ARIES简介

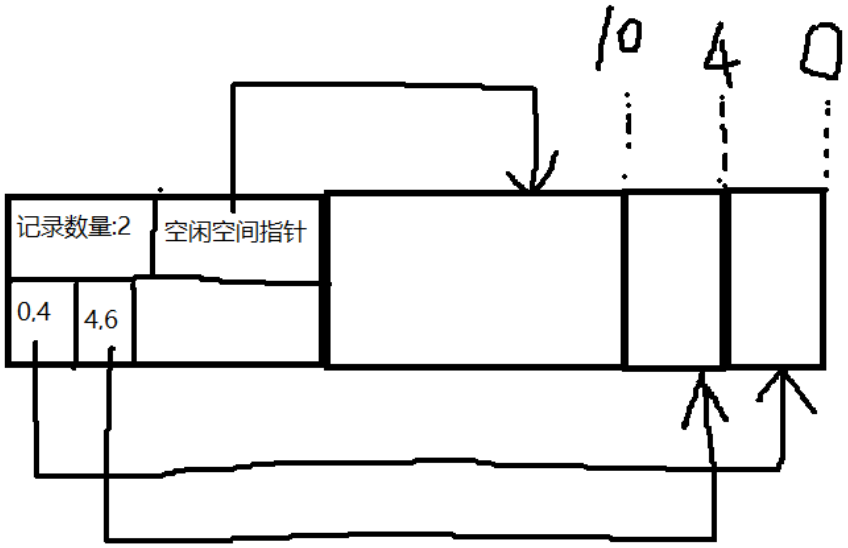
ARIES是一种数据库日志恢复系统，在PG中用到的WAL应该是ARIES的简化版本。

ARIES为何不采用物理日志

要了解物理日志的缺点，首先我们先针对广泛应用的分槽页结构的特点进行说明。

分槽页结构是一种在一页中存储很多变长记录的页结构，下面是一个例子：

{% asset_img image-20210917100042046.png %}



如果将存储在0号位置的记录删除掉，然后将0号位置后面的所以记录前移，那么物理日志就会将记录两个信息：0号位置的记录删除了，4号位置的记录移动到前面。这样的话，日志就会很长。

ARIES算法的特点

1. 使用日志顺序号（LSN）来标识日志记录，如果某条日志对应的操作已经更新在了数据库页中了，那么就可以把这条日志的LSN存储在页中。
2. 支持物理逻辑操作。受影响的页从物理上标识出来，比如删除操作的日志会记录删除的元组存储在哪个页中，但是对于删除操作本身，不会记录它对其他元组的影响（比如元组要前移来填补删除元组的位置），只是记录有这样一个删除操作。

数据结构

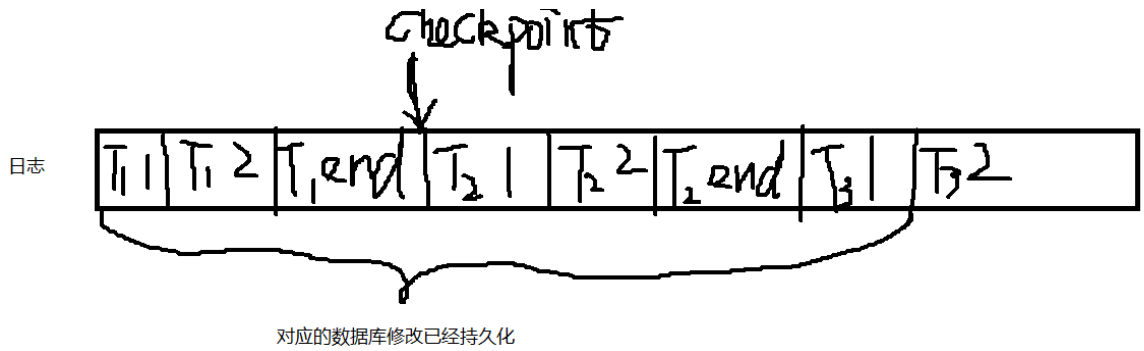
其实最开始的日志是比较简单的，包括undo日志和redo日志，这个时候的日志系统设计主要考虑一下几点：

1. 一条日志记录持久化了之后，对应的数据库操作才可以持久化到数据库中；
2. 在checkpoint时，将整个系统锁住，不允许有任任何对缓冲块的写操作，保证日志缓存和脏页都持久化。这里有个问题，脏页中都反映了哪些事务修改呢，我们有两个可能：1.A日志在缓存中，但是A日志的事务修改却没有反应在缓存中, 2. A日志在缓存中，则A日志的事务修改也反映在缓存中。答案其实是第二个。

在一个事务提交时，它的最后一条end日志记录肯定已经持久化了，但是包含该事务修改的数据项的块（脏页）却不一定持久化，可以在以后的某个时间点输出，这个时间点可以是后台写进程，也可以是checkpoint进程。

3. 如果在系统崩溃后，我们可以严格地在日志的checkpoint点之后进行恢复，不用考虑checkpoint点之前的日志了，或者说，可以直接把checkpoint点之前的日志都删除了也没什么问题，因为在checkpoint记录之前的任何日志对应的数据库修改，都必然持久化了。在我们考虑的日志范围内，我们将整个日志对应的数据库操作都重新执行一遍，然后考虑undo-list中的事务，将它们的事务操作回滚。

{% asset_img image-20210918085628547.png %}



若在T3 2后面系统崩溃了，其实我们可以发现T2事务和T3事务的所有操作都要redo，然后T3事务的所有操作都要undo
这是因为T2事务提交了，而T3事务没有提交。

从上面的例子中，我们可以发现一些问题：

1. 在事务恢复的过程中，明明T2事务的数据库修改已经持久化了，但是最终系统崩溃后还要对T2事务的所有操作进行再次进行持久化，这样有两个问题：1.如果采用物理日志的形式，那么也就是性能出现问题了，2.但是如果是物理逻辑日志，那么出现错误了，因为T2的事务操作做了两遍，一遍在崩溃之前做的，一遍在这次恢复过程中做的，可能会在数据库中出现两条同样的记录。
2. 在checkpoint的时候，不允许整个系统进行写缓冲操作，这样会造成系统在checkpoint时完全没有可用性，这是不能忍受的。

PageLSN

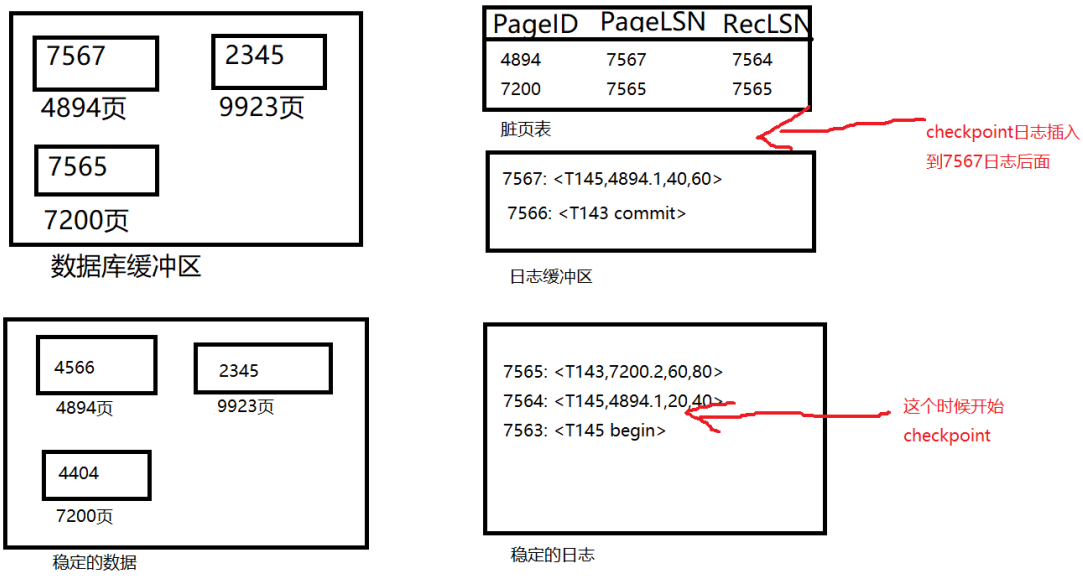
对于上述第一个问题，首先采用了PageLSN的解法，它的核心思想是在缓存页上增加一项用于表示最后一个在此页上的事务修改对应的日志号是什么。有了这个PageLSN，就可以知道了，“哦，在redo的时候，如果要redo L1日志记录，但是发现L1日志记录对应的页的PageLSN大于等于L1，那么就可以不用管L1，即使这个在checkpoint后持久化日志的”。

脏页表

为了解决上述第二个问题，我们设置了脏页表。因为在checkpoint时也会修改缓存页，但是这部分修改相关的日志和数据库修改可能在checkpoint日志之前写进了磁盘中，因此如果不做任何事情的话，可能会丢失这部分日志（毕竟，默认checkpoint日志之前的日志不会被考虑的）。 脏页表包含一个在数据库缓冲区中已更新的页的列表，它为每一页保存其PageLSN和一个称为RecLSN的字段，其中RecLSN表示第一个导致该Page变脏的LSN。

当正在执行checkpoint时，脏页表会被更新，里面记录了**在checkpoint期间及以前**做的缓冲区修改，包括哪个页被修改了，最近一次修改是哪个LSN，第一次使得页变脏的是哪个LSN。这个信息是有价值的，在checkpoint日志写入磁盘时，也会将这个脏页表作为checkpoint日志的一部分持久化了，在以后的分析阶段会重点用到脏页表。

{& asset_img image-20210927123850047.png %}



如上图所示，在7563日志已经持久化了的时候，我们开始checkpoint，但是系统并没有禁止更新缓冲区，所以，在checkpoint的过程中，7564和7567号日志修改了4894页，7565号日志修改了7200页，这些信息保存在脏页表中。然后checkpoint日志的日志号是7568。因此，如果没有脏页表，我们在恢复时，就直接从7568开始看了，而忽略了7564号及7568以前的日志，这就不对了。

在checkpoint日志罗盘时，顺便也把脏页表给罗盘了，这样我们就知道了：“哦，最早应该从日志记录的7564号开始考虑，而不是7568”。

恢复算法

分析阶段

一旦分析阶段发现一个更新日志记录（Update Log Record），而且对应的page不在脏页表中，分析阶段就将它添加进脏页表中，并设置对应的RecLSN项为该日志记录的LSN。（end应该指的是事务的commit和abort，在所有操作都持久化到数据库中后才会写end日志记录到磁盘中。）

{% asset_img image-20210927134837668.png %}

7571: <T146 commit>

7570: <T146,2390.4,50,90>

7569: <T146 begin>

7568: checkpoint

Txn	lastLSN
T145	7567

PageID	PageLSN	RecLSN
4894	7567	7564
7200	7565	7565

7567: <T145,4894.1,40,60>

7566: <T143 commit>

7565: <T143,7200.2,60>

7564: <T145,4894.1,20,40>

7563: <T145 begin>

7562: <T143,7200.2,60,80>