline的节点再加入集群中，需要重启集群。

Greenplum中的高可用方案

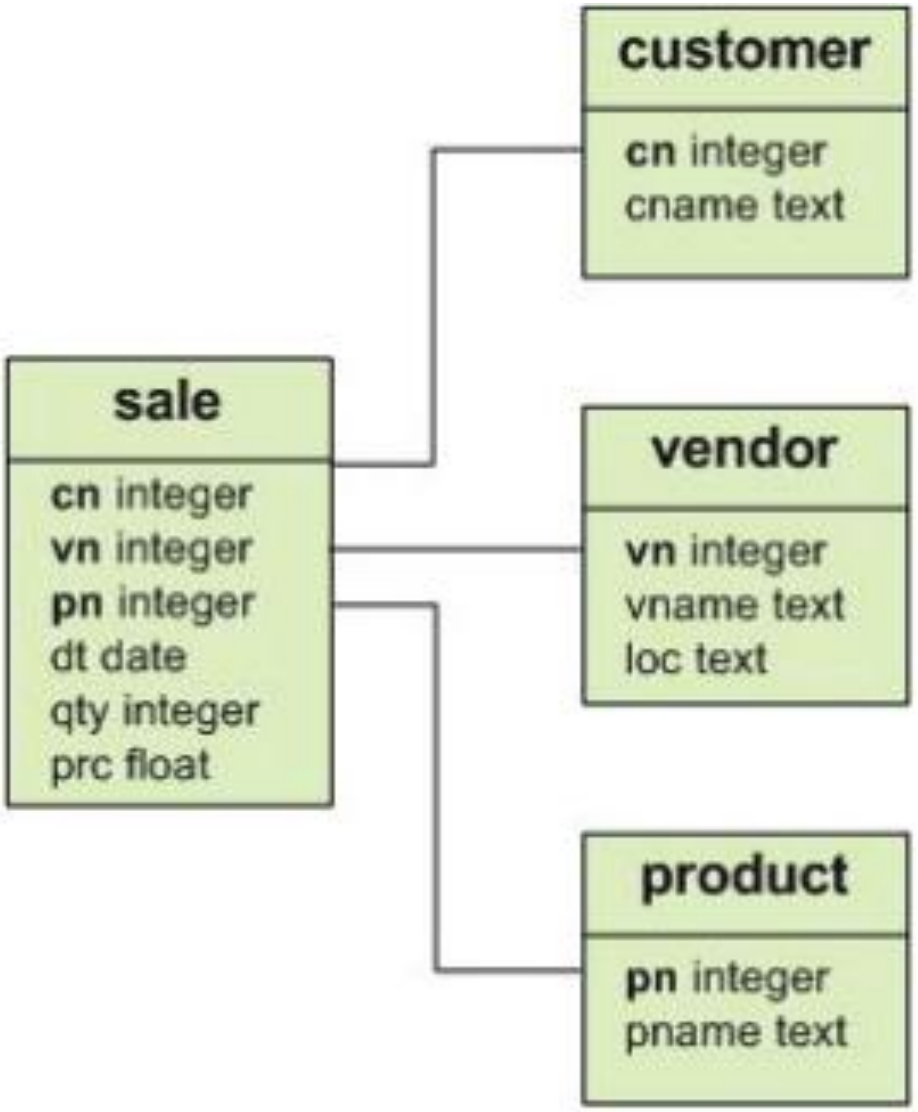
对于Greenplum 3.X的版本，segment 的primary与mirror之间是做的逻辑同步，mirror端的数据库实际上也是可以读写的。而Greenplum4.0版本后，primary与mirror实际上是物理同步，这时mirror一直处于恢复状态，不能读也不能写。

高可用之Master Mirroring

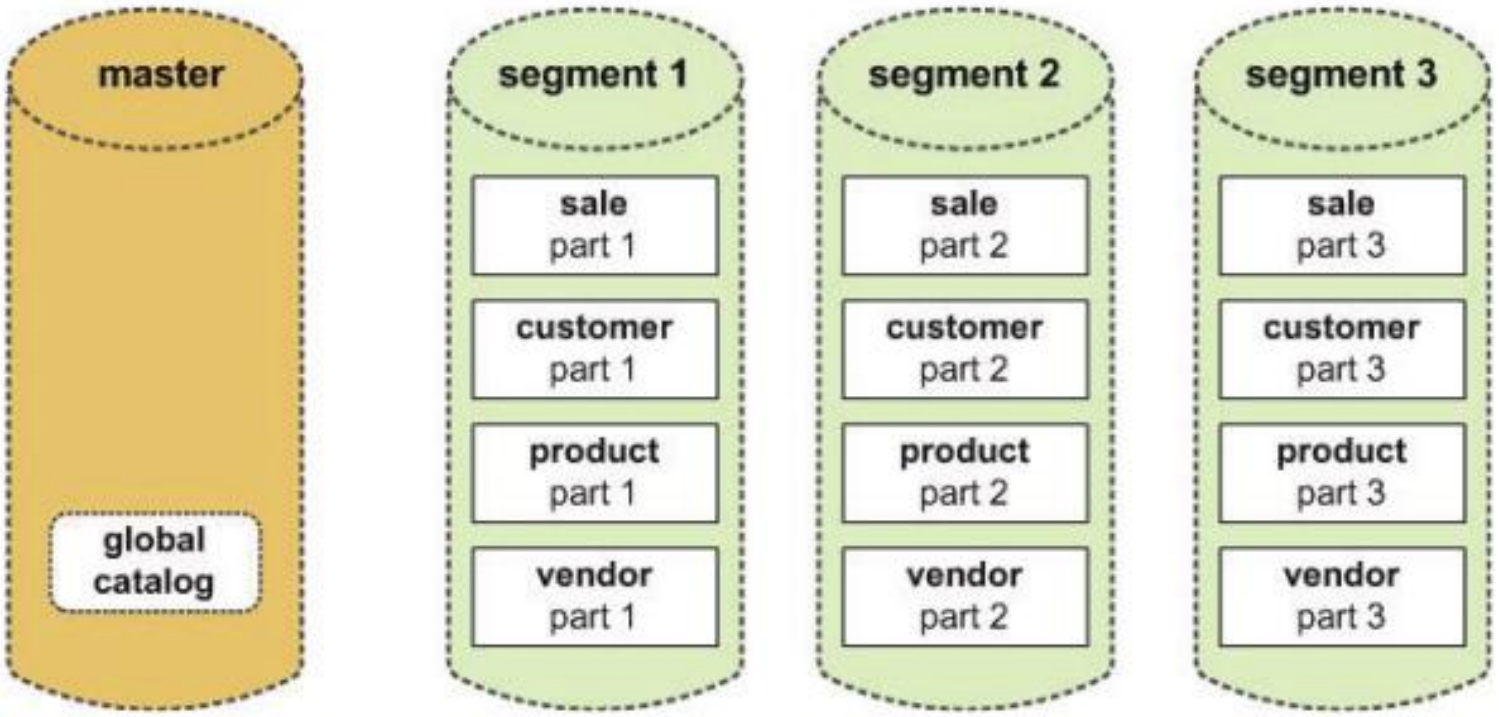


对于Greenplum Master的primary与mirror之间的同步就是使用PostgreSQL的日志同步方案。master的mirror可以在建库时建，也可以在建完greenplum后再添加。

理解greenplum分布式数据库



理解GP分布式数据库


master
global catalog
segment 1
sale part 1
customer part 1
product part 1
vendor part 1
segment 2
sale part 2
customer part 2
product part 2
vendor part 2
segment 3
sale part 3
customer part 3
product part 3
vendor part 3

理解GP的数据分布策略

Hash分布：按分布键对数据时行hash分布，这个hash分布算法没有公布，只有greenplum内部知道数据是如何hash分布的。

随机分布：数据随机分布在数据库，每次查询都会查询所有的 segment。

GP的SQL支持程度

基本上绝大多数PostgreSQL支持的SQL，在greenplum都支持，如常见的建表、建索引的ddl，以及一般的dml语句。
Greenplum3.X不支持表空间。4.0提供了支持tablespace功能。
不支持trigger。

建表语句多了distributed by 子名外， 其它的SQL语法基本上都与PostgreSQL一样:

```
CREATE TABLE products
(name varchar(40),
prod_id integer,
supplier_id integer)
DISTRIBUTED BY (prod_id);
```

GP的表增强

greenplum除支持普通的表外，还增加了PostgreSQL中没有的表类型:

append-only table:

```
CREATE TABLE bar (a int, b text) WITH
(appendonly=true)  DISTRIBUTED BY (a);
```

column-oriented table:

```
CREATE TABLE bar (a int, b text)
WITH (appendonly=true, orientation=column)
DISTRIBUTED BY (a);
```

append-only的表支持压缩:

```
CREATE TABLE foo (a int, b text)
WITH (appendonly=true, compresstype=zlib,
compresslevel=5);
```

GP使用人性化的partion语法支持分区表

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
( START (date '2008-01-01') INCLUSIVE
END (date '2009-01-01') EXCLUSIVE
EVERY (INTERVAL '1 day') );
```

GP使用人性化的partion语法支持分区表

```
CREATE TABLE sales (id int, date date, amt
decimal(10,2))
DISTRIBUTED BY (id)
PARTITION BY RANGE (date)
( PARTITION Jan08 START (date '2008-01-01')
INCLUSIVE ,
PARTITION Feb08 START (date '2008-02-01')
INCLUSIVE ,
PARTITION Mar08 START (date '2008-03-01')
INCLUSIVE ,
PARTITION Apr08 START (date '2008-04-01')
INCLUSIVE ,
END (date '2009-01-01') EXCLUSIVE );
```

GP使用人性化的partion语法支持分区表

Defining Numeric Range Table Partitions

```
CREATE TABLE rank (id int, rank int, year int, gender
char(1), count int)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
( START (2001) END (2008) EVERY (1),
DEFAULT PARTITION extra );
```

GP使用人性化的partion语法支持分区表

Defining List Table Partitions

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

GP的表的数据可以重分布

```
ALTER TABLE sales SET DISTRIBUTED BY
(customer_id);

ALTER TABLE sales SET DISTRIBUTED
RANDOMLY;

ALTER TABLE sales SET WITH
(REORGANIZE=TRUE);
```

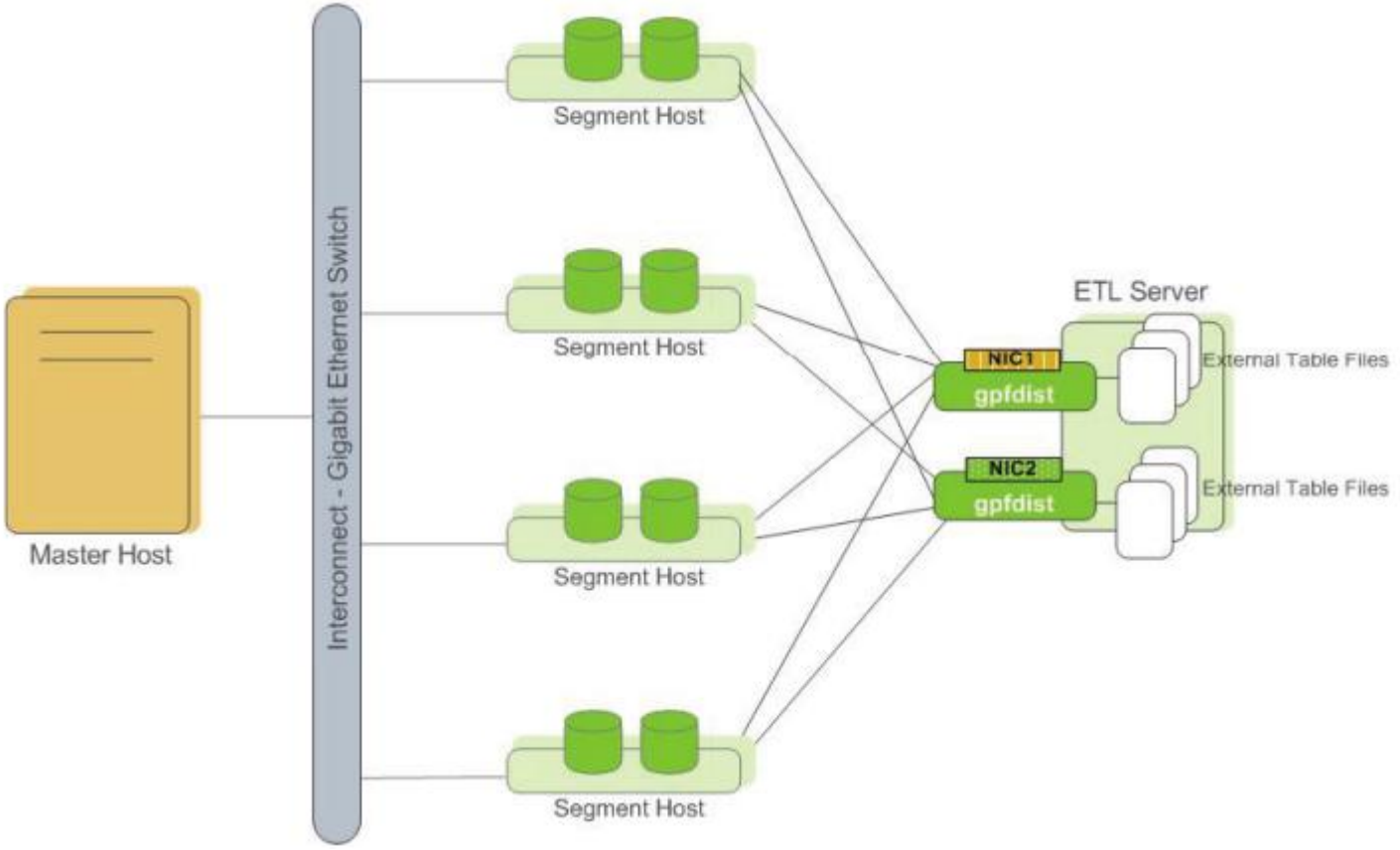
重分布代价比较大，一般是通过迁移segment节点的方式完成加节点。

GP支持bitmap索引

PostgreSQL不支持bitmap 索引，greenplum支持bitmap索引。

```
CREATE INDEX gender_bmp_idx ON employee
USING bitmap
(gender);
```

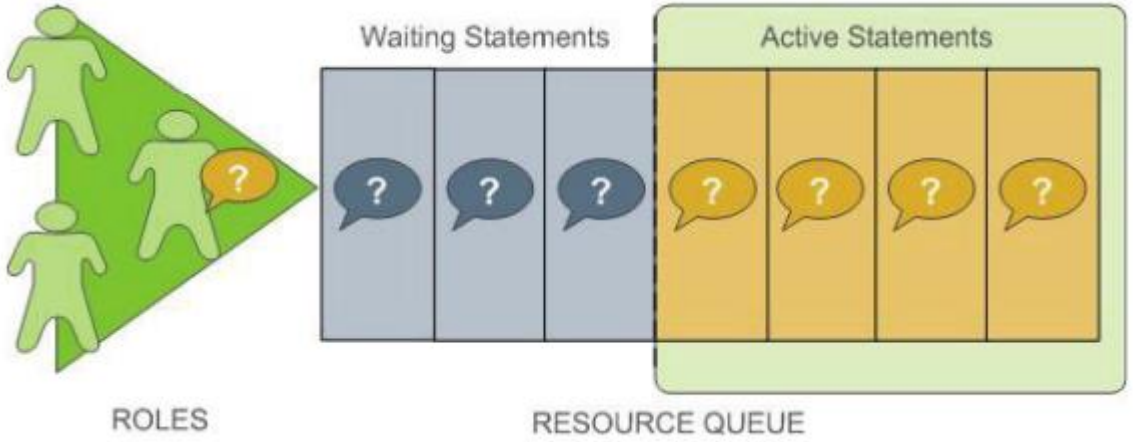
并行数据装载



Greenplum Database's parallel file server(gpfdist),可以达到1小时装载2T数据。

GP的工作负载的资源控制

GP提供了对工作负载和资源控制的功能。
在GP可以建一个资源队列（resource queue)，然后把用户加入到这个队列中，然后就可以控制:
1. 所有活动的SQL的cost值最多是多少？
2. 活动的SQL最多是多少个
3. 控制活动的SQL的优先级（4.0的新功能）



GP的查询处理

用户提交一个SQL到master，master解析这个SQL，生成一个分布式的执行计划，然后把这个分布式的执行计划分发到各个segment上，然后segment执行它自己的特定数据集的本地数据库业务。

所有的数据库操作，如表扫描、表连接（joins)、聚集（ aggregations），排序，这些操作都会在所有的segment上并行执行。每个segment执行这些操作时都不依赖其它的segment。

除了上面这引起典型的数据库操作，Greenplum的数据库有一个额外的操作类型，称为的motion。

motion操作就是把查询处理过程中涉及到的其它节点上的数据在各个节点中做移动。

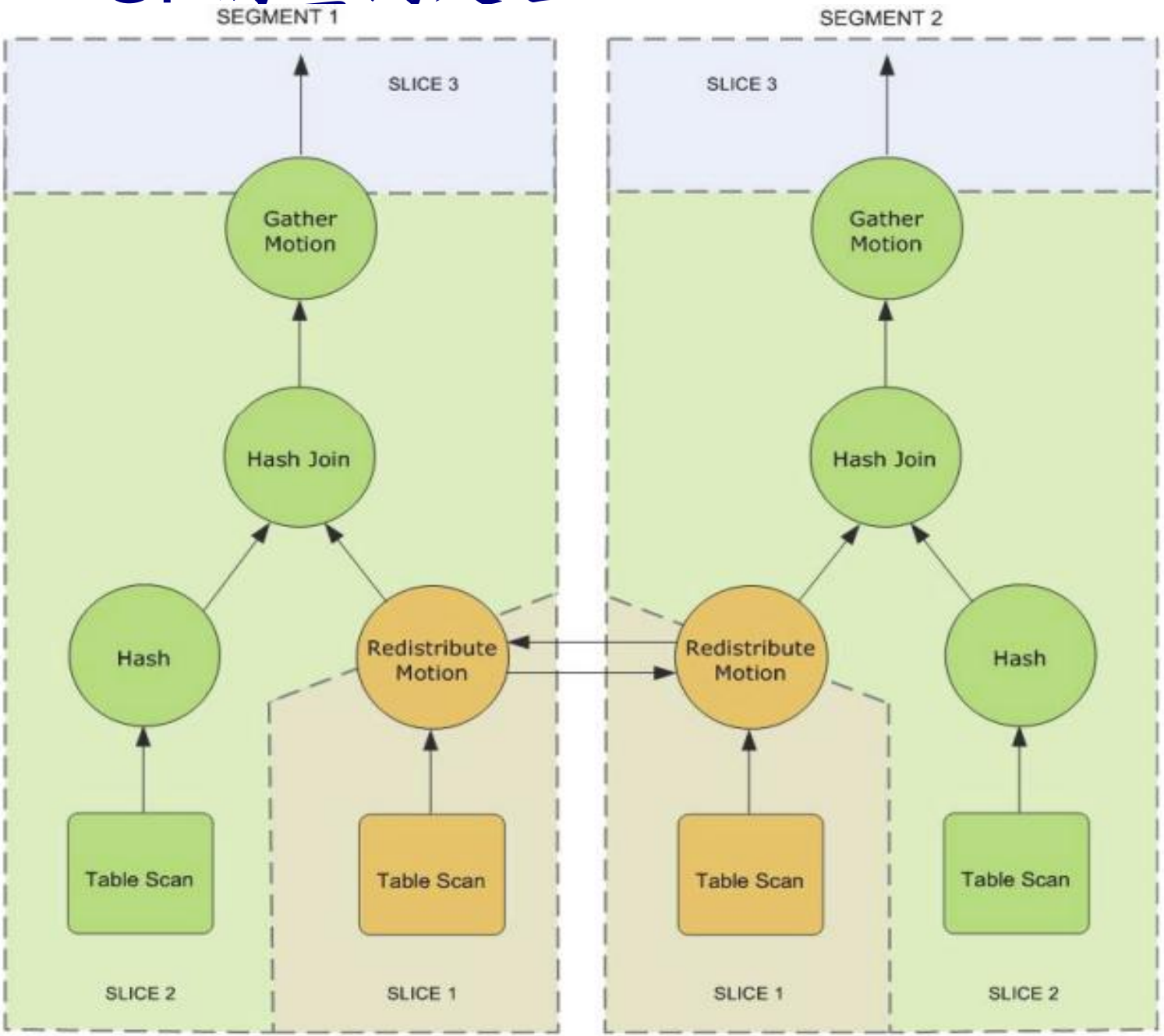
GP的查询处理

为了提高执行的性能，Greenplum把执行计划进行切片(slice)。

例如:

```
SELECT customer, amount
FROM sales JOIN customer USING (cust_id)
WHERE dateCol = '04-30-2008';
```

GP的查询处理

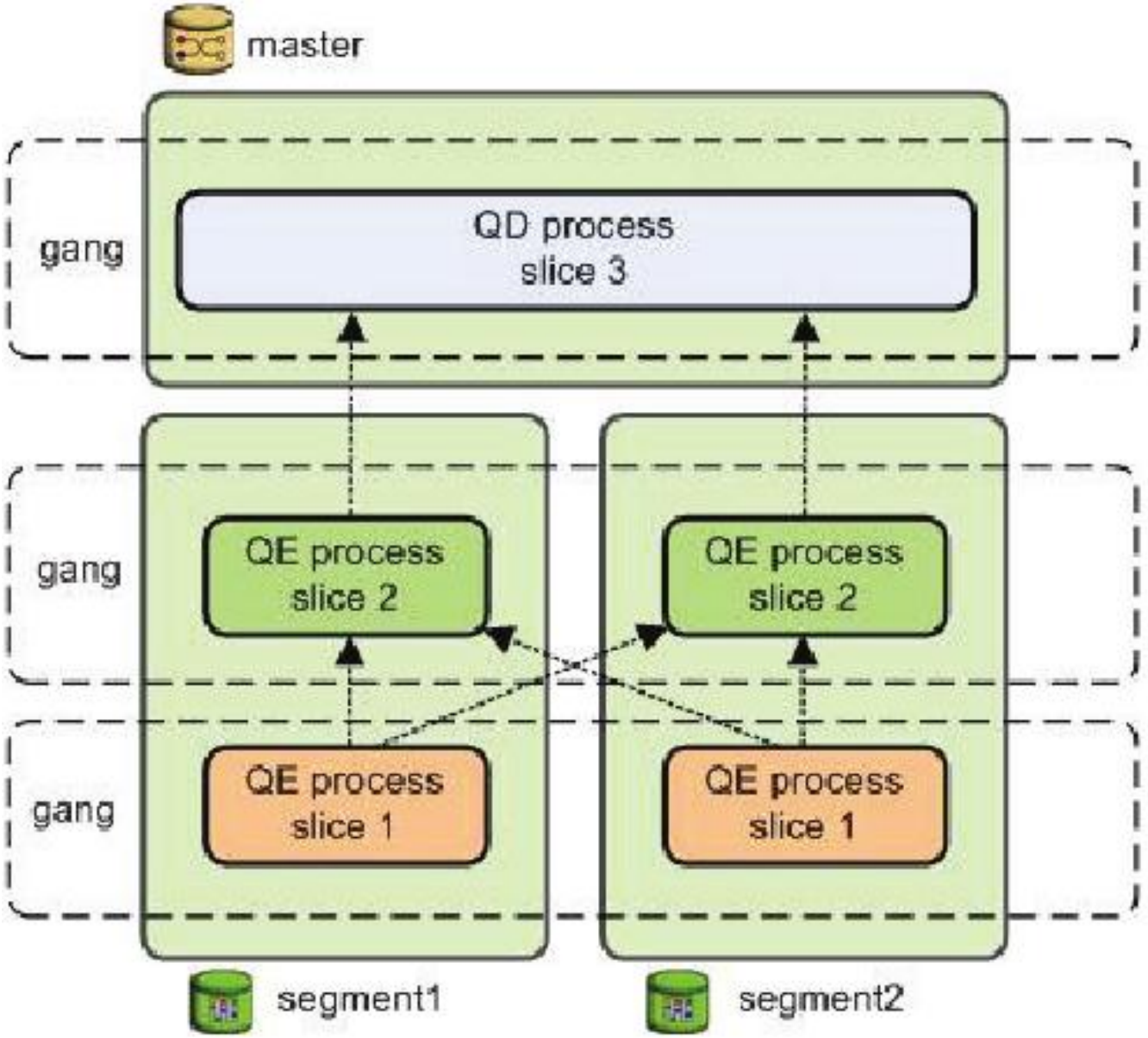

SEGMENT 1
SEGMENT 2
SLICE 3
SLICE 3
Gather Motion
Gather Motion
Hash Join
Hash Join
Hash
Redistribute Motion
Redistribute Motion
Hash
Table Scan
Table Scan
Table Scan
Table Scan
SLICE 2
SLICE 1
SLICE 1
SLICE 2

GP的查询处理

在master上，查询的工作进程叫query dispatcher(QD)
在segment上，查询的工作进程叫query executor (QE)

执行计划的每个切片(slice)至少分配一个工作进程。

GP的查询处理


master
gang
QD process
slice 3
gang
QE process
slice 2
QE process
slice 2
gang
QE process
slice 1
QE process
slice 1
segment1
segment2

GP的执行计划

建测试表:

```
create table t1(id int primary key,cn int,name varchar(40)) distributed by (id);
create table t2(id int primary key,cn int,name varchar(40))
distributed by (id) ;
create table t3(id int primary key,cn int,name varchar(40))
distributed by (id) ;
insert into t1 select
generate_series(1,1000000),generate_series(1,1000000),generate_series(1,1000000);
insert into t2 select
generate_series(1,1000000),generate_series(1,1000000),generate_series(1,1000000);
insert into t3 select
generate_series(1,100),generate_series(1,100),generate_series(1,100);
```

GP的执行计划

```
tangdb=# explain select count(*) from t1;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=13884.48..13884.49 rows=1 width=0)
   ->  Gather Motion 4:1  (slice1; segments: 4)  (cost=13884.40..13884.46 rows=1 width=0)
         ->  Aggregate  (cost=13884.40..13884.41 rows=1 width=0)
               ->  Seq Scan on t1  (cost=0.00..11377.12 rows=250728 width=0)


tangdb=# explain select id,count(*) from t1 group by id;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Gather Motion 4:1  (slice1; segments: 4)  (cost=16391.68..28928.08 rows=250728 width=4)
   ->  HashAggregate  (cost=16391.68..28928.08 rows=250728 width=4)
         Group By: id
         ->  Seq Scan on t1  (cost=0.00..11377.12 rows=250728 width=4)
```

GP的执行计划

```
tangdb=# explain select name,count(*) from t1 group by name;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Gather Motion 4:1  (slice2; segments: 4)  (cost=51493.60..64030.00 rows=250728 width=6)
   ->  HashAggregate  (cost=51493.60..64030.00 rows=250728 width=6)
         Group By: t1.name
         ->  Redistribute Motion 4:4  (slice1; segments: 4)  (cost=16391.68..36449.92
rows=250728 width=6)
               Hash Key: t1.name
               ->  HashAggregate  (cost=16391.68..16391.68 rows=250728 width=6)
                     Group By: t1.name
                     ->  Seq Scan on t1  (cost=0.00..11377.12 rows=250728 width=6)
```

如果在单机上的执行计划:

```
osdba=# explain select name,count(*) from t1 group by name;
                          QUERY PLAN
---------------------------------------------------------------
 HashAggregate  (cost=24346.00..24346.01 rows=1 width=36)
   ->  Seq Scan on t1  (cost=0.00..19346.00 rows=1000000 width=36)
```

GP的执行计划

```
tangdb=# explain select t1.id,t1.name,t2.name from t1,t2 where t1.id=t2.id;
                                  QUERY PLAN
-------------------------------------------------------------------------------------
 Gather Motion 4:1  (slice1; segments: 4)  (cost=23913.52..50334.32 rows=250728 width=16)
   ->  Hash Join  (cost=23913.52..50334.32 rows=250728 width=16)
         Hash Cond: t1.id = t2.id
         ->  Seq Scan on t1  (cost=0.00..11377.12 rows=250728 width=10)
         ->  Hash  (cost=11377.12..11377.12 rows=250728 width=10)
               ->  Seq Scan on t2  (cost=0.00..11377.12 rows=250728 width=10)
(6 rows)
```

GP的执行计划

```
tangdb=# explain select t1.id,t1.name,t2.name from t1,t2 where t1.id=t2.cn;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------
 Gather Motion 4:1  (slice2; segments: 4)  (cost=23913.52..70392.56 rows=250728 width=16)
   ->  Hash Join  (cost=23913.52..70392.56 rows=250728 width=16)
         Hash Cond: t2.cn = t1.id
         ->  Redistribute Motion 4:4  (slice1; segments: 4)  (cost=0.00..31435.36 rows=250728
width=10)
               Hash Key: t2.cn
               ->  Seq Scan on t2  (cost=0.00..11377.12 rows=250728 width=10)
         ->  Hash  (cost=11377.12..11377.12 rows=250728 width=10)
               ->  Seq Scan on t1  (cost=0.00..11377.12 rows=250728 width=10)
```

单机上执行的:

```
osdba=# explain select t1.id,t1.name,t2.name from t1,t2 where t1.id=t2.cn;
                            QUERY PLAN
-------------------------------------------------------------------------
 Hash Join  (cost=32789.00..90144.00 rows=1000000 width=46)
   Hash Cond: (t1.id = t2.cn)
   ->  Seq Scan on t1  (cost=0.00..19346.00 rows=1000000 width=40)
   ->  Hash  (cost=15406.00..15406.00 rows=1000000 width=10)
         ->  Seq Scan on t2  (cost=0.00..15406.00 rows=1000000 width=10)
```

GP的执行计划

```
tangdb=# explain select t1.id,t1.name,t2.name from t1,t2 where t1.cn=t2.cn;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Gather Motion 4:1  (slice3; segments: 4)  (cost=43971.76..90450.80 rows=250728 width=16)
   ->  Hash Join  (cost=43971.76..90450.80 rows=250728 width=16)
         Hash Cond: t1.cn = t2.cn
         ->  Redistribute Motion 4:4  (slice1; segments: 4)  (cost=0.00..31435.36 rows=250728
width=14)
               Hash Key: t1.cn
               ->  Seq Scan on t1  (cost=0.00..11377.12 rows=250728 width=14)
         ->  Hash  (cost=31435.36..31435.36 rows=250728 width=10)
               ->  Redistribute Motion 4:4  (slice2; segments: 4)  (cost=0.00..31435.36
rows=250728 width=10)
                     Hash Key: t2.cn
                     ->  Seq Scan on t2  (cost=0.00..11377.12 rows=250728 width=10)
```

GP的执行计划

```
tangdb=# explain select t1.id,t1.name,t3.name from t1,t3 where t1.cn=t3.cn;
                                    QUERY PLAN
-------------------------------------------------------------------------------------------------
 Gather Motion 4:1  (slice2; segments: 4)  (cost=15.00..13901.40 rows=25 width=12)
   ->  Hash Join  (cost=15.00..13901.40 rows=25 width=12)
         Hash Cond: t1.cn = t3.cn
         ->  Seq Scan on t1  (cost=0.00..11377.12 rows=250728 width=14)
         ->  Hash  (cost=10.00..10.00 rows=100 width=6)
               ->  Broadcast Motion 4:4  (slice1; segments: 4)  (cost=0.00..10.00 rows=100
width=6)
                     ->  Seq Scan on t3  (cost=0.00..5.00 rows=25 width=6)
```

Q&A