武汉大学计算机学院
本科生课程报告

postgreSQL 源码阅读分析报告
——MultiXact 日志管理器

专 业 名 称　：计算机科学与技术弘毅班

课 程 名 称　：数据库系统实现

指 导 教 师　：彭智勇

学 生 学 号　：2020300004052

学 生 姓 名　：李哲

二〇二三年三月

一、整体介绍

MULTIXACT 日志是 PostgreSQL 系统用来记录组合事务 ID 的一种日志。由于 PostgreSQL 采用了多版本并发控制，因此同一个元组相关联的事务 ID 可能有多个，为了在加锁（行共享锁）的时候统一操作，PostgreSQL 将与该元组相关联的多个事务 ID 组合起来用一个 MultiXactID 代替来管理。MultiXact 允许多个事务并发地持有一个记录的锁，而不会发生死锁。

而 MultiXact 日志管理器便是用来记录了哪些事务持有了哪些 MultiXact 锁的，方便对事务进行管理。MultiXact 日志管理器的**核心作用**是存储和检索 MultiXactMember 数组，以支持行级锁机制的正确实现。它实现了多事务并发控制（MVCC）协议的一部分，确保多个事务可以同时操作数据库而不会发生冲突。

接下来将分成不同的模块分别介绍 MULTIXACT 日志管理器实现的功能、所使用的数据结构和算法。

二、具体论述

（一）MultiXactID 的分配和回收

MultiXactID 是一个多对一的映射关系，需要在事务 ID 数组中标记哪一段映射到一个 MultiXactID，也需要能够方便地管理和维护 MultiXact 状态。

所以我们定义了数据结构 MultiXactStateData 来对 MultiXactID 进行分配和回收，MultiXactStateData 的数据结构和含义如表 1 所示：

表 1　MultiXactStateData

```
typedef struct MultiXactStateData
{
        MultiXactId     nextMXact;              //下一个可分配的 MultiXactID
        MultiXactOffset nextOffset;             //下一个对应 MultiXactID 的起始偏移量
        MultiXactId     lastTruncationPoint;    //上一次删除位置
        MultiXactId     perBackendXactIds[1];   //MultiXactID 队列起始位置
}MultiXactStateData;
```

涉及的算法：

- 分配：MultiXact ID 的分配使用 MultiXactStateData 中的“nextMXact”计数器，每次分配时将计数器加 1，并将结果分配给新的 MultiXact ID。
- 回收：MultiXact ID 回收使用的算法是标记-清除算法，其中 MultiXactStateData 中的“oldestMultiXactId”跟踪最早的尚未回收的 MultiXact

ID，回收时会将已经不再使用的 MultiXact ID 标记为“过时的”，并将“oldestMultiXactId”向前移动到下一个最早的未回收 MultiXact ID。

（二）MultiXactID 的扩增

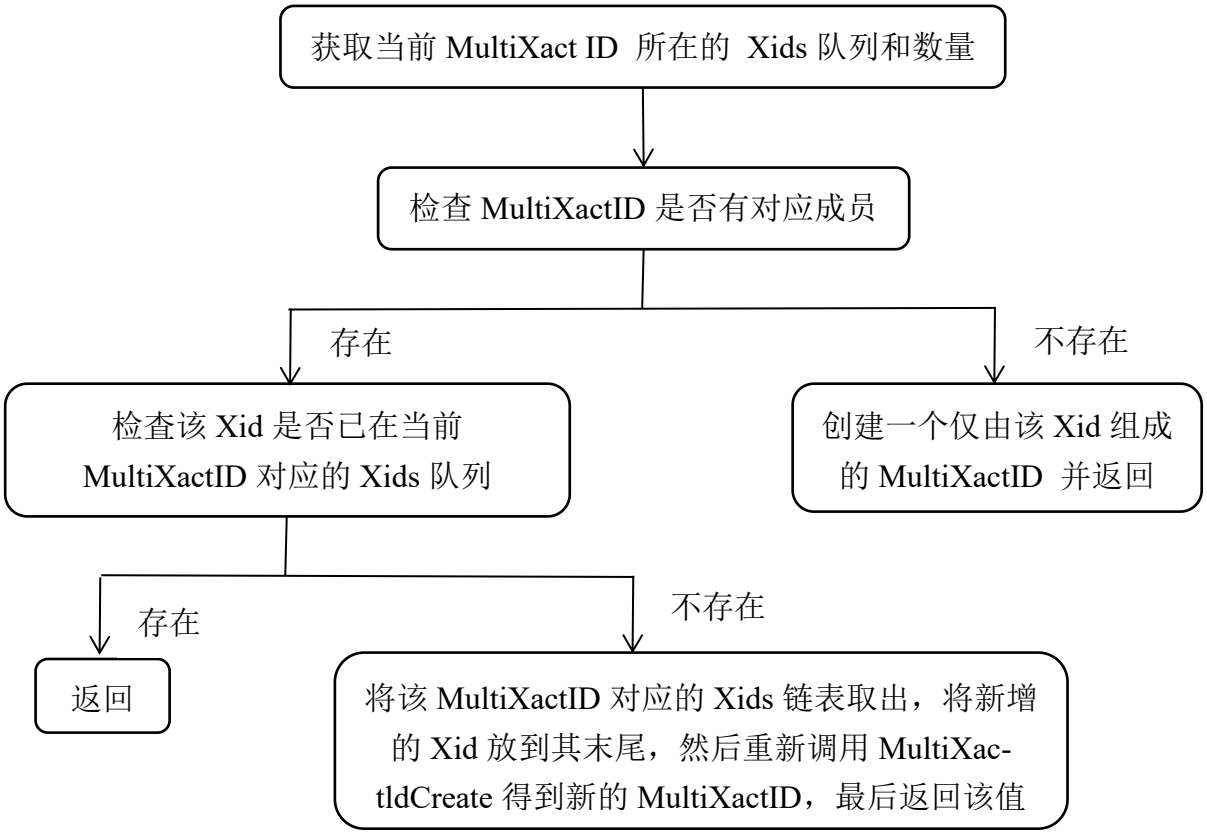
因为 MultiXact ID 只有 4 个字节，最大值为 2^32-1。当系统需要分配的 MultiXact ID 超过时，需要进行扩增操作，以增加 MultiXact ID 的可用范围。

所用到的数据结构主要是 MultiXactStateData 和 MultiXactOffsetCtl，其中 MultiXactStateData 已经在（一）中提及，MultiXactOffsetCtl 用于管理 MultiXact ID 偏移量，其数据结构如表 2 所示：

表 2 MultiXactOffsetCtl

```
typedef struct MultiXactOffsetCtl
{
    int             maxoffsets;   //表示 MultiXactOffset 数组的大小
    MultiXactOffset freelist;     //表示空闲 MultiXactOffset 的链表头
    MultiXactOffset nextOffset;   //表示下一个可用的 MultiXactOffset
}MultiXactOffsetCtl;
```

涉及的算法主要在函数 ExtendMultiXactId()中，流程图：



（三）MultiXact 的成员管理

MultiXact 成员管理模块主要负责管理和维护 MultiXact ID 中包含的成员列表，确保 MultiXact ID 包含的所有事务 ID 都能被正确地添加、移除和查找。此外，它还负责对 MultiXactMember 进行内存管理，确保内存分配和释放的正确性和高效。其数据结构均是和 MultiXactMember 直接相关的，具体见表 3。

表 3　成员管理相关数据结构

MultiXactMember	//表示一个 MultiXact ID 包含的所有事务 ID
MultiXactMemberList	//表示一个 MultiXact ID 包含的所有成员
MultiXactMemberCtl	//用于管理 MultiXact 成员列表的分配和释放

涉及的算法：

- 添加成员：在 MultiXact 成员链表中添加一个新的 MultiXactMember。添加操作需要进行内存分配和链表操作。
- 移除成员：从 MultiXact 成员链表中删除指定的 MultiXactMember。移除操作需要进行链表操作和内存释放。
- 查找成员：在 MultiXact 成员链表中查找指定的 MultiXactMember。查找操作需要遍历链表进行比较。
- 内存管理：对于每个 MultiXactMember，需要进行内存的分配和释放。MultiXactMemberCtl 通过分配和释放内存块来管理 MultiXactMember。

成员管理涉及到的内容及函数较多，在此列举一下我在源码分析过程中所整理的函数：

① ReadNextMultiXactId，用于读取将被分配的下一个 MultiXactId。

② ReadMultiXactIdRange，获取当前系统中 MultiXact 的 ID 范围，以及可以被关系引用的最旧的 MultiXact ID。

③ RecordNewMultiXact，实现 MultiXact 的成员信息的持久化写入。具体来说，它将给定的 MultiXactId、MultiXactOffset 和成员列表写入到 MultiXact 成员表中。

④ GetNewMultiXactId，获取下一个 MultiXactId 的函数，同时也预留了“成员”区域所需的空间。

⑤ GetMultiXactIdMembers，检索一个给定的多重事务 ID（MultiXactId）的成员列表（MultiXactMember）

⑥ mxactMemberComparator，用于比较 MultiXactMember 结构体的比较函数，可以用于对 MultiXactMember 结构体的数组进行排序

⑦ MultiXactShmemSize，计算 MultiXact 相关共享内存的大小

⑧ MultiXactShmemInit，初始化 MultiXactOffset 和 MultiXactMember 的共享缓冲区

（四）MultiXact 的状态转换

MultiXact 状态转换模块的主要功能是处理 MultiXact 的状态转换，并通过各种数据结构和算法来管理和维护 MultiXact 的状态信息。MultiXact 的状态转换主要包含以下几个功能：

- 状态转移：根据 MultiXact 成员集合的变化，将 MultiXact 状态从“进行中”（in-progress）变为“已提交”（committed）或“已中止”（aborted）。
- 检查当前状态是否允许添加新成员或移除已有成员。
- 对状态进行转换时，要确保线程安全，防止并发修改状态导致数据一致性问题。

所用到的主要数据结构见表 4 所示：

表 4　状态转换相关数据结构

MultiXactStateData	//存储 MultiXact 的状态信息
MultiXactMember	//存储 MultiXact 成员集合
MultiXactMemberctl	//MultiXactMember 的控制结构
MultiXactOffset	//记录 MultiXact 成员在 MultiXactMember 中的索引位置
MultiXactOffsetctl	//MultiXactOffset 的控制结构
PGPRO 和 ProcArray	//用于实现线程安全的状态转换

所涉及到的算法主要包含在下面几个函数中，算法流程不再详细介绍：

① MultiXactIdExpand（在（二）中介绍过），用于扩增 MultiXact ID。

② MultiXactIdIsRunning，用于判断 MultiXact ID 是否处于“进行中”状态。

③ MultiXactIdWait，用于等待其他事务释放对 MultiXact 的锁定。

④ MultiXactLockWaiterCount，用于等待其他事务释放对 MultiXact 的锁定。

⑤ MultiXactSetNextMXact，用于设置下一个可用的 MultiXact ID。

⑥ MultiXactIdSetOldestVisible，用于设置最早可见的 MultiXact ID。

这些函数共同运行，才能保证状态转换的正确执行，比如，MultiXactIdSetOldestMember 和 MultiXactIdSetNextMXact 函数在 MultiXact 状态从“正在运行”变为“已提交”时，更新了 MultiXact 的状态信息。而 MultiXactIdCreate 函数则在 MultiXact 状态从“已提交”变为“新建”时，生成了新的 MultiXactId 并将其分配给事务。这些函数使用了 MultiXactStateData 和 MultiXactMemberCtl 等数据结构，以实现 MultiXact 状态的转换和管理。

（五）MultiXact 状态的缓存

MultiXact 状态的缓存主要是为了提高性能和降低锁竞争的频率。在多个事务同时申请同一个 MultiXactID 时，如果每次都要访问磁盘读取和更新状态信息，会严重降低系统的性能，而且可能会导致死锁和竞争条件的出现。

所以我们定义了 mXactCacheEnt 数据结构，用来缓存 MultiXact 状态。具体来说，mXactCacheEnt 对象用于实现 MultiXact 状态的本地缓存，以避免多次访问磁盘上的 MultiXact 状态记录。当 MultiXact 日志管理器需要访问 MultiXact 状态时，它会先在 mXactCacheEnt 中查找相应的记录，如果找到了则直接返回该记录；如果没有找到，则需要从磁盘上读取该记录，并将其添加到 mXactCacheEnt 中以供下次使用。

mXactCacheEnt 的数据结构和含义如表 5 所示：

表 5　mXactCacheEnt

```
typedef struct mXactCacheEnt
{
    struct mXactCacheEnt    *next;      //表示缓存的 MultiXact 状态的成员事务列表
    MultiXactId             multi;      //表示缓存的 MultiXact 状态的标识符
    int                     nxids;      //表示 mXactCacheEnt 对象的引用计数
    TransactionId           xids[1];
}mXactCacheEnt;
```

涉及的算法：

- 查找：使用哈希表（hash table）实现，其中哈希表的键是 MultiXactId，值是对应的 mXactCacheEnt 对象的指针。
- 插入和更新：当需要插入或更新某个 MultiXact 状态时，MultiXact 日志管理器会首先查找 mXactCacheEnt 中是否已经存在相应的记录。如果不存在，则需要分配一个新的 mXactCacheEnt 对象，并将其插入哈希表中；如果已经存在，则需要更新相应的记录。
- 删除：当需要删除某个 MultiXact 状态时，MultiXact 日志管理器会首先查找 mXactCacheEnt 中是否存在相应的记录。如果存在，则需要将其从哈希表中删除，并将相应的 mXactCacheEnt 对象释放。
- 缓存替换：MultiXact 日志管理器使用 LRU（Least Recently Used）算法实现缓存替换，即将最近最少使用的对象替换掉。

（六）MultiXact 日志的记录与恢复

MultiXact 日志管理器需要实现持久化，以保证在系统重启或崩溃后，MultiXact 记录的状态和元数据信息能够恢复到先前的状态。这就涉及到日志记录与恢复的相关问题。为了实现持久化，MultiXact 日志管理器需要记录和管理以下数据结构，如表 6 所示：

表 6　日志的记录与恢复相关数据结构

XLogReaderState	//表示 XLOG 的读取器，从 XLOG 中读取并解析记录
SimpleLruWritePage	//简单 LRU 算法实现的共享缓存管理器，用于写入多个事务的事务 ID 和锁定信息的共享缓存页
MultiXactMemberCtl	//分配、释放和操作 MultiXactMember 的函数
MultiXactStateCtl	//管理 MultiXactState 的分配和释放，以及将 MultiXactState 持久化到磁盘的函数
MultiXactOffsetCtl	//管理 MultiXactOffset 的分配和释放，以及 MultiXactOffset 持久化到磁盘的函数

涉及的算法：

- MultiXactState 的持久化：通过 WAL (Write Ahead Log) 实现持久化的，即在每次修改 MultiXactState 时，都会将其相关信息写入 WAL 中。在系统恢复时，可以通过 WAL Replay 来恢复 MultiXactState 的状态。
- MultiXactOffset 的持久化：通过 Shared Memory Snapshot 实现持久化的，即在系统关闭前将其保存到磁盘中，并在系统启动时读取该文件以恢复 MultiXactOffset 的状态。
- 日志的恢复：通过 WAL（Write-Ahead Logging）技术实现。WAL 技术的基本原理是在数据库引擎执行对数据的修改操作之前，先将这些操作写入到 WAL 日志中，然后再执行这些操作。这样，即使数据库引擎在执行修改操作的过程中发生了故障或者崩溃，只要 WAL 日志没有损坏，就可以通过重放 WAL 日志中的操作，将数据库恢复到故障发生之前的状态。
- 当 WAL 缓冲区被填满时，WAL 日志记录会被写入到 WAL 日志文件中，并且这个操作会被认为是一个原子性的写操作，因为在数据页被修改之前，WAL 日志记录就已经被写入到了 WAL 日志文件中。
- 当数据库需要进行恢复时，系统会先通过 WAL 日志文件来恢复 MultiXact 状态。具体来说，系统会先将 WAL 日志文件中的 WAL 日志记录读入到内存中，然后根据这些 WAL 日志记录的内容来还原 MultiXact 状态。如果 WAL 日志文件中的所有 WAL 日志记录都被处理完了，那么系统就可以认为 MultiXact 状态已经被完全恢复了。
- 检查点操作：在进行检查点操作时，系统会将当前时刻的所有 MultiXact 状

态信息写入到持久化存储介质中。这个操作会被认为是一个原子性的写操作，因为在数据页被修改之前，这个操作就已经被写入到了持久化存储介质中。这样，即使数据库在检查点之后发生了崩溃，也可以通过读取持久化存储介质来恢复 MultiXact 状态。

下面是我在源码分析中筛选出的一些关于日志恢复和检查点的函数：

① `multixact_redo`，过读取记录并根据记录类型执行相应的操作，对多重事务进行恢复

② `MultiXactGetCheckptMulti`，获取当前多重事务状态的检查点信息。

③ `CheckPointMultiXact`，将内存中的脏 `MultiXact` 数据写入磁盘中的文件，并且会将这些写操作的信息加入到后续的同步请求队列中。

④ `MultiXactSetNextMXact`，用于设置下一个可用的 `MultiXact ID`，并将其写入 `WAL` 日志，以便在崩溃恢复时进行恢复

三、经典的数据结构和算法

上述数据结构和算法是在各自的功能板块进行分析的，为了更加整体地表现 MultiXactID 所采用的数据结构和算法，我选取了其中最为重要的数据结构和算法在此分析。

数据结构：

- TransactionId：32 位的无符号整数，该类型表示事务的 id，用来唯一标识一个事务的身份，用于在事务执行期间进行事务状态的跟踪和管理
- MultiXactId：32 位的无符号整数，用于唯一标识一个 MultiXact 状态。MultiXact 状态指的是一个包含多个事务参与的共享锁或排他锁的状态，它可以表示一个事务需要等待的锁定状态。每当有多个事务需要共享或互斥访问同一资源时，multixact 日志管理器会分配一个新的 MultiXactId，并将所有参与的事务的 TransactionId 记录在 MultiXact-Member 数据结构中。当多个事务同时尝试在同一行上进行修改时，可能会出现并发冲突。在这种情况下，事务需要协调以确定谁有权继续进行修改，以避免数据的不一致或丢失。
- MultiXactStateData：用于管理 MultiXact 状态的数据结构，MultiXactStateData 维护了 MultiXact 的全局状态，包括分配、释放、查询 MultiXactID 的状态信息等。同时，MultiXactStateData 还支持了日志记录和恢复机制，以确保 MultiXact 状态的稳定性和持久性。
- MultiXactMember：由一个 32 位无符号整数和一个 16 位的标志位组成。其中，32 位无符号整数表示一个事务的 ID，16 位标志位表示该事务对该数据的访问模式。在实际应用中，MultiXactMember 通常被用来实现锁和事务的控制。例如，在执行一个更新操作时，需要先获取该数据的独占锁，这就需

要使用 MultiXactMember 来跟踪该事务 ID 和锁模式。当该事务提交时，相应的 MultiXactMember 将被释放。

- MultiXactOffset：32 位无符号整数，表示一个 MultiXactId 在 multixact 的位图中的偏移量，它对应了一个具体的位图块（bitmap block），一个 MultiXactId 可以对应多个位图块，因此 MultiXactOffset 需要与 MultiXactMember 一起使用才能确定一个 MultiXactId 的确切含义。
- mXactCacheEnt：用于缓存 MultiXact 状态的结构体，它保存了一个 MultiXact 中的所有 MultiXactMember 的状态，主要用于多重事务的管理和维护。它可以帮助加速查询多重事务的速度，并且可以避免多次从磁盘读取多重事务的信息。当需要查询一个多重事务时，首先会检查 mXactCache 中是否已经存在该多重事务，如果存在则直接返回缓存中的结果，否则需要从磁盘上读取多重事务的信息并将其加入到 mXactCache 中。
- MultiXactMemberCtl：用于管理 MultiXactMember 的结构体，它包含了当前可用的 MultiXactMember 的数组和空闲链表，负责管理可用的 MultiXactMember，通常与 MultiXact 日志缓冲区配合使用，实现高并发事务的处理。
- MultiXactOffsetCtl：用于管理 MultiXact Offset 的结构体，它包含了当前可用的 MultiXact Offset 的数组和空闲链表，负责管理可用的 MultiXact Offset，通常与 MultiXact 日志缓冲区配合使用，实现高并发事务的处理。

算法

- 基于多版本时间戳（MVCC）的并发控制算法：
- 在 MVCC 算法中，每个事务都被分配一个唯一的时间戳，用于标识它的读写权限。当一个事务需要读取某个数据时，它会比较自己的时间戳和该数据的版本号，只有当自己的时间戳早于数据的版本号时才能读取该数据。当一个事务需要修改某个数据时，它会将数据的旧版本复制一份，并给它分配一个新的版本号，然后在新版本上进行修改操作，最后将新版本的数据提交到数据库中。
- SLRU 算法：
- 在 MultiXact 中，SLRU 算法用于管理多版本事务 ID 缓存（MultiXact ID Cache）。MultiXact ID Cache 采用了分段（Segmented）的 SLRU 算法，将缓存分为多个段，并对每个段单独进行管理。每个段都有自己的 LRU 链表和 LFU 链表，用于记录缓存中最近使用和最少使用的 MultiXact ID。
- 当需要将一个 MultiXact ID 加入缓存时，SLRU 算法会先查找该 ID 是否已经存在于缓存中。如果存在，则更新该 ID 的使用信息，并将其移动到 LRU 链表的头部。如果不存在，则将该 ID 插入到 LRU 链表的头部，并将其计数器（counter）初始化为 1。当缓存达到最大容量时，SLRU 算法会从缓存中选

择一些 MultiXact ID 进行淘汰。

- 采用 SLRU 算法，MultiXact 可以高效地管理 MultiXact ID 缓存，避免频繁的缓存淘汰操作对性能的影响，并确保缓存中的 MultiXact ID 信息是最近和最常使用的。
- 延迟清理（Delayed Cleanup）算法：
- 该算法可以延迟清理不再需要的多版本信息，以提高性能。当一个事务完成时，它会将其持有的多版本信息放入待清理队列中，而不是立即清理。在待清理队列中的多版本信息会在后续的操作中进行批量清理，以避免频繁的清理操作对性能的影响。
- 自适应哈希（Adaptive Hashing）算法：
- 在 MultiXact 中，由于 MultiXact ID 缓存中的 MultiXact ID 数量可能会不断变化，为了保证哈希表的高效性和空间利用率，采用了自适应哈希算法来动态调整哈希表大小。
- 哈希表调整阶段：当哈希表的负载因子（load factor）超过一个预先设定的阈值时，就进入哈希表调整阶段。在这个阶段，MultiXact 会新建一个更大的哈希表，并将原有哈希表中的数据重新哈希到新的哈希表中。这个过程中，MultiXact 会将原有的哈希表锁住，以保证数据一致性和并发性。
- 哈希表重建阶段：当哈希表的负载因子低于一个预先设定的阈值时，就进入哈希表重建阶段。在这个阶段，MultiXact 会释放原有哈希表的内存，并使用新的哈希表来代替原有哈希表。这个过程中，MultiXact 会将原有的哈希表锁住，以保证数据一致性和并发性。
- 通过采用自适应哈希算法，MultiXact 可以根据实际的负载情况动态调整哈希表大小，从而保证哈希表的高效性和空间利用率，提高数据库系统的性能和可扩展性。

四、总结

通过两周的源码阅读和资料查阅，我从零开始逐渐了解到 postgreSQL 中 MultiXact 日志管理器的功能及实现。它通过维护 MultiXactID 的分配回收等各种操作，管理 MultiXact 的成员，对 MultiXact 状态进行方便查阅的缓存和稳定的持久性存储，以及出现错误的日志恢复，支持了行级锁机制的正确实现，确保数据库的并发性和可靠性，防止了死锁或其他竞争条件的发生。

五、参考文献

[1] PostgreSQL 数据库内核分析.彭智勇

[2] CSDN 网站文章