**应对数据库故障的对策**

Created by：蔡观洋

Time：S-2014.01.20 E-2014.01.27 M-

Type：读书笔记+总结

**目录**

[1. 故障模式 2](#_Toc378600964)

[2. 应对“错误数据输入” 2](#_Toc378600965)

[3. 应对“介质故障” 2](#_Toc378600966)

[3.1 介绍 2](#_Toc378600967)

[3.2 对策 2](#_Toc378600968)

[3.2.1 RAID 2](#_Toc378600969)

[4. 应对“灾难性故障” 4](#_Toc378600970)

[5. 应对“系统故障” 4](#_Toc378600971)

[5.1 预备知识(事务) 4](#_Toc378600972)

[5.2 undo日志 5](#_Toc378600973)

[5.2.1 操作原语 5](#_Toc378600974)

[5.2.2 公共日志记录形式 5](#_Toc378600975)

[5.2.3 undo日志记录内容 6](#_Toc378600976)

[5.2.4 undo日志规则 6](#_Toc378600977)

[5.3 redo日志 7](#_Toc378600978)

[5.3.1 概述 7](#_Toc378600979)

[5.3.2使用redo日志恢复 7](#_Toc378600980)

[5.4 undo/redo日志 8](#_Toc378600981)

[5.4.1 概述 8](#_Toc378600982)

[5.4.2 使用undo/redo日志恢复 8](#_Toc378600983)

[6. 参看文献： 9](#_Toc378600984)

1. 故障模式
   1. 错误数据输入：应用层的约束，为了适应此情况，DBMS提供了约束和触发器机制，由应用人员根据业务需求编写对应的规则来检测数据的合法性，如电话号码的合法性；
   2. 介质故障：存储数据的磁盘发生故障，可以采用RAID，备份等技术来避免；
   3. 灾难性故障：数据中心失火等意外，采用分布式的备份，冗余等来避免；
   4. 系统故障：导致事务状态丢失的问题，典型的系统故障包括掉电和软件错误，解决的方法是采用日志记录对所有数据库的更新，必要时加以恢复；

（支持可恢复性的基础技术是日志）

1. 应对“错误数据输入”

DBMS提供的有一些约束机制，如不同类型的字节长度，默认值，可用值等信息，应用开发人员可以根据业务需要使用，来减少用户录入错误的数据信息。当然也可以直接从业务端编程来检测并限制用户输入错误数据，这超出了DBMS的作用范围..

1. 应对“介质故障”

3.1 介绍

介质故障指外存储设备发生了故障，主要有磁盘损坏，磁头碰撞盘面，突然的强磁场干扰，数据传输部件出错，磁盘控制器出错等。这类故障通常称为硬故障，比系统故障发生的可能性小的多，但是破坏性很大，直接会导致数据库本身不能使用。

3.2 对策

1. 备份数据库：

将数据库在磁带或光盘这样的存储介质中做一个拷贝，可以通过全部或增量式的周期性维护这样的拷贝；

2. 联机保存数据库的冗余拷贝，这些拷贝最好分布在多个节点上，以应对数据库所在节点的突发事件；

3. RAID：

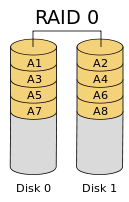
独立磁盘冗余阵列**[1]**（RAID， Redundant Array of Independent Disks），基本思想就是将多个相对便宜的硬盘组合起来，成为一个硬盘阵列组，使性能达到甚至超过一个价格昂贵，容量巨大的硬盘；

3.2.1 RAID

RAID把多个硬盘组合成一个逻辑扇区，因此操作系统只会把它当做一个硬盘，常被用在服务器计算机上；

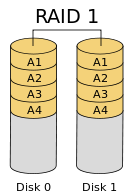
1. **RAID 0：**

将多个磁盘合并成一个大磁盘，不具有冗余，并行I/O，速度最快。



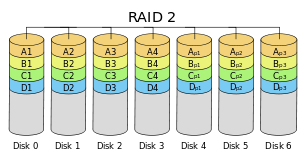
1. **RAID 1：**

两组以上的N个磁盘互作镜像，读取速度有很大提高，但是由于要做镜像，导致写入速度有稍微的降低。



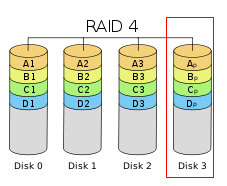
**3. RAID 2：**

是RAID 0的改良版，以**海明码[2]**的方式将数据进行编码后分区为独立的比特，并将数据写入硬盘中。



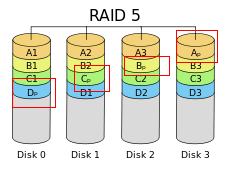
**4. RAID 4：**

采用一个冗余盘，存储数据盘的奇偶校验值，读的时候一般没必要读冗余盘，但是写的时候需要更改冗余盘对应位的值，以便它能保持为所有数据盘相应块的奇偶校验。但是冗余盘使用过多，导致对磁盘的损耗也会提高不少。

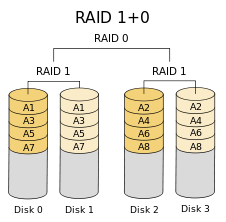


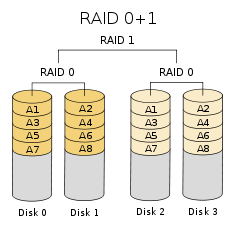
**5. RAID 5:**

它至少需要3个硬盘，会把数据和相对应的奇偶校验信息存储到组成RAID5的各个磁盘上。当一个磁盘发生损坏后，可以利用剩下的数据和响应的奇偶校验信息去恢复被损坏的数据。此方式比较常用。由于多了奇偶校验信息，导致写数据非常缓慢。



**6. RAID 10/01**





1. 应对“灾难性故障”

对于灾难性故障，像数据中心发生了火灾，遇到了地震等强大自然灾害，RAID和奇偶校验盘不会带来任何帮助，我们可以通过备份，异地冗余和分布式拷贝来防止此类故障。

1. 应对“系统故障”

当系统发生故障时，我们应通过某种机制来保障数据的完整性，使数据库能恢复到某一个正确的状态，继续支撑应用的正常使用。而解决由系统错误所引起问题的基本方法是在分离的，非易失性的日志中记录所有数据库的更新，必要时加以恢复。接下来介绍一下三种不同类型的日志：undo, redo, undo/redo日志。

5.1 预备知识(事务)

事务是数据库执行的基本单位，数据库的状态与这些事务的状态息息相关，当执行的所有事务状态都是正确的，那么我们也可以认为此时数据库的状态是一致的，正确的。

注意：我们不能混淆事务状态和数据库状态，事务状态是事务局部变量的取值，而不是数据库元素的取值；

1. 事务正确性原则的假设：

事务正确性：如果事务在没有其他任何事务和系统错误的情况下执行，并且在它开始执行时数据库处于一致的状态，那么当事务结束时数据库仍然处于一致的状态。

2）事务的基本特征（ACID）：

原子性（atomicity）：一个事务是一个不可分割的工作单位，包括的操作要不全做，要不都不做。如果仅有部分执行，可能导致数据库状态不一致；

一致性（consistency）：事务必须使数据库从一个一致状态变到另一个一致性状态；

隔离性（isolation）：一个事务的执行不能被其他事务干扰；

持久性（durability）：一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。（这个特性也跟数据库实现有关，可能为了减少调用磁盘IO的次数，会使用缓存技术，延迟刷新到磁盘）

3）日志管理和事务管理结构图：

事务管理器

查询处理器

缓冲管理器

恢复管理器

数据和

日志

日志管理器

5.2 undo日志

5.2.1 操作原语

1. INPUT(X)：将包含数据库元素X的磁盘块拷贝到主存缓冲区；

2.OUTPUT(X)：将包含X的缓冲区中的块拷贝回磁盘；

3.READ(X, t)：将数据库元素X拷贝到事务的局部变量t；

4.WRITE(X, t)：将局部变量t的值拷贝到主存缓冲区中的数据库元素X；

5.2.2 公共日志记录形式

当事务执行时，由日志管理器负责在日志中记录每个重要的事件；同时由“缓冲管理器”负责在主存中分配空间来存储日志，在合适的时机，将日志快写到非易失性存储介质中。

<START T>：表示事务T已开始；

<COMMIT T>：事务T已成功完成，并对数据库元素不再会有修改；

<ABORT T>：事务T不能成功完成；

5.2.3 undo日志记录内容

<T, X, v>：用三元组来记录更新记录，表示事务T改变了数据库元素X，而X原来的值时v；

注意：1）日志记录是对写入内存的WRITE动作做出的反应，而不是对写入磁盘做出的响应；

2）undo日志不记录数据库元素的新值，而只记录旧值。

5.2.4 undo日志规则

采用undo日志时需要事务和缓冲管理器遵守以下的约定：

1. 如果事务T改变了数据库元素X，那么形如<T, X, v>的日志记录必须在X的新值写入磁盘之前写到磁盘中；
2. 如果事务提交，那么<COMMIT T>的日志记录必须在事务改变的所有数据库元素先写到磁盘后再写到磁盘中，但应尽快

另外由于一些后台活动，可能对日志和缓冲区造成影响，若实际DBMS都是同事处理多个事务的，因此它们的日志通常是交叉共存的，导致日志进入磁盘的时机比我们预期的要早一些。但是最本质的方针是我们必须等到T的OUTPUT动作都完成后才写<COMMIT T>记录到磁盘中。当两个事务同时操作一个磁盘块时，由于一个事务大刷新操作，导致另一个事务的信息也刷新到磁盘中，但日志并未刷新到磁盘，违反规则1，这时我们需要在事务上采用一些加锁机制。

**5.2.5 使用undo日志恢复**

由于可能同时有多个事务在执行，导致日志也是交叉记录的，如可能有多个事务同时修改了某个值X，因此恢复的过程也应该是有顺序的，我们可以从尾部开始扫描日志。

恢复管理器首先应能够分辨出已提交的事务和未提交的事务，当扫描过程中发现<Commit T>时，根据规则2可知，事务T已经提交，因此我们可以不用处理此事务相关的记录信息。在恢复过程中，恢复处理器要能够记住所有有<Commit T>或者<Abort T>的事务。步骤如下：

1. 如果T的<Commit T>或<Abort T>已经被检查到，则对此记录不做任何处理；
2. 否则事务T未提交或未完成，我们要恢复T改变的磁盘值信息；

当这些都处理完后，恢复管理器必须为以前未中止且未完成的每个事务T写入一个日志记录<Abort T>，刷新日志。

根据undo日志的特征，我们可知恢复步骤是“幂等”的，他们重复执行多次与执行一次的效果一样。

**5.2.6 检查点**

针对上节介绍的日志恢复方式，它需要扫描整个日志文档，如果DBMS运行了很长一段时间，会导致日志文件会很大，这时扫描整个文档，显然是比较费时的。由于一旦事务提交，那些undo日志对今后的恢复工作也没用任何帮助，因此我们可以设计一个检查点，定期执行，恢复时恢复管理器只需要管理最新检查点中那些未提交的事务即可。

1. 静止检测点：

做检查点时我们需要做到：

1. 停止接收新事务；
2. 等到所有活跃事务提交或者中止，并且在日志中写入COMMIT或ABORT记录；
3. 将日志刷新到磁盘；
4. 写入日志记录<CKPT>，并再次刷新日志；
5. 重新开始接收事务。

问题：在做检查点的过程中，系统处于假死状态，不能再为外界服务

1. 非静止检查点：

它的主要特征是在做检查点的过程中，仍然可以接收新事务。

主要的处理是在开始做检查点的时候写日志<START CKPT t1,t2…tN>，指定那些当前活跃的事务，当活跃事务都执行完时，写入<END CKPT>。在这两个日志记录之间也会现事务tN+1 tN+2 等日志，因此恢复过程也稍微复杂一些。也是从尾部开始扫描日志

1. 当首先遇到的是<END CKPT>，我们可知未完成的事务肯定在前面的一个<START CKPT..>与日志结尾之间，之前的日志就可以抛弃了。
2. 当首先遇到的是<START CKPT..>，则未提交的事务在此标记内所有事务中的第一个开始执行标记<START tN>但未提交到日志结尾。

5.3 redo日志

5.3.1 概述

undo日志存在的弊端是在提交事务之前，必须保证此事务修改的所有数据都以成功写回磁盘中，过多的磁盘IO会影响DBMS的吞吐量。如果有一中机制可以保障修改先暂存在主存中，当发生故障中可以恢复数据库到一个一致状态，就可以认为是安全的。

日志记录<T, X, v>：事务T为数据库元素X写入新值v

Redo日志规则：

在修改磁盘上的任何数据元素X以前，要保证与X的这一修改相关的多有日志记录，包括更新记录<T, X, v>和<COMMIT T>记录，都必须出现在磁盘上。

5.3.2使用redo日志恢复

由于“修改日志”先于“数据修改”到磁盘，因此恢复时我们只需要重做那些已提交的事务即可，那些未提交的事务，肯定对数据的修改也为更新到磁盘中。

恢复的过程中我们从首部开始扫描日志记录。

确定已提交的事务

从首部开始扫描日志

扫描日志，<T, X, v>

T是否提交

否

为X写入值v

对每个未完成的事务，写<Abort T>日志，并刷新

是

日志是否结束

否

是

redo日志也可以结合使用非静止检查点技术。开始做检查点后，要首先将那些已提交事务，已经写到缓冲区中但还未写到磁盘中的数据元素写到磁盘中，然后写日志<START CKPT t1,t2…tN>到磁盘，记录哪些正活跃的事务，当所有活跃事务都完成后，写日志<END CKPT>。

5.4 undo/redo日志

5.4.1 概述

由于undo日志，redo日志在检查点过程中如何处理缓冲区方面都存在矛盾，如同一个缓冲区中可能包含了已提交事务的修改和未提交事务的修改，这样当我们刷新缓存时可能造成磁盘保存的数据并非出于一致性状态。

新的方式记录<T, X, v, w>：事务T改变了数据库元素X的值，改变之前为v， 改变之后为w。

日志规则：

在由于某个事务T所做改变而修改磁盘上的数据元素X前，更新记录<T, X, v, w>必须出现在磁盘上。

也即在刷新缓存之前，要先刷新日志信息。

通过此日志允许一个未提交事务的全部或部分修改保存到磁盘上，也会出现已提交事务的部分或全部不会出现在磁盘上。

5.4.2 使用undo/redo日志恢复

1. 按照从前往后做顺序，重做所有已提交的事务；

2. 按照从后往前做顺序，撤销所有未提交的事务；

由于磁盘保存事务数据存在不一致性，因此这两个步骤都需要做。

当此日志模型采集检查点技术时，对事务有个要求：事务在不确定其不会中止之前不能写入任何值（甚至连写到主存缓冲区也不允许）

1. 参看文献：

[1] RAID介绍： <http://zh.wikipedia.org/wiki/RAID>

[2] 海明码：<http://blog.csdn.net/lycb_gz/article/details/8214961>

<http://blog.csdn.net/lycb_gz/article/details/8232684>

[3] book.数据库系统实现：第二章—辅助存储管理；第六章—系统故障对策