# Hacking Greenplum

## 工程师视角

吕正华

2021.05

# Table of Contents



GREENPLUM DATABASE®

# 自我介绍

- VMware 软件工程师，开发 Greenplum 内核
- 个人主页：https://kainwen.com
- 2014 年毕业于清华电子系智能感知实验室，工学硕士
- 曾经在豆瓣担任算法工程师

GREENPLUM DATABASE®

# Table of Contents



GREENPLUM DATABASE®

# Greenplum as MPP database

- Massively: 数据量太大了，必须要很多机器存
- Parallel: 多核多机器，计算上也可以利用并行
- Postgres
- 开源 Apache 许可证
- 部署灵活

# What is Greenplum

## Greenplum is Massively Parallel Postgres for Analytics

**Power at Scale**

Petabyte-scale Data Volumes

Unique cost-based optimizer for large-scale data

**True Flexibility**

Deploy Anywhere

Provide you more control over the software you deploy

**From BI to AI**

All In One Environment

A single, scale-out environment for converging analytic and operational workloads with AI supported.

**Open Source**

Avoid Propietary Vendor Lock-in

Apache license and based on PostgreSQL

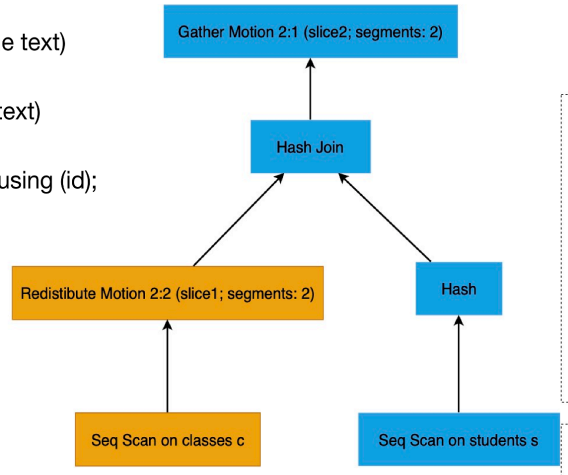
Figure: What is Greenplum

GREENPLUM DATABASE®

# 分布式优化器和分布式执行器

## Distributed plan and distributed excutor

```
create table student (id int, name text)
distributed by (id);
create table class (id int, name text)
distributed randomly;
select * from student join class using (id);
```



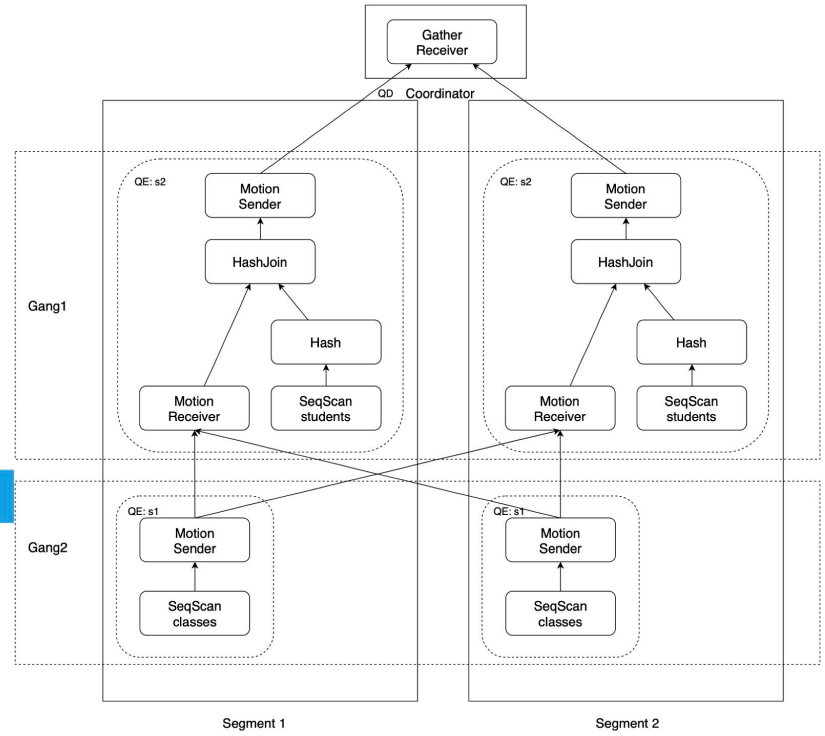




Figure: Greenplum 的分布式优化器和执行器

GREENPLUM DATABASE®

# 社区一瞥

**Fix plan for select-statement with locking clause** ···

Commit 6ebce73 do some optimization for select statement
with locking clause for some simple cases. It also applies
such optimization to select statement with limit.

In Greenplum, the results of limit can only be known on QD,
but we can only lock tuples on QEs. So this commit fixes this.

For replicated table, select statement may only execute
on one segment, but update statement has to happen on all segments.
We should also turn off such optimization on locking clause for
replicated table.

Also, Currently, Greenplum uses two-stage sort to implement
order by, and might generate a gather motion to QD.

If the processing order is: first order-by then rowmarks,
we might put a lockrows plannode above a gather motion.
However, we cannot lock tuples on QD.

This commit changes the order. The plan for the query
`select * from t order by c for update` on a hash-distributed
table or randomly-distributed table t is:

```
previous:
                QUERY PLAN
------------------------------------------
LockRows
    ->Gather Motion 3:1  (slice1; segments: 3)
      Merge Key: c
      ->  Sort
          Sort Key: c
           ->  Seq Scan on t

after this commit:
              QUERY PLAN
------------------------------------------
Gather Motion 3:1  (slice1; segments: 3)
    Merge Key: c
    ->  Sort
        Sort Key: c
         ->  LockRows
             ->  Seq Scan on t
```

kainwen committed on Jun 5, 2019 ✓

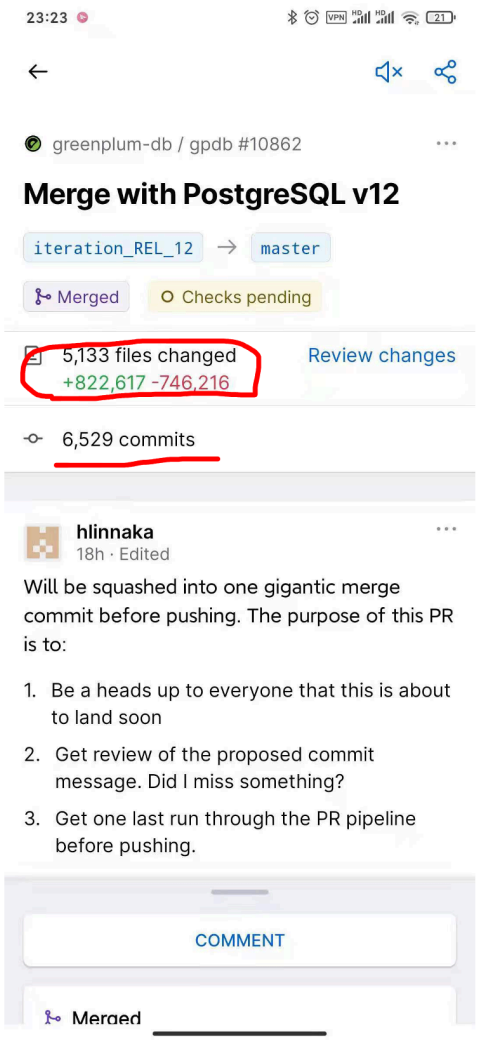



Figure: Greenplum Commit

GREENPLUM DATABASE®

# Table of Contents



# 锁模式调度问题

- Greenplum 6 进入 Global Deadlock Detector, OLTP 性能飙升 (SIGMOD 2021: Greenplum: A Hybrid Database for Transactional and Analytical Workloads)
- Greenplum 是少数 MPP 数据库支持更新分布键的，采用 Split-Update 技术
- Greenplum 的 MVCC, Read Committed 隔离级别下，同时 update 同一个 tuple 会引起等待，恢复后的事务需要用 EvalPlanQual 机制重新判断 tuple 是否有效
- DML 之间完全正确的调度比预想的难

GREENPLUM DATABASE®

# 不同 DML 的例子

create table t1(a int, b int) distributed by (a);

create table t2(a int, b int) distributed randomly;

```
gpadmin=# explain (costs off) update t1 set a = a + 1;
                          QUERY PLAN
-------------------------------------------------------------
 Update on t1
   ->  Explicit Redistribute Motion 3:3  (slice1; segments: 3)
         ->  Split
               ->  Seq Scan on t1
 Optimizer: Postgres query optimizer
```

```
gpadmin=# explain (costs off) update t1 set b = t1.a + 1
from t2 where t2.a > t1.a;
                        QUERY PLAN
-----------------------------------------------------------
 Update on t1
   ->  Nested Loop
         Join Filter: (t2.a > t1.a)
         ->  Broadcast Motion 3:3  (slice1; segments: 3)
               ->  Seq Scan on t2
         ->  Materialize
               ->  Seq Scan on t1
 Optimizer: Postgres query optimizer
(8 rows)
```

Figure: DML plan

GREENPLUM DATABASE®

# 使用 Prolog 计算最佳锁模式

```
conflict(A, split_update) :-
    hold_xid_lock(A).

conflict(A, B) :-
    mark_tuple_heap_updated(A),
    has_motion(B).
```

```
virtualdml_conflict(X1, X2, X3, X4, X5) :-
    member(X1, [1,2,3,4,5,6,7,8]),
    member(X2, [1,2,3,4,5,6,7,8]),
    member(X3, [1,2,3,4,5,6,7,8]),
    member(X4, [1,2,3,4,5,6,7,8]),
    member(X5, [1,2,3,4,5,6,7,8]),
    conflict(X1, X3),
    conflict(X1, X4),
    conflict(X1, X5),

    conflict(X2, X3),

    conflict(X3, X1),
    conflict(X3, X2),
    conflict(X3, X3),
    conflict(X3, X4),
    conflict(X3, X5),

    conflict(X4, X1),
    conflict(X4, X3),
    conflict(X4, X4),
    conflict(X4, X5),

    conflict(X5, X1),
    conflict(X5, X3),
    conflict(X5, X4).
```

Figure: Prolog 程序

GREENPLUM DATABASE®

# 决定数据存储位置中的哈希算法

- Hash-Distributed 表中数据具体应该送到哪个 Segment 要经历下面两步:
  - 根据数据本身和类型，选择 Hash 函数，这一步是把原始数据，映射到一个二进制串 (uint64): $tuple \rightarrow uint64$
  - 根据集群规模和上一步的二进制串，利用某种哈希算法，计算出数据目的地:$uint64 \rightarrow int$
- 其中第二步的可以选择一致性哈希算法，在分布式环境中集群扩容会减少数据移动开销
- 一致性哈希算法需要保证均匀性和单调性

GREENPLUM DATABASE®

# 哈希过程示意图

```
cdbhashinit(h);

i = 0;
foreach(hk, hashkeys)
{
    ExprState  *keyexpr = (ExprState *) lfirst(hk);
    Datum       keyval;
    bool        isNull;

    /*
     * Get the attribute value of the tuple
     */
    keyval = ExecEvalExpr(keyexpr, econtext, &isNull);

    /*
     * Compute the hash function
     */
    cdbhash(h, i + 1, keyval, isNull);
    i++;
}
target_seg = cdbhashreduce(h);
```

Figure: Greenplum evalHashKey 函数片段

GREENPLUM DATABASE®

# Jump consistent Hashing

- Lamping, John, and Eric Veach. "A fast, minimal memory, consistent hash algorithm." arXiv preprint arXiv:1406.2294 (2014).
- Greenplum 6 配合在线扩容引入了 Jump Consistent Hashing
- 巧妙的思路:
  - 利用数据本身初始化伪随机数生成器的种子，保证了确定性
  - 可以用概率手段进行算法分析
  - Jump 的思路常见于采样算法，可以优化得到对数时间复杂度

# 基于查找表的一致性哈希算法: Maglev

- 来自 Heikki 老师的邮件: https://groups.google.com/a/greenplum.org/g/gpdb-dev/c/TW0CEsylrNQ/m/VKcstawbAwAJ
- Google 的磁悬浮列车算法: Eisenbud, Daniel E., et al. "Maglev: A fast and reliable software network load balancer." 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16). 2016.
- Maglev 非常复杂，且不保证单调性，且没有均匀性分析，怎么看都是一个糟糕的算法

# 可能是最好的基于查找表的哈希算法: Pivotal Hash

- https://kainwen.com/2019/02/13/pivotal-hash-a-new-consistent-hash-algorithm/
- 来自 Knuth 书籍里的数学公式: $n = \sum_{k=0}^{m-1} \lfloor \frac{n-k}{m} \rfloor$
- $\lfloor \frac{n-k}{m} \rfloor$ 关于 $m$ 单调递减
- 均匀 $\lfloor \frac{n}{m} \rfloor - \lfloor \frac{n-m+1}{m} \rfloor \leq 1$
- 天下没有免费的午餐: Cache Miss

# Pivotal Hash Python 实现

```
def build_lookup(self):
    lookup = [0] * self.M
    for i in range(1, self.Nsegs):
        lookup = self._build_bookup(i, lookup)
    return lookup

def _build_bookup(self, i, lookup):
    """
    from i-1 enlarge to i
    """
    prev_config = self.compute_config(i-1)
    curr_config = self.compute_config(i)
    delta = [(n1-n2)
             for n1, n2 in zip(prev_config, curr_config)]

    cp_lookup = deepcopy(lookup)
    for index, seg in enumerate(lookup):
        d = delta[seg]
        if d == 0:
            continue
        cp_lookup[index] = i
        delta[seg] -= 1

    self.is_good(lookup, cp_lookup)

    return cp_lookup

def compute_config(self, i):
    ii = i + 1
    return [int(ceil(float(self.M - x) / ii))
            for x in range(ii)]
```

Figure: Pivotal Hash

GREENPLUM DATABASE®

# Greenplum 的多阶段聚合

- MPP 环境下一般都青睐多阶段聚合，第一阶段能够减少大量数据
- 聚合执行节点属于内存消耗型，必要时候需要启动外存
- Greenplum 多阶段有 Streaming 技术
- 如果第一阶段大量 spill 或者大量 streaming，则第一阶段几乎没有必要
- 需要对第一阶段 spill 或者 streaming 的数目严格量化分析

# 聚合查询计划

```
gpadmin=# explain (costs off) select b, sum(a), avg(c)
gpadmin-# from t group by b;
                        QUERY PLAN
-----------------------------------------------------------
 Gather Motion 3:1  (slice2; segments: 3)
   ->  HashAggregate
         Group Key: t.b
         ->  Redistribute Motion 3:3  (slice1; segments: 3)
               Hash Key: t.b
               ->  HashAggregate
                     Group Key: t.b
                     ->  Seq Scan on t
 Optimizer: Postgres query optimizer
```

```
gpadmin=# explain (costs off) select b, sum(a), avg(c)
gpadmin-# from t group by b;
                        QUERY PLAN
-----------------------------------------------------------
 Gather Motion 3:1  (slice2; segments: 3)
   ->  HashAggregate
         Group Key: b
         ->  Redistribute Motion 3:3  (slice1; segments: 3)
               Hash Key: b
               ->  Seq Scan on t
 Optimizer: Postgres query optimizer
(7 rows)
```

Figure: 多阶段和单阶段查询计划对比

# 第一阶段 spill 或 streaming 量化模型

- 一份数据流数目是 $N$, 基数是 $d$, 哈希表尺寸是 $h$, 每个分组的大小 $g = \frac{N}{d}$
- 哈希表满了后，就清空，或 spill 或 streaming
- Q: 估计 streaming 到下一轮或者 spill 的数据量
- A: $h * g / \ln \frac{d}{d-h}$, for $d > h$
- 完整版本推理: https://kainwen.com/2020/09/19/coupon-collectors-problem-sample-without-replacement/
- 简化版但是精度很高: $EGF(z) = (e^z - 1)^M$
- 这些技术还是来自 Knuth 的书籍
- 最终可以智能选择 1 阶段或者 2 阶段，某些 TPCDS 查询性能翻倍

GREENPLUM DATABASE®

# 其他解析组合问题

- If you can specify it, you can analyze it
- Hyperloglog (https://kainwen.com/2020/06/21/a-tour-of-understanding-hyperloglog/)
- Hyperbitbit
- https://www.cs.princeton.edu/ rs/talks/AC11-Cardinality.pdf

# Table of Contents



GREENPLUM DATABASE®

# 媒体资料

## 自媒体

社区网站、微信、微博...


## 技术社区平台

CSDN、开源中国、SegmentFault...


## 视频平台

B站、腾讯视频..

Figure: Greenplum 中文社区

GREENPLUM DATABASE®

# Table of Contents



GREENPLUM DATABASE®

# 致谢

Q & A

谢谢!

GREENPLUM DATABASE®

# 关注 Greenplum 中文社区


Figure: greenplum 中文社区

GREENPLUM
DATABASE®