Greenplum 内核揭秘之执行引擎

目录

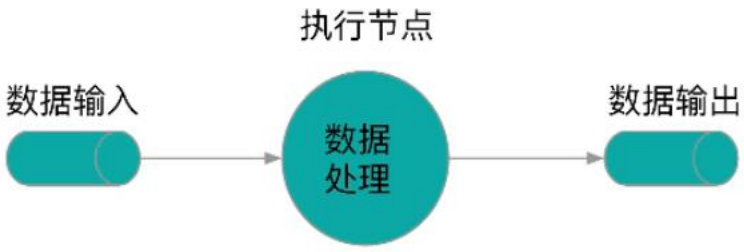
1 执行器介绍

1.1 什么是执行器

执行器是处理一个由执行计划节点组成的树，并返回查询结果

1.2 PlanNode(执行计划节点)

本质上就是数据处理



1.3 PlanTree(执行计划树)

如以下所示的 Query 查询树

```
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)
```

Seq Scan：原发性的扫描节点
Hash/Hash Join：非原发性扫描节点

1.3 执行模型

1.3.1 迭代模型(Pipeline 模型，Pull 方式)

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

1.3.2 向量化模型(VECTORIZATION Model)

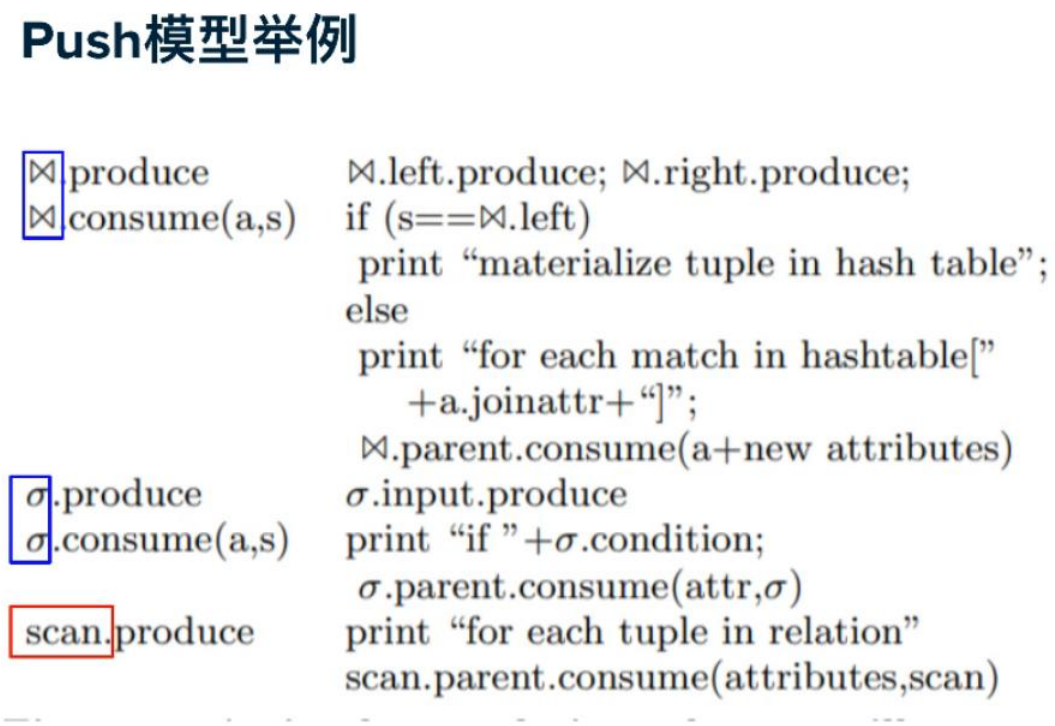
和迭代模型一样，每一个执行节点实现一个 next 函数，区别在于每一次迭代，执行节点返回一组 tuple，而非一个 tuple

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 对列存友好(可以利用压缩特性等)

1.3.3 PUSH 执行模型

每一个执行节点定义两个函数
1、produce 函数: 对原发性扫描节点，该函数名副其实，产生数据，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，用于调用节点的 produce 函数，快速定位到数据源头。

3、consume 函数:被下层节点调用，接受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。



1.3.4 PUSH 模型的优势

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

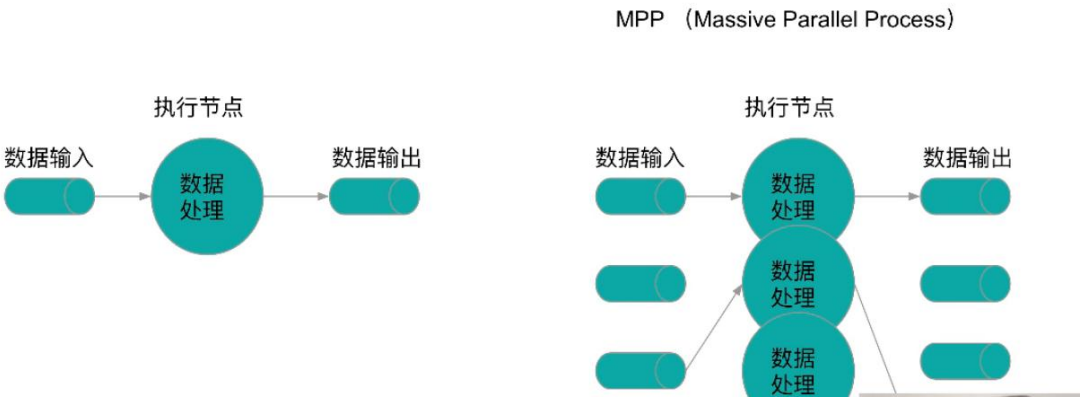
1.3.5 GPDB 使用的模型

GPDB 使用的是迭代模型，但是 GPDB 正在积极探索向量模型和 PUSH 模型

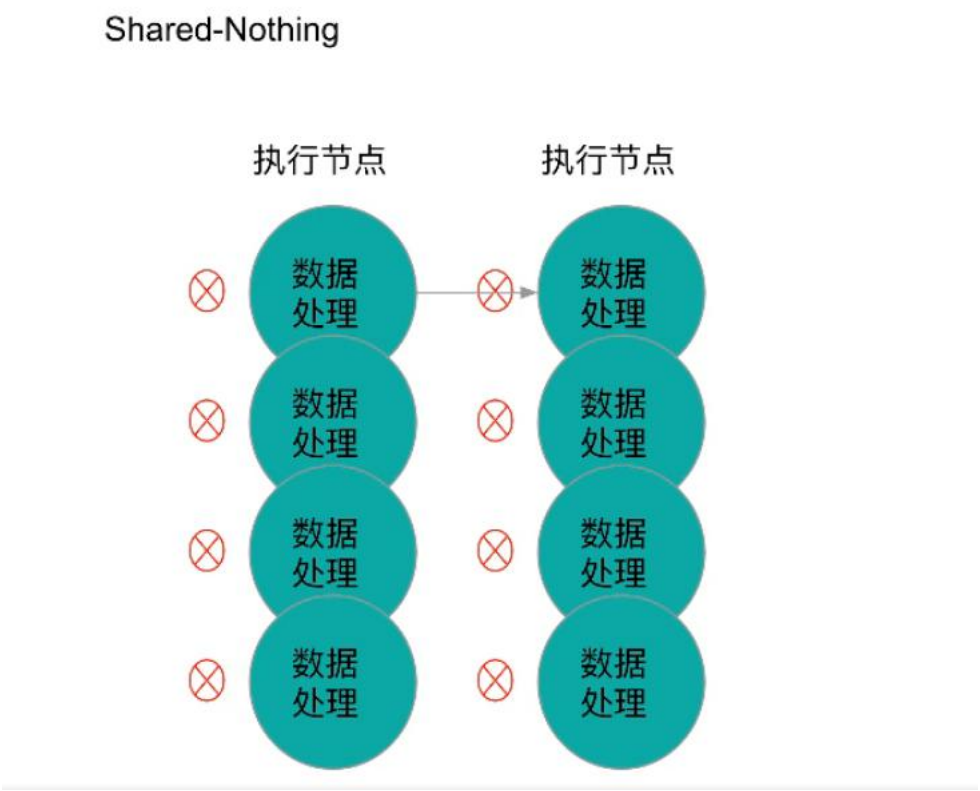
1.4 GPDB 执行器面临更大的挑战

1.4.1 MPP 架构的设计

Greenplum 使用的是 MPP(Massive Parallel Process)架构的设计



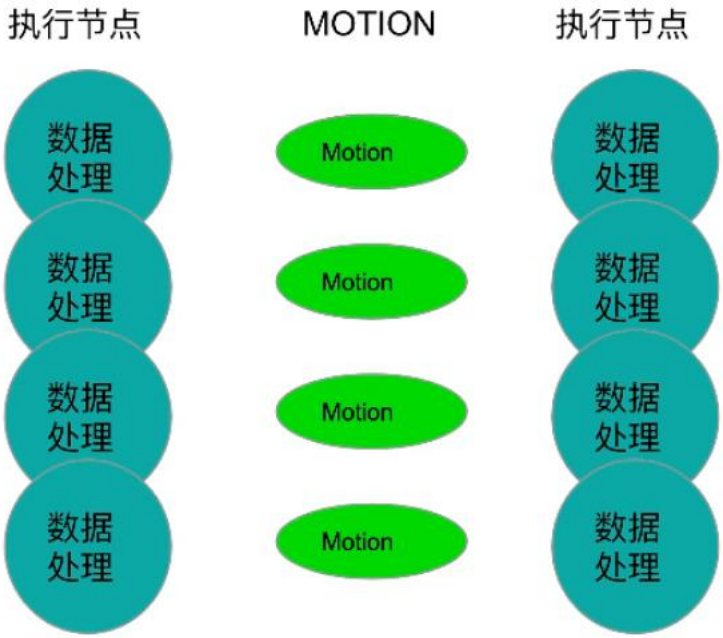
1.4.2 Shared-Nothing 进行数据传输



1.4.3 新的执行节点 MONTION

新的执行节点采用 MONTION，是一个并行化的执行节点

新的执行节点MOTION



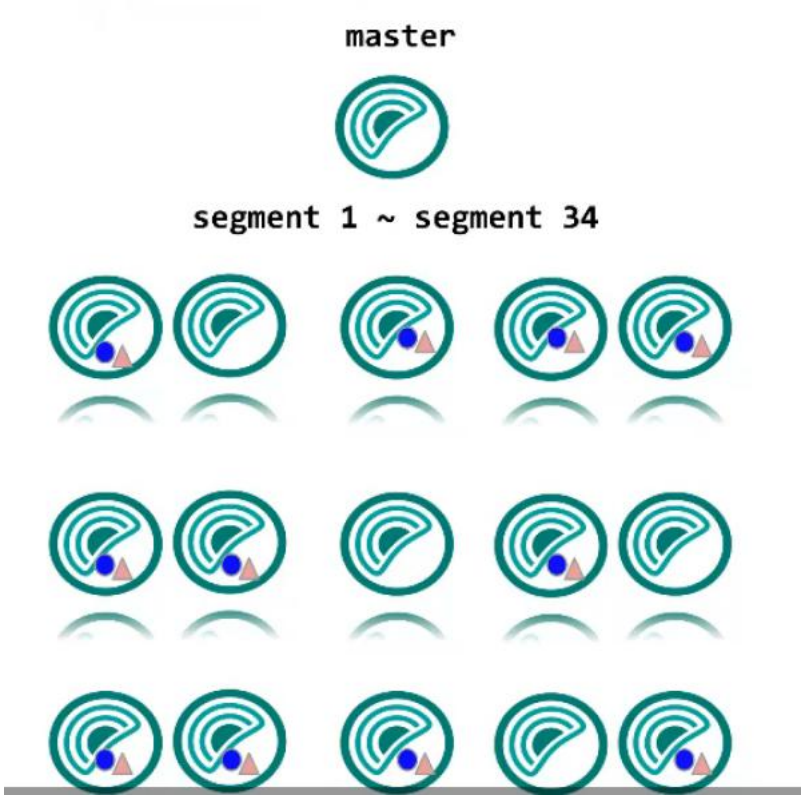
1.4.4 包含 Motion 的执行计划树

```
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)
```

2 并行化 PLAN

2.1 并行计划 PLAN 示意图



2.2 实例相亲案例

2.2.1 单身男女都居住在户籍所在地

策略
各省的部分单独自办相亲会，将本省的适龄单身男女组织到相亲场地相亲，并将结果返回总部。

2.2.2 单身女在户籍所在地,单身男在外地

策略 1
各省的部分单独办相亲会
1、首先每个省的单身男青年找出来，并将他们通过火车派送到原户籍坐在地
2、然后每个省接待这些男青年，并在本省找出女单身女青年，对他们进行相亲

策略二:
各省的部分自办相亲会
1、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
2、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，找出户籍一直

的相亲。

2.2.3 单身男女随机分布在全国各地

策略 1：
在总部举办相亲会，各省把单身男女通过火车派送到总部，总部接待并安排相亲，这个成本会很大。

策略 2:
各分部举办相亲会：
1、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。
2、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。
3、最后各省接待全国归来的男女，进行相亲。

2.2.4 相亲策划思路

1、人多力量大的原则，尽量利用各省的部分
2、首先分析当前男女的区域分布
3、必要时使用交通工具来打破地域的限制

2.3 GPDB 采用的分布情况

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

2.3.1 Locus 信息

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(随机表等)

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的信息

2.3.2 通过 Motion 来打破物理上的隔离

1、Redistribute Motion
2、Gather / Gather Merge Motion
3、Broadcast Motion
4、Explict Redistribute Motion

2.3.3 GPDB 优化器在对 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 优化时,参考 cdbllize_decorate_subplans_with_motions 函数
   d. .....

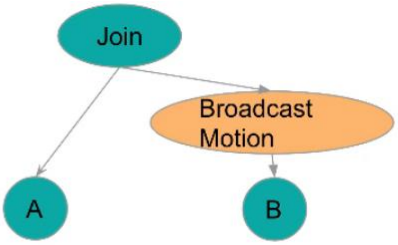
2、对已有数据集合进行 INSERT/UPDATE/DELETE 操作时
   参考 create_modifytable_path - > adjust_modifytable_subpaths

2.3.4 分布式 Join

select * from A inner join B;

A：键值表　CdbLocusType_Hashed

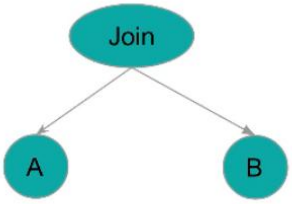
B：键值表　CdbLocusType_Hashed



select * from A inner join B;

A：键值表　CdbLocusType_Hashed

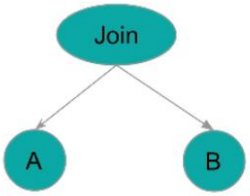
B：复制表　CdbLocusType_SegmentGeneral



select * from A inner join B;

A： 键值表　CdbLocusType_Hashed

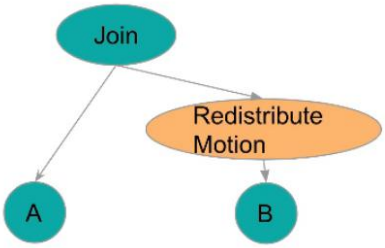
B： generate_series(1, 10)　CdbLocusType_General



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

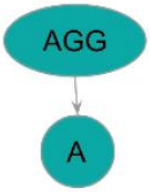
A： 键值表　CdbLocusType_Hashed on c1

B： 键值表　CdbLocusType_Hashed on c1

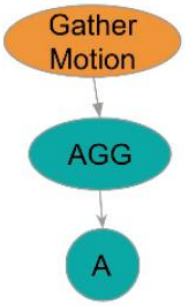


select count(*) from A group by c1;

A： 键值表　CdbLocusType_Hashed on c1



Agg(A)：CdbLocusType_Hashed



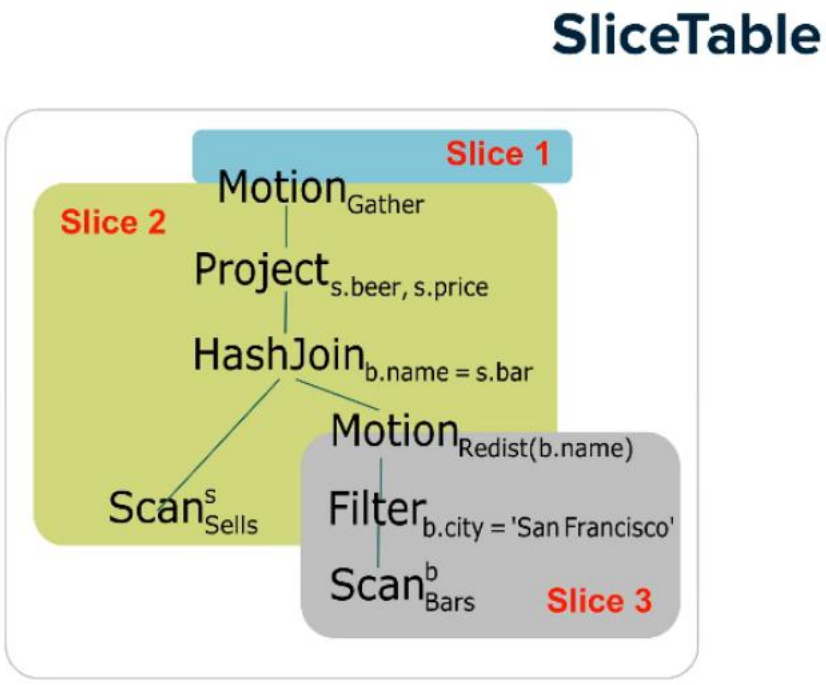
3 Dispatcher

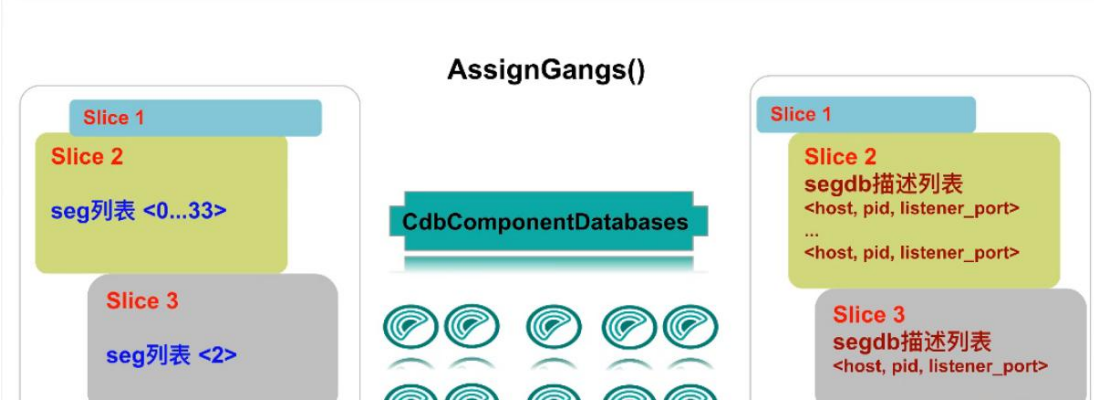
3.1 GPDB 执行器面对的问题

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

3.2 dispatcher 分配 QE 资源





AllocatwGang()
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使用(或者闲置一段时间后被清楚)

3.3. dispatcher 功能分发任务

CdbDispatchPlan Plan + SliceTable
CdbDispatchCommand
CdbDispatchDtxProtocolCommand
CdbDispatchUtilityStatement

3.4 dispatcher 功能协调控制

cdbdisp_checkDispatchResult(等待模式)
等待模式
1、blocking 阻塞等待所有 QEs 完成执行或者出现异常
2、Non -blocking 检查所有 QEs 的状态，若 QEs 有异常则报错，否则立即返回
3、Finish 给所有活动的 QEs 发送 QueryFinish 消息提前结束任务，QE 不报错
4、Cancel 给所有活动的 QEs 发送 QueryCancel 消息，终止任务。

3.5 典型的 dispatcher 程序

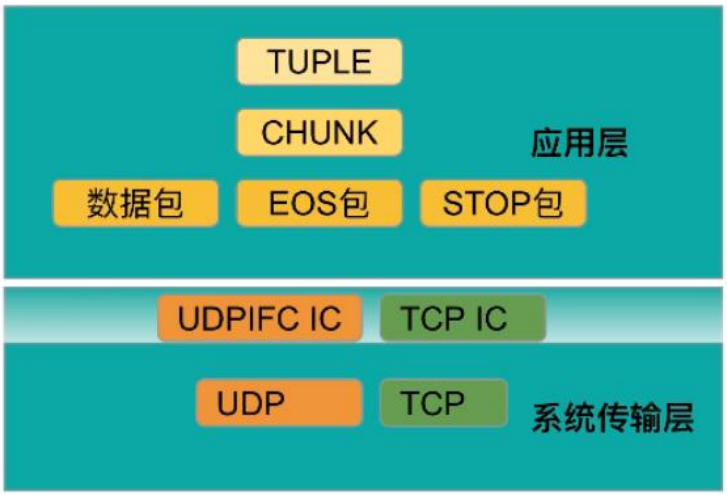
```
ds = cdbdisp_makeDispatcherState
primaryGang = AllocateGang(ds,GANGTYPE_PRIMARY_WRITER,segment)
cdbdisp_dispatchToGang(ds,primaryGang,-1)
cdbdisp_waitDispatchFinish(ds)
cdbdisp_checkDispatchResult(ds,DISPATCH_WAIT_NODE)
cdbdisp_getDispatchResults(ds,&qeError)
cdbdisp_destroyDispatcherstate(ds)
```

4 Interconnect

4.1 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

4.2 Interconnect Layout 实现



UDPIFC 定义

1、GPDB 自己实现的一种 RUDP(Reliable User Datagram Protocol)协议
2、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。
3、GPDB 当初引入 UDPIFC 主要为了解决复杂 OLAP 查询在大集群中使用链接资源过多的问题。

4.3 UDPIFC 线程模型

为什么使用线程模型?
1、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
2、QE 在 fork 的时候呼启动一个 udpifc 线程，该线程服务该 session 所有将要可能的查询
3、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。
4、设计的函数
线程细节可参考 rxThreadFunc 函数
接受端逻辑可参考 RecvTupleChunkFrom*函数
发送端逻辑可参考 SendChunkUDPIFC

4.4 可能有新的 Interconnect 类型

QUIC 协议
Proxy 协议