POSTGRESCONF CN 2020 | PGConf.Asia 11.17-11.20

Greenplum的锁

吕正华 VMware高级软件工程师

https://cn.greenplum.org
https://2020.postgresconf.cn

内容提要

- Greenplum架构简介
- Greenplum中的锁
- 死锁问题和全局死锁检测
- 降锁后的各类问题

PART 01

Greenplum架构简介

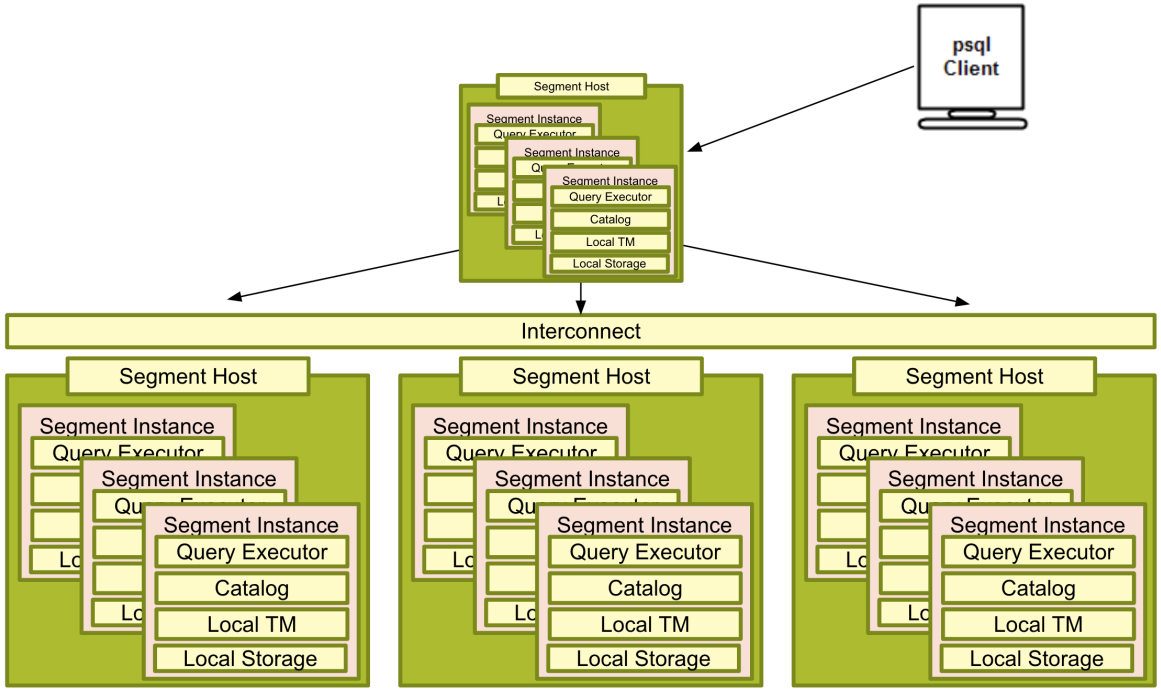
为什么要MPP架构?

- 单机无法存下海量数据 ➔ 必须要很多机器分布式存储数据 M(Massively)
- 单机无法利用分布式计算的优势 ➔ 必须要在多机都起进程协同并发处理数据 P(Parallel)
- 单机无法实现高可用
- 需要MPP架构高效的解决:
  - 分布式存储
  - 分布式查询计划
  - 分布式执行引擎
  - 分布式事务管理
  - 高可用

Greenplum MPP架构

- 一个Greenplum集群由很多运行着的Postgres实例组成
- 一个Master
  - 管理集群元信息
  - 直接和客户端通信
  - 生成分布式查询计划
  - 控制管理分布式事务
  - 不存储用户数据
- 众多Segment
  - 存储用户数据
  - 执行查询计划
  - 通过Interconnect进行通信数据

Greenplum MPP架构

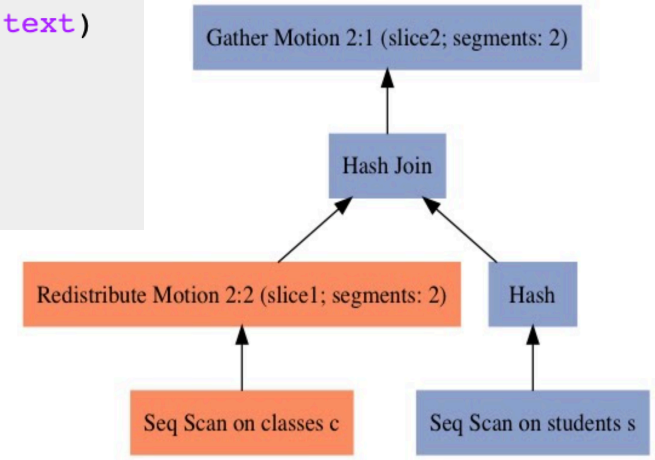


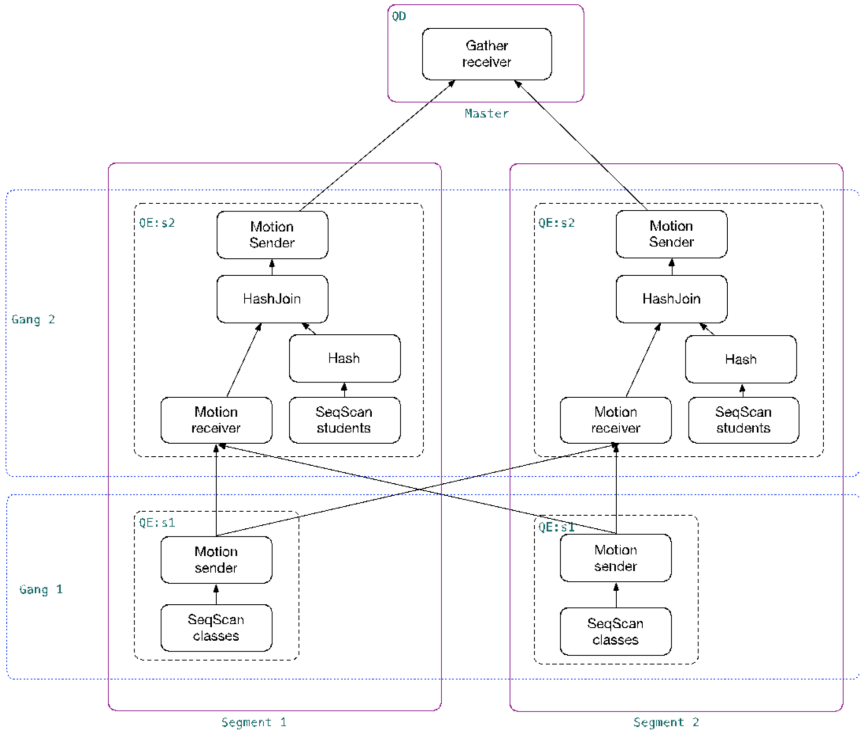
分布式查询计划和分布式执行器

- 数据必须分片的存储在各个segment机器，每个机器局部只有一小部分数据
- 简单的把单机查询计划发给每个segment再局部执行后收集结果返回并不工作，因为针对join或者group by的查询，可能发生跨机器数据匹配
- 杜绝跨机器匹配的影响➔分布式计算必须涉及数据通信
- Greenplum通过引入Motion计划节点实现了分布式查询计划
  - 重分布
  - 广播
  - 收集
- Motion代表网络（进程间）通信，天然的把查询计划切割成不同的slices
- SPMD(single program, multiple data)完成分布式计算

分布式查询计划和分布式执行器

```
create table student(id int, name text)
distributed by (id);

create table class(id int, name text)
distributed randomly;

select * from
student s, class c
where s.id = c.id;
```





PART 02

Greenplum中的锁

三种层次的锁

- Spinlock：
  - 保护对共享对象的修改
  - 关键区域较短，不必交出CPU使用权
- Lwlock：
  - 保护共享对象的修改
  - 使用信号量
  - 等候需要交出
- Object Lock：
  - 用来保护数据库对象的读写
  - 如表锁，事务锁等
  - 两阶段锁

对象锁的模式和冲突表

定义在: src/include/storage/lockdefs.h 和 src/backend/storage/lmgr/lock.c

Requested Lock Mode	Current Lock Mode							
	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCLUSIVE					X	X	X	X
SHARE UPDATE EXCLUSIVE				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCLUSIVE			X	X	X	X	X	X
EXCLUSIVE		X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

两阶段对象锁模式和生命周期

- select * from t;
  - 语义分析阶段生成查询树的过程中对要访问的表持锁，事务结束才会释放表锁
  - 单纯的select对表加最低级别的AccessShareLock
- insert into t | delete from t | update t …
  - DML语句也是在语义分析阶段对目标表持锁，事务结束才会释放
  - 在执行器修改tuple时候，会讲事务id写入tuple的xmax|xmin, 这相当于对tuple持有了事务锁
  - 在Postgres里，DML对应的锁模式是RowExclusiveLock
  - 在Greenplum里，delete和update有可能升级锁模式避免分布式死锁
    - 如果GlobalDeadlock Detector打开，且目标表是heap表，Greenplum会保持和Postgres一致的锁模式
    - 否则Greenplum对目标表持有ExclusiveLock
- select … from t for (update|share|…)
  - 不能优化的时候，直接对表持有ExclusiveLock而忽视lockingclause，否则
  - 与单纯selec比，锁模式提升到RowShareLock，也是事务结束才会释放
  - 在执行器访问tuple的时候，会根据显示的语义，锁住tuple，这也是事务锁

PART 03

死锁问题和全局死锁检测

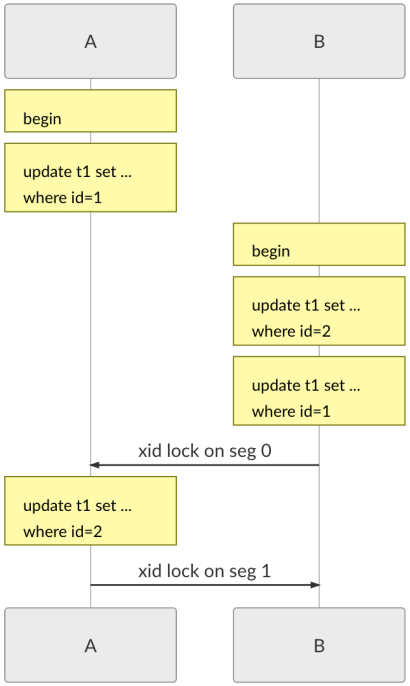
局部死锁和全局死锁

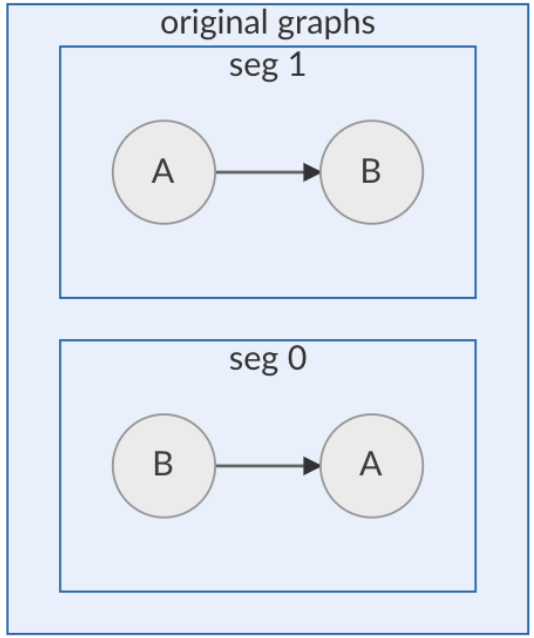
- 局部死锁指的是发生在具体某一个Postgres实例内部，不同的事务直接互相等候
- Postgres有完善的机制可以发现并破解局部死锁
- 全局死锁是指在Greenplum集群中任何一个局部Segment实例内都没有局部死锁，但是在不同的segment之间发生了循环等待，使得死锁的全局事务都无法前进一步

```
create table t(a int, b int);
insert into t values (1,1), (2,2);

tx1: begin;
tx1: update t set b = 999 where a = 1;

tx2: begin;
tx2: update t set b = 888 where a = 2;

tx1: begin;
tx1: update t set b = 777 where a = 2;

tx2: begin;
tx2: update t set b = 666 where a = 1;
```

```
1  tx1: begin;
2  tx2: begin;
3
4  tx1: lock t in ROW EXCLUSIVE mode;
5  tx2: lock t in ROW EXCLUSIVE mode;
6
7  tx1: lock t in EXCLUSIVE mode;
8  tx2: lock t in EXCLUSIVE mode;
```

全局死锁等候图示例


A
B
begin
update t1 set ...
where id=1
begin
update t1 set ...
where id=2
update t1 set ...
where id=1
xid lock on seg 0
update t1 set ...
where id=2
xid lock on seg 1
A
B


original graphs
seg 1
A
B
seg 0
B
A

全局死锁检测

- Greenplum5及以前的版本只能通过对Delete和Update升级锁模式，使得对同一个表的Update和Delete的事物在Master上就串行起来，牺牲性能保证没有分布式死锁
- 大幅度提升Greenplum的OLTP性能必须让Update和Delete真正并发
- Greenplum6引入了全局死锁检测模块
  - Master上运行一个全局死锁检测的Daemon
  - 周期性的采集各个segment（含master）的数据库对象锁的等待关系
  - 利用GDD算法检测是否发生死锁
  - 如果发生死锁，采用既定策略（如结束最年轻的事务）破解死锁

GDD算法的关键点

- 贪心原理保证了不会误判，算法有sound性质
- 只考虑数据库对象锁的等候关系，在分布式系统里还有网络等的等待，因此不能检测出全部的全局死锁，算法严格说并不complete
- 从客户反馈来的全局死锁的案例极少说明GDD算法工作得很好
- 在最终判决之前，在master上锁住procarray，检测死锁环的每个顶点代表的事务都存在，才判决死锁

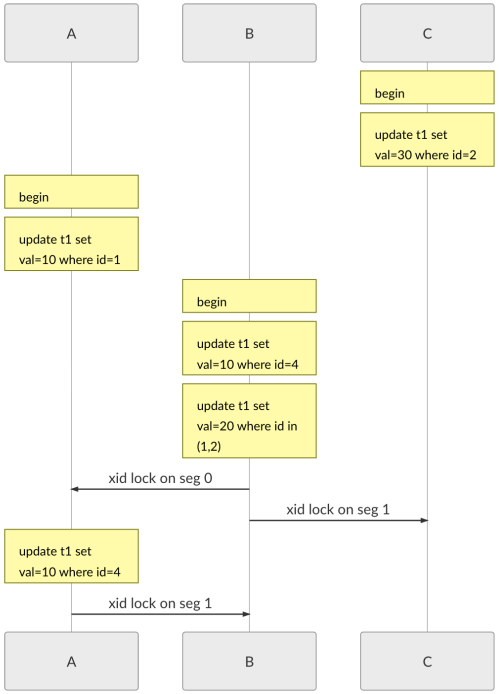
GDD算法

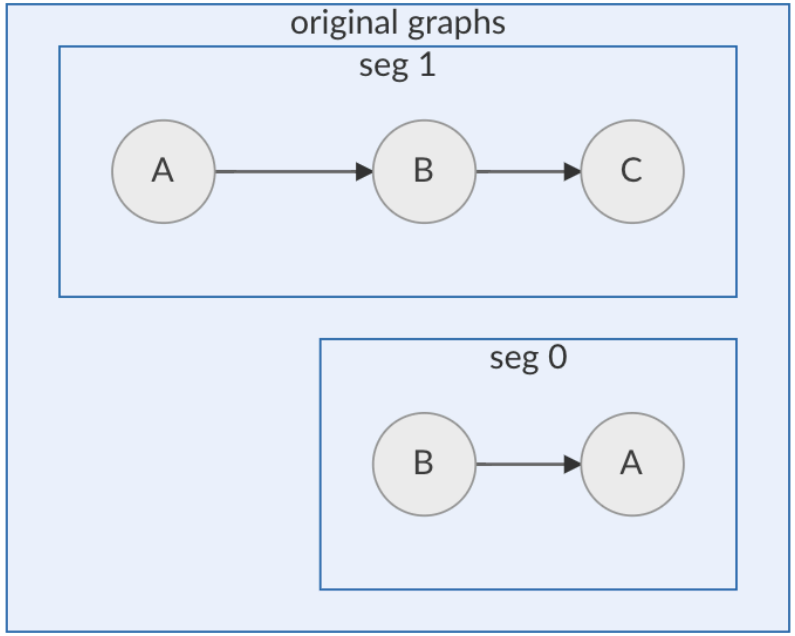
Algorithm 1: Global deadlock detect algorithm

```
build global waiting graph G;
while True do
    /* remove vertices with no outgoing edges */
    for v ∈ Vertex(G) do
        if global_out_degree(v) == 0 then
            remove all edges directly point to v
        end
    end
    /* remove edges in each local waiting graph */
    for local_graph ∈ G do
        for v ∈ Vertex(local_graph) do
            if local_out_degree(v) == 0 then
                remove all dotted edges directly point to v
            end
        end
    end
    if no removals happen then
        break
    end
end
if there remains edges ∧ all transactions exists then
    report global deadlock happens
end
```

- 等候图由很多有向边构成，顶点代表全局事务
- 全局等候图是各个segmen上的局部等候图的集合
- 等候边由两种：实线边和虚线边
- 实线边等待A→B表明只有当事务B结束(提交或回滚)后该等待关系才会消失
- 虚线边等待A··>B描述了除了实线以外的其他等候关系，它的生命周期不依赖事务B是否结束
- 核心思路贪心：没有依赖的事务，假设它一下瞬间立刻提交

GDD算法实例(全局死锁)





PART 04

降锁后的各种问题

EvalPlanQual问题

- 当GDD打开后，并发的对heap表update会同时发生segment上的执行器
- 如果同时更新同一个tuple，后到的事务会等第一个事务完成
- 如果第一个事务提交，且在read committed隔离级别下，第二个事务需要重新判断被第一个事务修改的tuple是否还满足过滤条件(planqual)
- 然后Greenplum里的执行器是分布式执行器，在segment上无法创建需要网络通信的分布式执行器
- 因此对于带有motion的EvalPlanQual需要主动回滚保证正确性

更新分布键问题

- Greenplum是MPP数据库里少有的支持更新表的分布键的数据库
- Greenplum支持更新分布键用的是split-update技术
- 由在旧tuple所在segment的Delete操作和在新tuple所在segment的Insert操作符合构成

```
gpadmin=# create table t(a int, b int) distributed by (a);
CREATE TABLE
gpadmin=# explain (costs off) update t set a = a + 1;
                          QUERY PLAN
---------------------------------------------------------------
 Update on t
   ->  Explicit Redistribute Motion 3:3  (slice1; segments: 3)
         ->  Split
               ->  Seq Scan on t
 Optimizer: Postgres query optimizer
(5 rows)
```

更新分布键问题

- Split-update无法给并发的update事务提供访问新tuple的链，因为新tuple在新segment上，且在本地是用Delete操作修改了旧tuple
- Github Issue： https://github.com/greenplum-db/gpdb/issues/8919
- 修复的思路：
  - 针对split-update升锁防止并发？---- 不可行，因为只有打开了表才知道分布键情况，从而判断是否splitupdate，然而打开表的时候就要确定好锁模式
  - 正确的解决思路：
    - 如果是split-update的delete操作，在tuple的头部写入特殊信息，表明这个tuple是split-update的delete
    - 后续并发的updat操作，可以读取tuple头部信息，从而回滚，保证数据库的正确性

Select语句带有locking clause问题

- Select语句带有locking clause可能会在segment上锁tuple的事务锁
- 在没有GDD的情况下，必须按UPDATE和DELETE的锁模式处理
- Greenplum的查询由motion节点，传递到最顶层lockrows执行节点的tuple可能不是本地的ctid，无法远程锁
- Select for update只在一些简单的情况可以优化：
  - GDD打开
  - Heap表
  - Rangetable list长度是1
  - 没有子查询
- 扩展协议查询的select for update问题： https://github.com/greenplum-db/gpdb/pull/8565

Upsert问题

- Greenplum maste分支已经升级到Postgres12内核，包含了upsert功能
- insert on conflict do update在语义分析阶段是当作INSERT命令处理
- 但它实际上可能在segment上执行update操作锁tuple的事务锁
- 因此在在Master上要把它当成updat语句才处理锁模式问题
- Github Issue: https://github.com/greenplum-db/gpdb/issues/9449
- Greenplum在Merge上游特性的同时，会仔细检查每个特性在MPP环境下是否正确


GREENPLUM
DATABASE®
微信技术讨论群
添加入群小助手：gp_assistant
微信公众号
技术干货、行业热点、活动预告
欢迎访问Greenplum中文社区：cn.greenplum.org

GREENPLUM DATABASE®

CONTACT

THANKS

CONTACT INFORMATION

CHINA POSTGRESQL ASSOCIATION | PCC POSTGRESCONF CN 2020 | PGConf.Asia