PgSQL · 引擎特性 · PostgreSQL Hint Bits 简介

前言

MVCC (Multiversion Concurrency Control) 是数据库系统中常用的并发控制方式，通过保存数据的多个快照版本，实现 读不阻塞写，写不阻塞读。不同数据库系统实现数据多版本的方式不尽相同，MySQL，Oracle 基于回滚段实现，PostgreSQL 则在堆表中实际存储每个元组(tuple)的多个版本，多个版本的元组通过指针构成一个版本链。事务在执行时，依次判断元组的可见性，访问对其可见的元组。判断元组可见性时，通常需要知晓插入或删除该元组的事务的状态(提交或者终止)。
PostgreSQL 将事务状态记录在 CLOG(Commit LOG) 日志中，在内存中维护一个 SLRU Buffer Pool 用于缓存 CLOG 内容。事务在判断元组可见性时，需要从 SLRU Buffer Pool 甚至磁盘中读取事务的状态。由于判断元组可见性的操作可能非常频繁，因此要求读取事务状态的操作尽量高效，为避免每次都从 CLOG 缓存或磁盘文件中读取，引入 Hint Bits，在元组头信息中直接标识插入/删除该元组的事务的状态。本文介绍 Hint Bits 的实现以及其带来的一些问题。

基础介绍

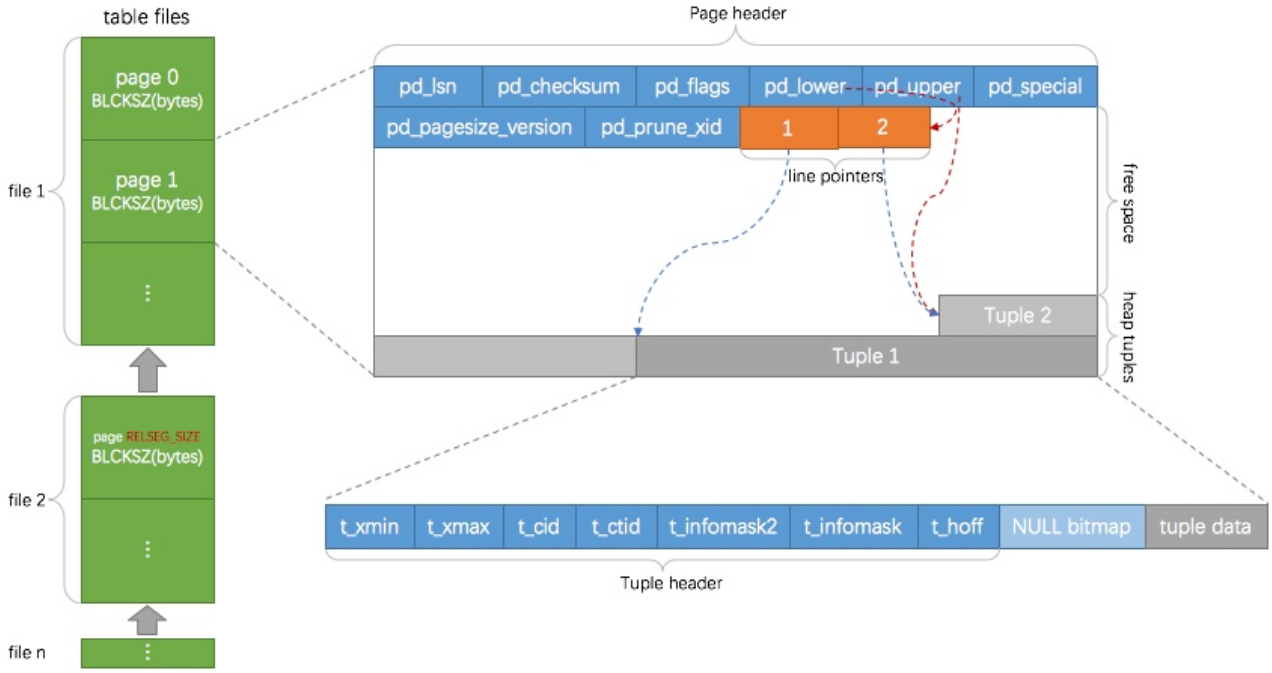
堆表结构

PostgreSQL 中的表是用堆组织的，即堆表。从文件系统来看，每个表由若干文件组成(表大小超过 RELSEG_SIZE 个块就会被切分成多个文件)，每个文件由若干块(block)构成，每个块大小为 BLCKSZ，默认 8KB。
块的结构分为两部分：页头(Page Header)和元组(Heap Tuple)，页头存储页的元信息，元组中存储真正的数据。元组又由两部分组成：元组头(Tuple Header)和元组数据(Tuple Data)，如下图所示：
t_xmin 记录插入该元组的事务 ID

t_xmax 记录删除该元组的事务 ID

t_ctid 指向新版本的数据，如果没有则指向自己

t_infomask 记录元组的状态信息，包括 Hint Bits



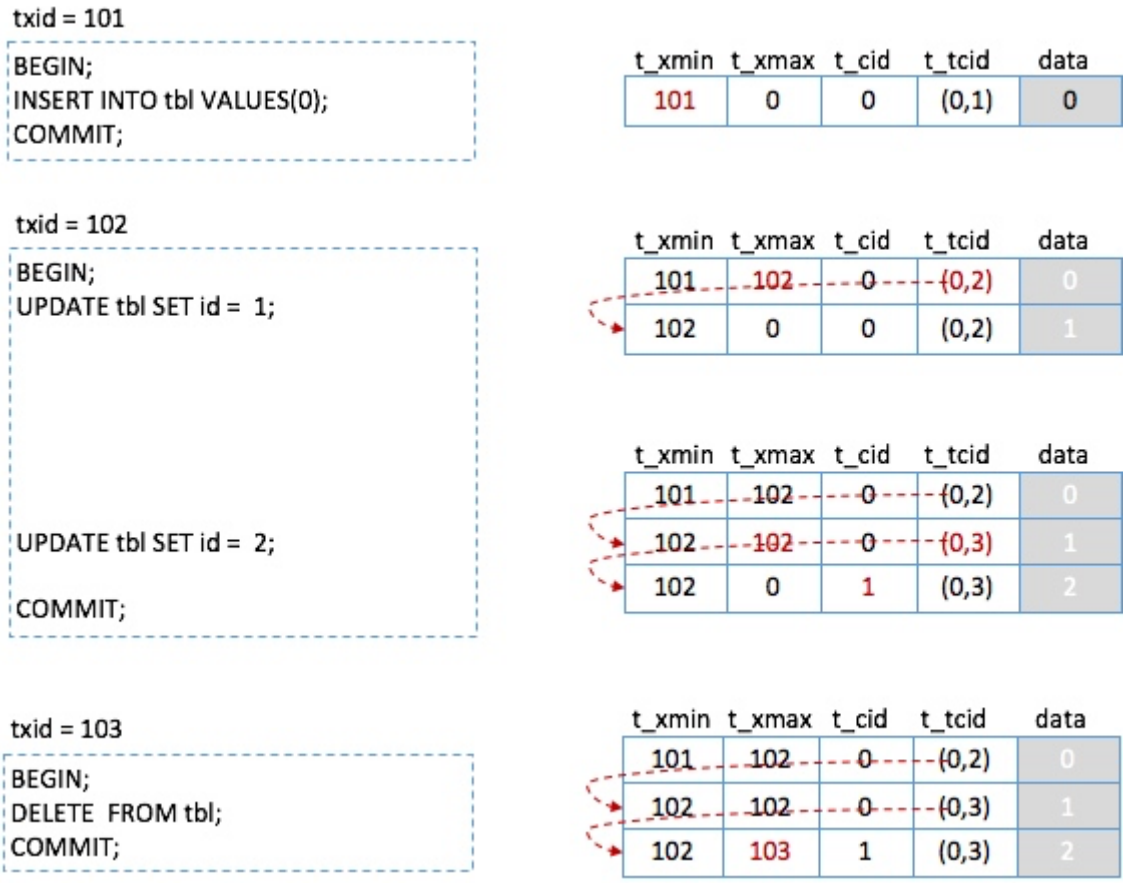
以上简单介绍 PostgreSQL 中堆表的大体结构以及本文依赖的一些字段，其他在此不做详细介绍，读者可参考其他文献和相关源码，如 src/include/access/htup_details.h，src/include/access/htup.h，src/include/storage/bufpage.h。

数据多版本

以下通过示例说明 PostgreSQL 中数据多版本的结构。

- 事务 101 插入元组，因此该元组的 t_xmin 为 101
- 事务 102 有两条更新语句，PostgreSQL 中的更新操作相当于 删除+插入，删除操作是标记删除，将元组的 t_xmax 设置为删除该元组的事务 ID，即 102；随后插入新的元组，旧元组的 t_tcid 指向新元组，构成多版本链

- 事务 103 将元组删除，因此将元组 t_xmax 设置为该事务的 ID 103 即可



可见，堆表中的数据经过增删改等操作，会保存多个版本，相互之间通过指针构成多版本链。多版本数据如果太多，会导致表膨胀严重，因此，PostgreSQL 中引入了 VACUUM 机制对数据进行清理，VACUUM 的相关实现可以参考之前月报的分析，如 AutoVacuum 机制之 autovacuum worker （http://mysql.taobao.org/monthly/2018/02/04/）和 AutoVacuum 机制之 autovacuum launcher （http://mysql.taobao.org/monthly/2017/12/04/）。

可见性判断

基于现在堆表的实现，一个事务查询数据时，如何判断哪些数据是对自己可见的呢？基于 interdb 的总结，可见性判断大致包括以下规则：

```
/* t_xmin status = ABORTED */
Rule 1: IF t_xmin status is 'ABORTED' THEN
            RETURN 'Invisible'
        END IF

        * t_xmin status = IN_PROGRESS */
        IF t_xmin status is 'IN_PROGRESS' THEN
             IF t_xmin = current_txid THEN
Rule 2:          IF t_xmax = INVALID THEN
                  RETURN 'Visible'
Rule 3:          ELSE  * this tuple has been deleted or updated by the current
transaction itself. */
                   RETURN 'Invisible'
                 END IF
Rule 4:        ELSE   * t_xmin ≠ current_txid */
                  RETURN 'Invisible'
               END IF
        END IF

        * t_xmin status = COMMITTED */
        IF t_xmin status is 'COMMITTED' THEN
Rule 5:     IF t_xmin is active in the obtained transaction snapshot THEN
                RETURN 'Invisible'
Rule 6:     ELSE IF t_xmax = INVALID OR status of t_xmax is 'ABORTED' THEN
                RETURN 'Visible'
            ELSE IF t_xmax status is 'IN_PROGRESS' THEN
Rule 7:         IF t_xmax =  current_txid THEN
                    RETURN 'Invisible'
Rule 8:         ELSE  * t_xmax ≠ current_txid */
                    RETURN 'Visible'
                END IF
            ELSE IF t_xmax status is 'COMMITTED' THEN
Rule 9:         IF t_xmax is active in the obtained transaction snapshot THEN
                    RETURN 'Visible'
Rule 10:        ELSE
                    RETURN 'Invisible'
                END IF
            END IF
        END IF
```

以上可见性判断的过程中，需要知晓 t_xmin 和 t_xmax 的事务状态，是 COMMITTED，ABORTED 还是 IN_PROGRESS，这些状态就需要通过 CLOG 来获取。可见性判断的细节在此也不做展开，具体可参考原文链接（http://www.interdb.jp/pg/pgsql05.html）或者参考源码实现 HeapTupleSatisfiesMVCC。

Commit Log (CLOG)

PostgreSQL 在 CLOG 中维护事务的状态，持久化存储在 pg_xact(PostgreSQL 10 之前是 pg_clog) 目录下，为了访问高效，会在内存中维护一块共享内存用于缓存 CLOG 的内容。PostgreSQL 中定义了以下四种事务状态:

```
#define TRANSACTION_STATUS_IN_PROGRESS    0x00
#define TRANSACTION_STATUS_COMMITTED    0x01
#define TRANSACTION_STATUS_ABORTED      0x02
#define TRANSACTION_STATUS_SUB_COMMITTED  0x03
```

CLOG 文件由一个或者多个块构成，CLOG 的内容从逻辑上构成一个数组，数组的下标是事务 ID（即可以根据事务 ID 计算出事务状态在文件中的偏移量，可以参考 TransactionIdGetStatus 这个函数），数组的内容是事务状态，四种事务状态仅需两个 bit 位即可记录。以一个块 8KB 为例，可以存储 8KB * 8/2 = 32K 个事务的状态。内存中缓存 CLOG 的 buffer 的大小为 Min(128, Max(4, NBuffers 512))。
CLOG 文件以 0000，0001 这种方式命名，每个文件最大 32（SLRU_PAGES_PER_SEGMENT）个块，默认 256KB。PostgreSQL 启动时会从 pg_xact 中读取事务的状态加载至内存。
系统运行过程中，并不是所有事务的状态都需要长期保留在 CLOG 文件中，因此 VACUUM 操作会清理不再使用的 CLOG 文件。

Hint Bits

以上介绍了 PostgreSQL 的表结构，数据多版本，可见性判断以及 CLOG。正如文章开头所讲，在进行可见性判断时，需要获取事务的状态，即元组中 t_xmin 和 t_xmax 的状态，这些事务状态保存在 CLOG 中，为加速获取事务状态的过程，PostgreSQL 引入了 Hint Bits。
所谓 Hint Bits，就是把事务状态直接记录在元组头中（HeapTupleHeaderData），避免频繁访问 CLOG 影响性能，元组头中对应的标识位如下:

```
#define HEAP_XMIN_COMMITTED    0x0100  /* t_xmin committed */
#define HEAP_XMIN_INVALID    0x0200  /* t_xmin invalid/aborted */
#define HEAP_XMIN_FROZEN    (HEAP_XMIN_COMMITTED|HEAP_XMIN_INVALID)
#define HEAP_XMAX_COMMITTED    0x0400  /* t_xmax committed */
#define HEAP_XMAX_INVALID    0x0800  /* t_xmax invalid/aborted */
```

需要注意的是，元组中的 Hint Bits 采用延迟更新的策略。PostgreSQL 并不会在事务提交或者回滚时主动更新所有操作过的元组的 Hint Bits，而是等到第一次访问（可能是 VACUUM，DML 或者 SELECT）该元组并进行可见性判断时，如果发现 Hint Bits 没有设置，则从 CLOG 中读取事务的状态，如果事务状态为 TRANSACTION_STATUS_COMMITTED 或 TRANSACTION_STATUS_ABORTED，则将其记录在 Hint Bits 中，对于正在执行的事务由于其状态还未到达终态，无需记录在 Hint Bits 中。否则直接读取 Hint Bits 的值。可见性判断过程中设置 Hint Bits 的函数入口为 SetHintBits。
因此，Hint Bits 可以理解为是事务状态在元组头上的一份缓存，减少访问链路的长度，让事务状态触手可及。

Hint Bits 与日志

如 PostgreSQL checksum（https://developer.aliyun.com/article/675942） 一文所说，在开启 CHECKSUM 或者 GUC 参数 wal_log_hints 为 true 的情况下，如果 CHECKPOINT 后第一次使页面 dirty 的操作是更新 Hint Bits，则会产生一条 WAL 日志，将当前数据块写入 WAL 日志中(即 Full Page Image)，避免产生部分写，导致数据 CHECKSUM 异常。读者可以参考德哥的文章（https://blog.csdn.net/postgrechina/article/details/49130743）了解更多细节。
因此，在开启 CHECKSUM 或者 GUC 参数 wal_log_hints 为 true 时，即便执行 SELECT，也可能更改页面的 Hint Bits，从而导致产生 WAL 日志，这会在一定程度上增加

WAL 日志占用的存储空间。如果读者在使用 PostgreSQL 中，发现执行 SELECT 会触发磁盘的写入操作，可以检查一下是否开启了 CHECKSUM 或者 wal_log_hints。
注意，以上写 Full Page Image 日志的行为与是否开启 full_page_writes 没有关系。相关代码实现可以参考 MarkBufferDirtyHint 这个函数。

总结

Hint Bits 是 PostgreSQL 中一个很小的功能特性，其目的是提升获取事务状态的效率，进而加速可见性判断。功能点虽小，但牵涉的模块众多，本文尽可能简单但又不失全面地介绍相关模块的背景知识，如堆表结构，多版本数据结构，可见性判断以及 CLOG，进而引出 Hint Bits 的作用、相关实现以及其与日志的关系。读者如果对相关模块的更多实现细节感兴趣，可以参考内核月报的其他文章以及相关文献。

评论