Lecture 14: 恢复系统

Database

Author: Forliage

Email: masterforliage@gmail.com

Date: June 9, 2025

College: 计算机科学与技术学院

Abstract

本讲笔记系统地介绍了关系型数据库的恢复系统机制，旨在在各种故障发生时保证事务的原子性、一致性与持久性。首先，通过“故障分类”一节分析了事务故障、系统崩溃与磁盘故障的不同类型及其对数据库状态的影响；随后，“基于日志的恢复”章节详细阐述了日志记录的格式与作用，比较了延迟修改与即时修改两种方案对写入顺序的要求，并引入检查点技术以加速恢复过程。接着，“并发事务恢复”部分扩展了单事务恢复方法，说明了在严格两阶段锁（2PL）并发控制下如何利用日志交叉排布实现多事务的回滚与重做。第四部分“缓冲区管理”讨论了日志缓冲与数据库缓冲的实现策略，明确了先写日志（WAL）规则与窃取 / 非强制策略对性能和恢复性的权衡；最后，“非易失性存储丢失故障”一节介绍了在线转储与模糊转储技术，以在硬件设备丢失时快速重建数据库状态。通过本讲内容，读者能够全面掌握日志驱动恢复、检查点与转储等核心技术，为设计高可靠数据库系统奠定坚实基础。

（该Abstract由ChatGPT-o4-mini-high生成）

Contents

1 故障分类

1.1 故障分类

事务故障：

- 逻辑错误：由于某些内部错误，事务无法完成。条件：溢出、输入错误、未找到数据等
- 系统错误：由于错误条件（例如死锁），数据库系统必须终止活动事务。错误条件（例如死锁）

系统崩溃：停电或其它硬件或软件故障导致系统崩溃。

故障停止假设：假定非易失性存储内容不会因系统崩溃而损坏（数据库系统有大量完整性检查，以防止磁盘数据损坏）

磁盘故障：磁头损坏或类似的磁盘故障会破坏全部或部分磁盘存储的：假定损坏是可检测的：磁盘驱动器使用校验和检测故障。

1.2 恢复算法

考虑事务T_i，它将\$50从账户A转移到账户B。两次更新：从账户A中减去50并向账户B中添加50.

事务T_i需要对A和B的更新输出到数据库。

- 在进行了其中一项修改单两项修改都未完成之前，可能会发生故障。
- 在未确保事务会提交的情况下修改数据库可能会使数据库处于不一致状态
- 如果在事务提交后立即发生故障，不修改数据库可能会导致更新丢失

恢复算法分为两部分：

1. 在正常事务处理期间采取的操作，以确保有足够的信息来从故障中恢复
2. 故障发生后采取的操作，将数据库内容恢复到确保原子性、一致性和持久性的状态

2 基于日志的恢复

2.1 基于日志的恢复

日志保存在为稳定存储的设备上“日志是一系列日志记录，用于记录数据库的更新活动。

当事务T_i开始时，它通过写入一个<T_i start>日志记录来注册本身。

在T_i执行写操作write(X)之前，写入一条日志记录$< T_i, X, V_1, V_2 >$其中V_1是写操作之前X的值（旧值），V_2是要写入X的值（新值）。

当T_i完成其最后一条语句时，会写入日志记录<T_i commit>

使用日志的两种方法：延迟数据库修改；立即数据库修改。

2.2 延迟数据库修改

延迟数据库修改方案将所有修改记录到日志中，但将所有写入操作推迟到部分提交后。一般不采用。

假设事务按顺序执行。事务通过向日志写入<T_i start>记录来启动。

写(X)操作会导致写入一条log 记录$< T_i, X, V >$，其中V是X的新值。注意：此方案不需要旧值。

此时不会在X上执行写入操作，而时将其推迟。当T_i部分提交时，<T_i commit>会被写入日志。最后，读取日志记录并用于实际执行之前推迟的写入操作。

2.3 即时数据库修改

即时修改方案允许在事务提交之前将未提交事务的更新写入缓冲区或磁盘本身，写的时间不受commit限制。

在写入数据项之前，必须先写入更新日志记录。我们假设日志记录直接输出到稳定存储。

在事务提交之前或之后的任何时间都可以将更新后的块输出到为你的难过存储。

块的输出顺序可能与编写顺序不同。

如何恢复？

恢复过程有两个操作而非一个：

- undo(T_i)从T_i的最后一条日志记录开始逆向操作，将T_i更新的所有数据项的值恢复为其旧值。
- redo(T_i)从T_i的第一条日志记录开始正向操作，将T_i更新的所有数据项的值设置为新值。

两个操作都必须是幂等的：也就是说，即使该操作执行多次，其效果与执行一次相同。恢复期间操作可能会重新执行，因此有此需求。

故障恢复时：

- 如果日志包含记录<T_i start>，但不包含记录<T_i commit>，则事务T_i需要回滚
- 如果日志中同时包含记录<T_i start>和记录<T_i commit>，则事务T_i需要重做。

先执行回滚操作，再执行重做操作。

2.4 检查点

重做/撤销日志中记录的所有事务可能非常缓慢：

1. 如果系统已经运行了很长时间，处理真个日志会很耗时
2. 我们可能会不必要地重做那些已经将更新输出到数据库的事务

通过定期执行检查点操作来简化恢复过程：

1. 将当前驻留再主内存中的所有日志记录输出到稳定存储设备上
2. 将所有修改过的缓冲区块输出到磁盘
3. 将日志记录<checkpoint L>写入稳定存储，其中L是检查点时刻所有活跃事务的列表：
   进行检查点操作时，所有更新都会停止

在恢复过程中，我们只需考虑在检查点之前开始的最近一次事务T_i以及在T_i之后开始的事务。

- 从日志末尾向后扫描，以找到最近的<checkpoint L>记录
- 只有处于L状态或在检查点之后开始的事务才需要重做或撤销
- 在检查点之前提交或中止事务，其所有更新已输出到稳定存储中。

日志的某些早期部分可能用于撤销操作：继续向后扫描，直到为L中的每个事务T_i找到一条记录<T_i start> 早于上述最早的<T_i start>记录的log部分不需要用于恢复，可随时擦除。

3 并发事务恢复

我们修改基于日志的恢复方案，以允许多个事务并发执行

- 所有事务共享一个磁盘缓冲区和一个日志文件
- 一个缓冲区可以包含由一个或多个事务更新的数据项

我们假设使用严格两阶段锁进行并发控制（2PL，X锁保持到事务结束）。即未提交事务的更新对其它事务不可见。

日志记录如前所述进行，不同事务的日志记录在日志中穿插排列。

检查点技术和恢复时采取的操作必须改变。因为在执行检查点时可能有多个事务处于活动状态。

检查点的执行方式与之前相同，只是现在检查点日志记录的形式为<checkpoint L>，其中L是检查点执行时活跃事务的列表。

我们假设在执行检查点时没有更新操作正在进行。

当系统从奔溃中恢复中：

1.它首先执行以下操作：

- 将撤销列表和重做列表初始化为空
- 从日志末尾开始反向扫描，当找到第一条<checkpoint L>记录时停止扫描。同时，反向扫描期间，对于日志中找到的每条记录：
  1. 如果记录是<T_i commit>，则将T_i添加到重做列表中
  2. 如果记录是<T_i start>，那么如果T_i不在重做列表中，则将T_i添加到撤销列表
  3. 如果记录是<T_i abort>，则将T_i添加到撤销列表
- 对于L中的每个T_i，如果T_i不在重做列表中，则将T_i添加到撤销列表。此时撤销列表包含必须撤销的未完成事务，重做列表包含必须重做的已完成事务。

2.现在恢复过程如下：

- 从日志文件末尾开始方向扫描日志，当为撤销列表中的每个T_i都遇到<T_i start>记录时停止。在扫描期间，对属于撤销列表中事务的每个日志记录执行撤销操作。
- 定位最近的<checkpoint L>记录
- 从<checkpoint L>记录开始向前扫描日志，直到日志末尾。在扫描期间，对属于重做列表中某个事务的每个日志记录执行重做操作。

4 缓冲区管理

4.1 日志记录缓冲

通常情况下，向稳定存储的输出以块为单位，并且通常日志记录比较小得多。

因此，日志记录被缓冲在主内存中，而不是直接输出到稳定存储。

当日志记录满足以下情况时，将其输出到稳定存储：

1. 缓冲区中的日志记录块已满
2. 或者执行了日志强制操作
3. 例如，检查点发生

强制日志通过将事务的所有日志记录（包括提交记录<T_i commit>）强制写入稳定存储来提交事务。

因此，可以使用一次输出操作输出多条日志记录，从而降低I/O成本。

如果对日志记录进行缓冲，则必须遵循以下4条规则：

1. 日志记录按照其创建顺序输出到稳定存储中
2. 事务T_i仅当日志记录<T_i commit>已输出到稳定存储时才进入提交状态
3. 在<T_i commit>可以输出到稳定存储之前，与T_i相关的所有日志记录都必须已输出到稳定存储。
4. 在将主内存中的一个数据块输出带数据库之前，与该数据块中的数据相关的所有日志记录都必须已输出到稳定存储（日志应先于数据写到磁盘）

此规则称为先写日志规则(write-ahead logging rule)或WAL。严格来说，先写日志规则仅要求输出撤销信息。

4.2 数据库缓冲

数据库维护数据块的内存缓冲区。

- 当需要一个新的数据块时，如果缓冲区已满，则需要从缓冲区中移除一个现有的数据块
- 如果选择移除的数据块已被更新，则必须将其输出到磁盘

恢复算法支持非强制策略，即事务提交时，更新的数据块不必写入磁盘。

强制策略：要求在提交时写入更新的数据块：提交成本更高。

恢复算法支持窃取策略，即包含未提交事务更新的块可以在事务提交之前写入磁盘。

如果将包含未提交更新的块输出到磁盘，首先会将包含这些更新的撤销信息的日志记录输出到稳定存储的日志中。

当一个块被输出到磁盘时，该块不应有正在进行的更新。可按如下方式确保这一点：

- 在写入数据项之前，事务会获取包含该数据项的块的X锁
- 写入完成后即可释放锁（这种短时间持有的锁称为latches）

将一个数据块输出到磁盘：

- 首先获取该数据块上的X锁latches（确保该数据块上没有正在进行的update操作）
- 然后执行日志刷新
- 然后将数据块输出到磁盘
- 最后松开模块上的latches

数据库缓冲区可以通过以下两种方式实现：

- 在为数据库预留的实际主存区域中
- 或者在虚拟内存中

在预留主存中实现缓冲区存在缺点：

- 内存预先在数据库缓冲区和应用程序之间进行了划分，限制了灵活性
- 需求可能会发生变化，尽管操作系统最清楚在任何时候应该如何划分内存，但它无法改变内存的划分方式

尽管存在一些缺点，但数据库缓冲区通常在虚拟内存中实现：

- 当操作系统需要换出一个已修改的页面时，该页面会被写入磁盘的交换空间
- 当数据库决定将缓冲区页面写入磁盘时，缓冲区页面可能在交换空间中，可能需要从磁盘的交换空间读取并输出到磁盘上的数据库，从而导致额外的I/O
- 理想情况下，当操作系统需要从缓冲区患处一个页面时，它应该将控制权交给数据库，而数据库又应该：
  1. 如果页面被修改，将其输出到数据库而非交换空间（确保先输出日志记录），如果它被修改
  2. 从缓冲区释放该页面，供操作系统使用。这样可以避免双重分页，但常见的操作系统不支持这种功能。

5 非易失性存储丢失故障

到目前为止，我们假设非易失性存储没有丢失

采用类似于监测点机制的机制的技术来处理非易失性存储丢失问题：

- 定期将数据库的全部内容转存储到稳定存储中
- 在转储过程中，不允许又事务处于活动状态；必须执行类似于检查点机制的操作：
  - 将当前驻留在主内存中的所有日志记录输出到稳定存储中
  - 将所有缓冲区块输出到磁盘
  - 将数据库内容复制到稳定存储设备
  - 将记录<dump>输出到稳定存储设备的日志中

从磁盘故障中恢复：

- 从最近的转储中恢复数据库
- 查阅日志并重新执行转储后提交的所有事务

可以扩展以允许事务转储期间处于活动状态；称为模糊转储或在线转储

各种商业数据库管理系统中使用了许多安全和可靠性方法

商业数据库产品中的各种工具可用于复值进行复值和恢复操作。