Hacking Greenplum

interesting and challengeable problems

吕正华

Staff Engineer I @ VMware

2021.06

Table of Contents

自我介绍

- VMware 资深软件工程师，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
- 2011 年毕业于北京邮电大学信息与通信工程学院，工学学士
- 曾经在豆瓣担任算法工程师

GREENPLUM DATABASE®

Table of Contents

Greenplum as MPP database

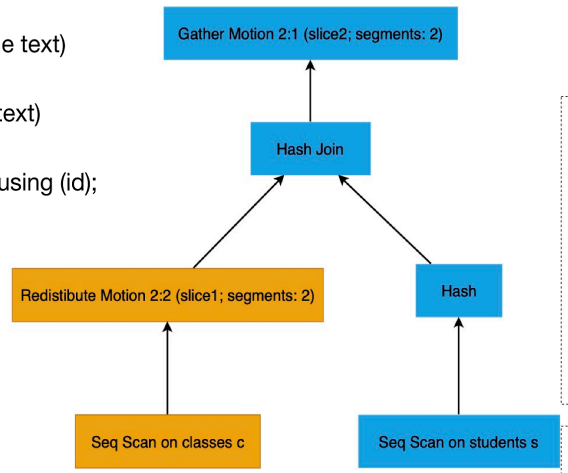
- Massively: 数据量太大了，必须要很多机器存
- Parallel: 多核多机器，计算上也可以利用并行
- Postgres
- 开源 Apache 许可证
- 部署灵活

分布式优化器和分布式执行器示例

Motion Node 是 Greenplum 构造数据搬移抽象的重要概念。

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);
```



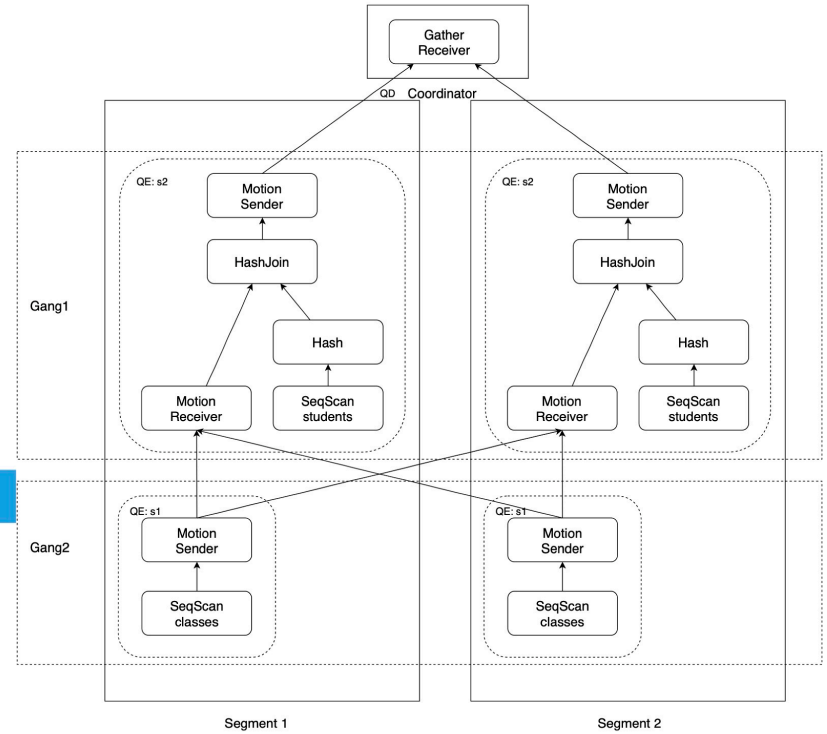


Figure: Greenplum 的分布式优化器和执行器

分布式优化器和分布式执行器的关键概念

- Slice: Motion node 把查询计划天然的切割开, 每一片查询计划称之为 slice
- Gang: 每一片查询计划会有一组分布式进程执行, 这组进程称之为 Gang
- Gang 与 Gang 之间通过 Interconnect 通信数据 (UDP or TCP)
- QD: coordinator fork 出来的 worker
- QEs: 所有的 Gang, 接受 QD 通过 dispatcher 下发的查询计划和控制命令
- Motion 是单向的数据通信
- Greenplum 的执行器模型和 Postgres 一致: One tuple a time; 但是 Motion node 有 buffer, buffer 满了后才发送出去

GREENPLUM DATABASE®

分布式查询优化器

- Greenplum 有两个分布式优化器, 各擅胜场:
  - Legacy planner: 基于 Postgres Planner 的分布式加强
  - ORCA[1]: Cascades 架构的杰出代表, 有很多特有的技术, 如 CTE optimization[2], Dynamic Partition Selector, Sublink Pullups, ...
- 今天着重讨论 Legacy planner 的核心技术

SQL 优化器——PL 视角

- 编译器是从字符串流到指令流的转换过程, 中间经历若干 IR, 每一层 IR 都有它的意义, 承上启下
- 参考 IUB 的编译原理课 [3]: Scheme $\rightarrow$ R3 $\rightarrow$ C2 $\rightarrow$ X86* $\rightarrow$ X86
- Postgres Planner: SQL string $\rightarrow$ Parse Tree $\rightarrow$ Query Tree $\rightarrow$ Path $\rightarrow$ Plan
- Query Tree 和 Path 是非常重要的中间表示:
  - RBO: 基于 Query Tree 可以做很多规则优化，如 Sublink Pullup, Outer Join Elimination, Predicate Pushdown
  - CBO: Path 产生于基于代价的优化部分

GREENPLUM DATABASE®

Postgres Planner Path

- Base Relation:
  - Index Scan Path
  - Sequence Scan path
  - Bitmap index Scan
- Join Relation: 基于动态规划搜索策略 [4]
  - Hash Join Path
  - Merge Join path
  - Nestloop Join Path
- Aggregation Relation:
  - GroupAgg path
  - HashAgg path
- Subquery Scan Path
- Path 里包含代价相关信息

GREENPLUM DATABASE®

Greenplum Legacy Planner 核心技术

- 既然 Motion Node 是分布式查询计划中最重要的概念，中间表示 IR 中必须要加入这方面的推理
- 这一部分是物理优化的一部分，因此需要在生成 Path 阶段考虑
- Q: 如何设计 Motion 的添加规则?
- A: Motion 改变数据的分布, Path 中必须添加额外信息，记录 Path 表述的数据分布
- Locus: Greenplum 给每个 Path 添加的表述数据分布的字段, 可以认为是一种分布式类型系统
- *Greenplum* 查询优化器关键技术介绍

GREENPLUM DATABASE®

Locus 和 Motion

```c
/*
 * CdbLocusType
 */
typedef enum CdbLocusType
{
    CdbLocusType_Null,
    CdbLocusType_Entry,         /* a single backend process on the entry db:
                                 * usually the qDisp itself, but could be a
                                 * qExec started by the entry postmaster.
                                 */
    CdbLocusType_SingleQE,      /* a single backend process on any db: the
                                 * qDisp itself, or a qExec started by a
                                 * segment postmaster or the entry postmaster.
                                 */
    CdbLocusType_General,       /* compatible with any locus (data is
                                 * self-contained in the query plan or
                                 * generally available in any qExec or qDisp) */
    CdbLocusType_SegmentGeneral,/* generally available in any qExec, but not
                 * available in qDisp */
    CdbLocusType_Replicated,    /* replicated over all qExecs of an N-gang */
    CdbLocusType_Hashed,        /* hash partitioned over all qExecs of N-gang */
    CdbLocusType_HashedOJ,      /* result of hash partitioned outer join, NULLs can be anywhere */
    CdbLocusType_Strewn,        /* partitioned on no known function */
    CdbLocusType_End            /* = last valid CdbLocusType + 1 */
} CdbLocusType;
```

```
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);
```

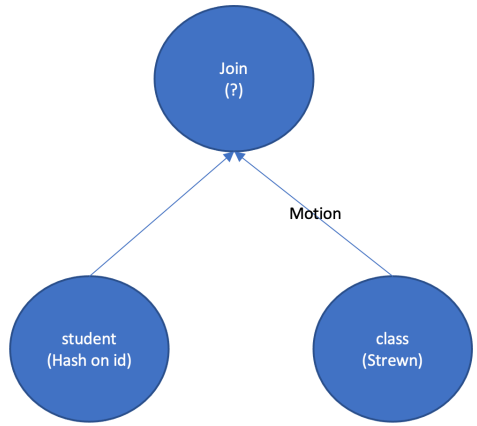


Figure: Locus 和 Motion 示意图

GREENPLUM DATABASE®

分布式事务模型

- 参考Greenplum 分布式数据库内核揭秘
- Postgres 和 Greenplum 都使用 MVCC 进行并发控制 [5], 且都是基于 Snapshot
- Greeplum 中每个 Segment(包含 Coordinator) 都会创建一个局部事务, 局部事务的提交是异步
- Greenplum 要保证各个 Segments 上的可见性是一致的, 因此引入了分布式事务:
  - 分布式事务 ID 和分布式 Snapshot 由 Coordinator 生成并下发
  - QEs 上维护一个局部事务 ID 和分布式事务 ID 的映射 (LRU cache)
- 写操作的事务的必须全局一致: 两阶段提交
- Slices 之间的一致性: Writer QE(负责生成局部事务 ID 和接受设置 Snapshot) 和 Reader QE

GREENPLUM DATABASE®

Greenplum 中的锁

- 锁是并发控制中另一个重要技术和概念
- Greenplum 中有三种锁:
  - Spin Lock: 保护共享对象, 不主动交出 CPU, 关键区域较小的时候可以避免过多的上下文切换
  - Lw Lock: 保护共享对象，互斥时候会 sleep 交出 CPU
  - Object Lock: 数据库对象锁，有 8 种模式，保护数据库对象，跟 SQL 类型紧密相关
- 两阶段锁: 第一阶段持锁，然后一直保持，在事务结束的时候释放[1]
- 当 Snapshot 遇上对象锁: 持锁发生等候，就总有唤醒的时刻，被唤醒就说明世界发生了改变，那么 Snapshot 是否需要变化?[2]

---

[1] Releasing Locks As Early As You Can: Reducing Contention of Hotspots by Violating Two-Phase Locking

[2] 参考 https://github.com/greenplum-db/gpdb/pull/11895

GREENPLUM DATABASE®

Greenplum 开源社区

- 只有一个 Greenplum repo，原厂工程师每天的工作也是在 Github 上工作
- Pull Request 有详细的 commit message 和注释，详细得如同一篇小教程
- Pull Request 都需要加测试，必须有其他人 approve 才能 merge
- 必要时需要向 Greenplum Dev Mailing List 发起讨论

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 ✓

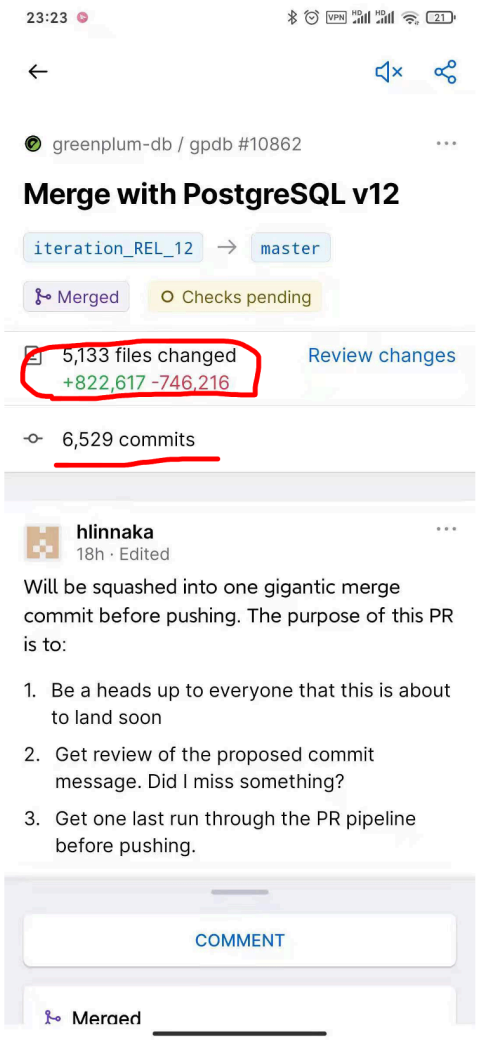


Figure: Greenplum Commit

GREENPLUM DATABASE®

Table of Contents

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

- Greenplum 6 配合在线扩容引入了 Jump Consistent Hashing[6]
- 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
- 均匀得到了保证
- 不需要维护一个" 沉重的状态"
- Analysis of Jump Consistent Hash Algorithm

GREENPLUM DATABASE®

基于查找表的一致性哈希算法: Maglev

- 来自 Heikki 老师的邮件:
  - Jump consistent hashing 引入额外 CPU 消耗
  - 可否用 $O(1)$ 代替 $O(lon(n))$
- Google 的磁悬浮列车算法 [7] 是基于查找表的
- 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

GREENPLUM DATABASE®

可能是最好的基于查找表的哈希算法: Pivotal Hash

- Pivotal Hash 算法分析
- 灵感: 原始数据均匀分散在各 Slot，再想办法解决单调性问题
- 《具体数学》[8] 的一个公式就浮现在脑海中:

$$n = \left\lceil \frac{n}{m} \right\rceil + \left\lceil \frac{n-1}{m} \right\rceil + \cdots + \left\lceil \frac{n-m+1}{m} \right\rceil . \tag{3.24}$$

Figure: 整数函数累加公式

Pivotal Hash 的均匀性和单调性

- 思路起点就是均匀性，形式化描述为 $\lceil \frac{n}{m} \rceil - \lceil \frac{n-m+1}{m} \rceil \leq 1$
- 解决单调性需要基于 Induction(归纳法), 核心是“只扩容一台机器的单调性”

$$n = \sum_{k=0}^{m-1} \lceil \frac{n-k}{m} \rceil = \sum_{k=0}^{m-1} s_k^{(m)}$$

$$= \sum_{k=0}^{m} \lceil \frac{n-k}{m+1} \rceil = \sum_{k=0}^{m-1} \lceil \frac{n-k}{(m+1)} \rceil + \lceil \frac{n-m}{m+1} \rceil = \sum_{k=0}^{m-1} s_k^{(m+1)} + t^{(m+1)}$$

- 注意到 $s_k^{(m)}$ 关于 m 单调递减
- $t^{(m+1)} = \sum_{k=0}^{m-1} \left(s_k^{(m)} - s_k^{(m+1)} \right)$
- 上面的公式可以解释单调性: m 台机器的系统扩容成 m+1 台, 旧的 m 台机器匀一些数据到新机器上，旧机器之前没有数据迁移

GREENPLUM DATABASE®

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®

免费的午餐?

- 工程中查找表的的尺寸一般要求远远大于集群规模 (100 倍以上)
- 假设是 100 台机器，最少也需要几十个 KBytes 的内存，会超过 Cache
- 对查找表的访问是随机访问，因此几乎永远 Cache Miss
- 扩容和数据重分布解藕的系统，需要维护多套查找表，系统复杂化
- 性能测试发现 Jump Consistent Hashing 工作得足够满意

GREENPLUM DATABASE®

解析组合数学的基本概念 [9]

- Class 是集合以及一个对元素的 size 函数: $|e| :: E \mapsto \mathbb{N}$
- $\mathcal{A}$ 是一个 Class, $a \in \mathcal{A}$ 是元素
- OGF: $A(z) = \sum_{a \in \mathcal{A}} z^{|a|} = \sum_{N \geq 0} A_N z^N$, $A_N = [z^n]A(z)$
- EGF: $A(z) = \sum_{a \in \mathcal{A}} \frac{z^{|a|}}{|a|!} = \sum_{N \geq 0} A_N \frac{z^N}{N!}$, $A_N = N![z^n]A(z)$
- 一个简单生成函数 (复函数) 包含了组合结构 $\mathcal{A}$ 的全部信息

GREENPLUM DATABASE®

解析组合数学一例

一个关于概率的数学问题?

有一个点在数轴上移动，每一次移动有0.5的概率向正方向移动一格，有0.5的概率向反方向移动一格。问这个点从原点开始移动m次，从来没有到过负半轴的概率?

（最后这个点不需要回到原点，不然就是Catalan数了）

Figure: 解析组合数学一例

- 所有满足条件的运动序列是一个组合结构 $\mathcal{A}, |s| = len(s)$
- $\mathcal{A} = \{\epsilon, +, ++, +-, +-+, ++-, +++, \dots\}$
- $\mathcal{B}$ 是所有上述序列中恰好回到原点的 Class
- $\mathcal{C}$ 是所有上述序列中最终落在正半轴的 Class

GREENPLUM DATABASE®

解析组合数学一例——推理解答

- $\mathcal{A} = \mathcal{B} \cup \mathcal{C} \implies A(z) = B(z) + C(z)$
- 已知 $B(z)$ 是 Catalan 多项式, 还需要一个方程消除 $C(z)$, 需要从组合结构入手
- 递归的构造 $\mathcal{A}$ 中序列:
  - 空序列 ϵ
  - $\mathcal{B}$ 中序列末尾拼接 (笛卡尔积) 一个 $+$
  - $\mathcal{C}$ 中序列末尾拼接 (笛卡尔积) 一个 $+$ 或者 $-$
- $A(z) = 1 + zB(z) + 2zC(z)$
- $A(z) = \frac{\sqrt{1-4z^2}+2z-1}{2z(1-2z)} = \frac{1}{\sqrt{1-4z^2}} + \frac{1-\sqrt{1-4z^2}}{2z\sqrt{1-4z^2}}$
- $[z^n]A(z) = \binom{N}{\lceil N/2 \rceil}$
- 广义斯特林公式: $x! = \Gamma(x+1) \sim \sqrt{2\pi x}(\frac{x}{e})^x$
- 概率为: $\frac{A_m}{2^m} \sim \frac{2}{\sqrt{2\pi m}}$

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

GREENPLUM DATABASE®

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
- 近似模型 1: 有放回的 Coupon collector 问题 [10]
- 近似模型 2: 无放回的 Coupon collector 问题 [11]
- 本质是在求解某个事件的平均等候时间
- $Exp(waiting\_time) = \sum_{m \geq 0} P(m \text{ times no happen})$
- 优化代价模型后，某些 TPC-DS 查询智能选择 1 阶段还是 2 阶段，性能翻倍

GREENPLUM DATABASE®

符号方法 [12]

- Coupon collector 序列: M 个非空 slot (类比容量是 M 的满 hash table)
- $R_M = SEQ_M(SET_{>0}(Z))$, 然后一步写出生成函数
- $R_M(z) = (e^z - 1)^M \implies R_{MN} = \sum_j \binom{M}{j} (-1)^j (M-j)^N$
- N 次搞不定的概率是 $P_{MN} = 1 - \frac{R_{MN}}{M^N}$
- 平均等候长度是 $\sum_{N \geq 0} P_{MN} = MH_M$
- $H_M \sim ln(M)$ 是 Harmonic Series
- 上面最后一步的化简用到 Knuth[8] 的 (公式 6.72)

GREENPLUM DATABASE®

解析组合数学在数据库中其他应用

- Hyperloglog 算法
- Hyperbitbit 算法

递归 CTE

- 递归 CTE 是 SQL99 引入的, 受到逻辑编程语言的影响 (Datalog, Prolog)
- 递归 CTE 和窗口函数的引入使得 SQL 本身是图灵完备的语言
- 近年持续有顶会论文在研究这个话题:
  - One WITH RECURSIVE is Worth Many GOTOs (SIGMOD 2021)
  - On the Optimization of Recursive Relational Queries: Application to Graph Queries (SIGMOD 2020)
  - RaSQL: Greater Power and Performance for Big Data Analytics with Recursive-aggregate-SQL on Spark (SIGMOD 2019) [3]
  - ...

[3]这个论文的优化其实 Postgres 早就做了:参考https://kainwen.com/2020/03/21/rasql-in-postgres/

GREENPLUM DATABASE®

递归 CTE 在 Postgres 中的查询计划和执行语义

```
zlyu=# explain (costs off)
 WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT * FROM t;
              QUERY PLAN
---------------------------------------
 CTE Scan on t
   CTE t
     ->  Recursive Union
           ->  Result
           ->  WorkTable Scan on t t_1
                 Filter: (n < 100)
(6 rows)
```

```
/* ----------------------------------------------------------------
 *      ExecRecursiveUnion(node)
 *
 *      Scans the recursive query sequentially and returns the next
 *      qualifying tuple.
 *
 * 1. evaluate non recursive term and assign the result to RT
 *
 * 2. execute recursive terms
 *
 * 2.1 WT := RT
 * 2.2 while WT is not empty repeat 2.3 to 2.6. if WT is empty returns RT
 * 2.3 replace the name of recursive term with WT
 * 2.4 evaluate the recursive term and store into WT
 * 2.5 append WT to RT
 * 2.6 go back to 2.2
 * ----------------------------------------------------------------
 */
static TupleTableSlot *
ExecRecursiveUnion(PlanState *pstate)
```

Figure: 递归 CTE plan 和执行器逻辑

脑洞：是否可以把 PLSQL 等高级语言直接编译到查询计划树?

GREENPLUM DATABASE®

用 SQL 编写 Dijkstra 算法

- 非递归部分记录最原始状态的信息
- 递归部分用 worktable 和地图表进行 join 等 SQL 操作，更新知识
- 目前 Postgres 里对递归 CTE 还是有诸多限制 (Greenplum 限制得更多):
  - 不能有 order by
  - worktable 不能在 subquery 里
  - worktable 不能 scan 两次
  - worktable 不能用在集合的 except 里
  - ...
- 详细代码参考: Dijkstra via SQL: a glance at Recursive CTE

GREENPLUM DATABASE®

Greenplum 事务调度的锁模式设计

- 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

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

完整的程序参考: lockmode.pl

GREENPLUM DATABASE®

Table of Contents

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®

关注 Greenplum 中文社区

Figure: greenplum 中文社区

GREENPLUM
DATABASE®

Table of Contents

References I


M. A. Soliman, L. Antova, V. Raghavan, A. El-Helw, Z. Gu, E. Shen, G. C. Caragea, C. Garcia-Alvarado, F. Rahman, M. Petropoulos *et al.*, "Orca: a modular query optimizer architecture for big data," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 2014, pp. 337–348.


A. El-Helw, V. Raghavan, M. A. Soliman, G. Caragea, Z. Gu, and M. Petropoulos, "Optimization of common table expressions in mpp database systems," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1704–1715, 2015.


J. G. Siek, C. Factora, A. Kuhlenschmidt, R. R. Newton, R. Scott, C. Swords, M. M. Vitousek, M. Vollmer, and A. Tolmach, "Essentials of compilation," 2021.


D. Gustafsson, "Through the joining glass," PostgreSQL Conference Europe, Tech. Rep., 2017. [Online]. Available: https://www.postgresql.eu/events/pgconfeu2017/sessions/session/1586/slides/26/Through_the_Joining_Glass-PGConfeu-DanielGustafsson.pdf


B. MOMJIAN, "Mvcc unmasked." [Online]. Available: https://momjian.us/main/writings/pgsql/mvcc.pdf


J. Lamping and E. Veach, "A fast, minimal memory, consistent hash algorithm," *arXiv preprint arXiv:1406.2294*, 2014.


D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, B. Cheyney, W. Shang, and J. D. Hosein, "Maglev: A fast and reliable software network load balancer," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 523–535.


R. L. Graham, O. Patashnik, and D. E. Knuth, "Concrete mathematics: a foundation for computer science," 1994.


P. Flajolet and R. Sedgewick, *Analytic combinatorics*. cambridge University press, 2009.

GREENPLUM DATABASE®

References II


Z. Lyu, "Coupon collector' s problem: an infinite-series perspective." [Online]. Available: https://kainwen.com/2019/08/20/coupon-collectors-problem-a-infinite-series-perspective/


——, "Coupon collector' s problem: sample without replacement." [Online]. Available: https://kainwen.com/2020/09/19/coupon-collectors-problem-sample-without-replacement/


R. Sedgewick and P. Flajolet, *An introduction to the analysis of algorithms*. Pearson Education India, 2013.

GREENPLUM DATABASE®

Table of Contents

致谢

Q & A

谢谢!

GREENPLUM DATABASE®