



备份和恢复

单 位：重庆大学计算机学院

你是如何保存你的重要文件的？

- 重要的代码、文件如何保存？
 - 直接存在硬盘上，c盘？桌面？
 - 备份一份在另外一块硬盘盘符上？
 - 备份到U盘、移动硬盘？

主要学习目标

- 备份和恢复
- 远程备份系统



思考问题

- 计算机硬盘是否一定不会出现什么问题？
- 如果出现问题，一般有什么什么问题？
- 计算机系统呢？
- 你的文件如何更安全？

系统可能发生的故障有很多种，每种故障需要不同的处理方法。

一、事务故障

有两种错误可能造成事务执行失败：

1. **逻辑错误**：事务由于某些内部条件而无法继续正常执行，这样的内部条件如非法输入、找不到数据、溢出或超出资源限制。
2. **系统错误**：系统进入一种不良状态（如死锁），结果事务无法继续正常执行，但该事务可以在以后的某个时间重新执行。

二、系统故障

- 硬件故障，或者是数据库系统或操作系统的漏洞，导致易失性存储器内容的丢失，并使得事务处理停止，而非易失性存储器仍完好无损。
- 硬件错误和软件漏洞导致系统终止，而不破坏非易失性存储器内容的假设称为故障-停止假设。设计良好的系统在硬件和软件层有大量的内部检查，一旦错误发生就会将系统停止。因此，故障-停止假设是合理的。

三、磁盘故障

在数据传送操作过程中由于磁头损坏或故障造成磁盘块上的内容丢失。其他磁盘上的数据拷贝，或三级介质（如DVD或磁带）上的归档备份可用于从这种故障中恢复。

四、其他故障

天灾（地震或火灾等自然灾害），人祸（人为破坏机房或数据），间谍或黑客攻击，等

系统如何从故障中恢复

- 首先需要确定用于存储数据的设备的故障方式
- 其次，考虑这些故障方式对数据库内容的影响
- 最后提出在故障发生后仍保证数据库一致性以及事务原子性的恢复算法。

- 如果想要恢复到故障发生前的状态，就必须知道当时的状态。那么如何实现呢？
- 引入日志记录数据库中所有修改。

日志

为了保证在发生故障以后，数据库系统能恢复到和故障发生之前一致的状态，必须记录数据库中的所有修改。

- 日志是最常用的用于记录数据库修改的结构，它是日志记录的序列，记录了数据库中的所有更新活动。
- 为了从系统故障和磁盘故障中恢复时能使用日志记录，**日志必须存放在稳定存储器中。**

稳定存储器

- 存储器是计算和保存的基础
- 计算过程中数据仅临时使用
- 而最终结果数据将永远保存

- 易失性存储器 (volatile storage) ?
- 非易失性存储器 (nonvolatile storage) ?
- 稳定存储器 (stable storage) ?

- 稳定存储器通常是用非易失性存储介质来近似实现，在多个非易失性存储介质上以独立的故障模式复制所需信息，并且以受控的方式更新信息，以保证数据传送过程中发生的故障不会破坏所需信息。
- 稳定存储器或者近似稳定存储器在恢复算法中起着至关重要的作用。

常用日志记录格式：

更新日志记录： $\langle T_i, X_j, V_1, V_2 \rangle$ ，描述一次数据库写操作。包含事务标识、数据项标识、旧值和新值等字段。

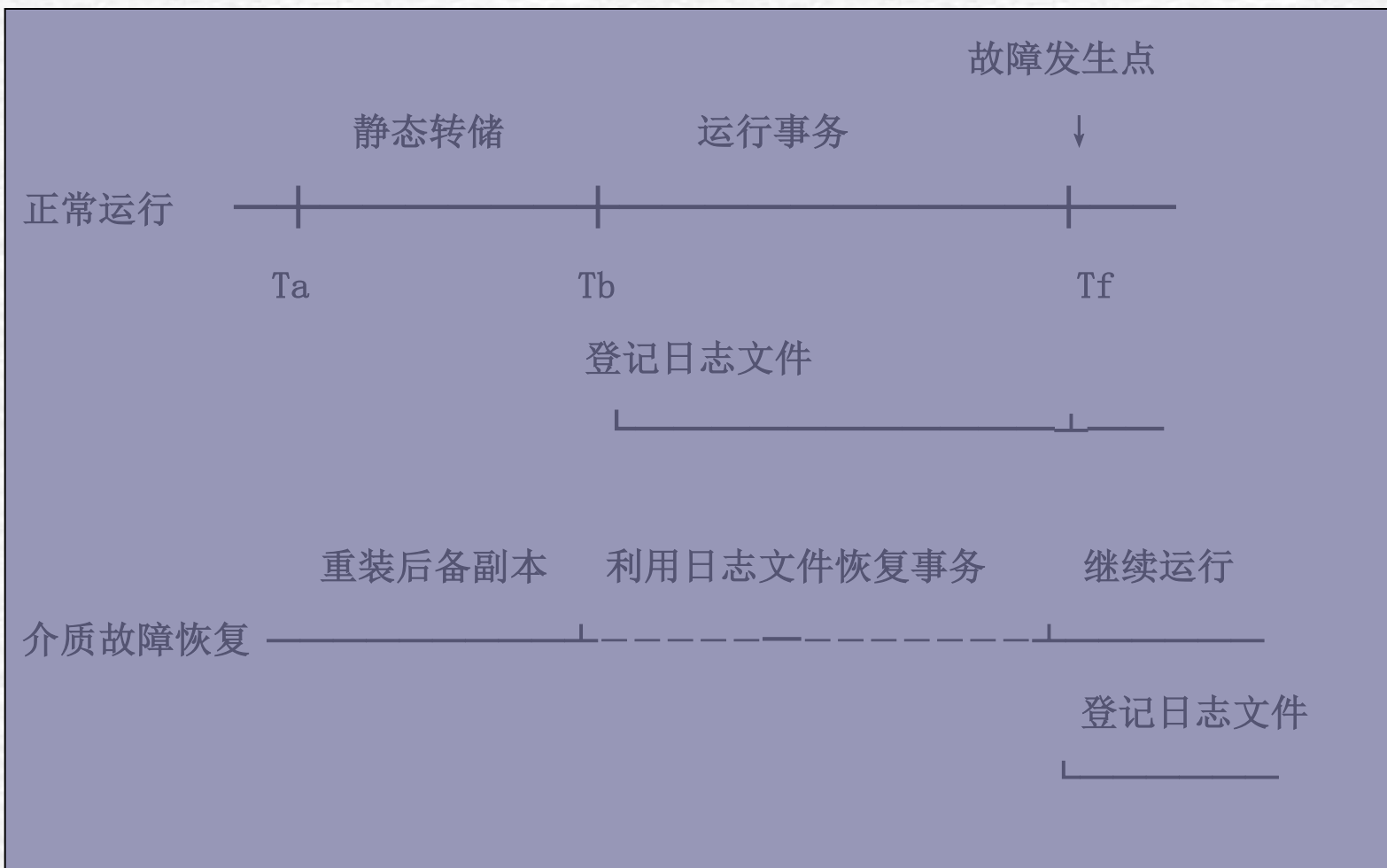
事务开始日志记录： $\langle T_i \text{ start} \rangle$ ，表明事务 T_i 开始执行。

事务提交日志记录： $\langle T_i \text{ commit} \rangle$ ，表明事务 T_i 提交。事务提交日志记录是一个事务最后的一个日志记录，当一个事务的提交日志记录输出到稳定存储器后，意味着该事务提交，也就是所有更早的日志记录都已经输出到稳定存储器中。因此，在日志中就有足够的信息来保证，即使系统崩溃，事务所做的更新也可以重做。

事务中止日志记录： $\langle T_i \text{ abort} \rangle$ ，表明事务 T_i 中止。

登记日志的原则

- 为保证数据库是可恢复的，登记日志文件时必须遵循两条原则
 - 登记的次序严格按并行事务执行的时间次序
 - 必须先写日志文件，后写数据库
 - 写日志文件操作：把表示这个修改的日志记录写到日志文件
 - 写数据库操作：把对数据的修改写到数据库中




- 在故障恢复机制中，采用日志记录数据库更新，日志记录在创建时都输出到稳定存储器。通常向稳定存储器的输出是以块为单位进行的，而大多数情况下，一个日志记录比一个块要小得多，每个日志记录的输出就转化成在物理上大得多的输出，因此这就增加了大量系统执行的开销。
- 那么如何减少与数据库交互的开销，提高效率呢？

日志缓冲区

- 先将日志记录输出到主存的日志缓冲区中，多条日志记录集中到日志缓冲区后，再用一次输出操作输出到稳定存储器。稳定存储器中的日志记录顺序必须与写入日志缓冲区的顺序完全一样。

➤由于使用了日志缓冲区，日志记录在输出到稳定存储器前可能有一段时间只存在于主存中。由于系统发生崩溃时这种日志记录会丢失，因此恢复技术要增加一些要求以保证事务的原子性：


1. 在日志记录<Ti commit> 输出到稳定存储器后，事务Ti进入提交状态。
2. 在日志记录<Ti commit> 输出到稳定存储器前，与事务Ti有关的所有日志记录必须已经输出到稳定存储器。
3. 在主存中的数据块输出到数据库前，所有与该数据块中数据有关的日志记录必须已经输出到稳定存储器。



➤这三条规则表明，在某些情况下某些日志记录必须已经输出到稳定存储器中。而提前输出日志记录不会造成任何问题。

➤因此，当系统发现需要将一个日志记录输出到稳定存储器时，如果主存中有足够的日志记录可以填满整个日志记录块，就将其整个输出。

➤如果没有足够的日志记录填入该块，那么就将主存中的所有日志记录填入一个部分填充的块，并输出到稳定存储器。



事务故障的恢复

恢复策略：

反向扫描日志文件，对该事务的更新操作执行**逆操作**（即将日志中更新前的数据写回到数据库中），直至事务的开始标志。

系统故障的恢复

恢复策略：撤销故障发生时未完成的事务，重做已完成的事务。

方法：扫描日志文件；找出故障发生前提交的事务，让该事务重做（REDO）；找出故障发生前未提交的事务，让其撤销（UNDO）。

系统故障的恢复步骤

1. 正向扫描日志文件（即从头扫描日志文件）
 - Redo队列：在故障发生前已经提交的事务 T_i
 - 日志包括 $\langle T_i \text{ start} \rangle$,
 - 以及 $\langle T_i \text{ commit} \rangle$ 或 $\langle T_i \text{ abort} \rangle$
 - Undo队列：故障发生时尚未完成的事务 T_i ,
 - 日志包括 $\langle T_i, \text{start} \rangle$,
 - 但不包括 $\langle T_i \text{ commit} \rangle$
 - 也不包括 $\langle T_i \text{ abort} \rangle$

系统故障的恢复步骤

2. 对Undo队列事务进行UNDO处理

反向扫描日志文件，对每个UNDO队列中 T_i 更新过的数据项恢复为旧值 $\langle T_i, X_j, V_1 \rangle$ ，直到遇到 $\langle T_i \text{ start} \rangle$ 记录，说明该事务撤销完成，写入一条 $\langle T_i \text{ abort} \rangle$ 记录

3. 对Redo队列事务进行REDO处理

正向扫描日志文件，对每个REDO队列中的 T_i 更新过的数据项设置为新值。

基于日志的恢复机制

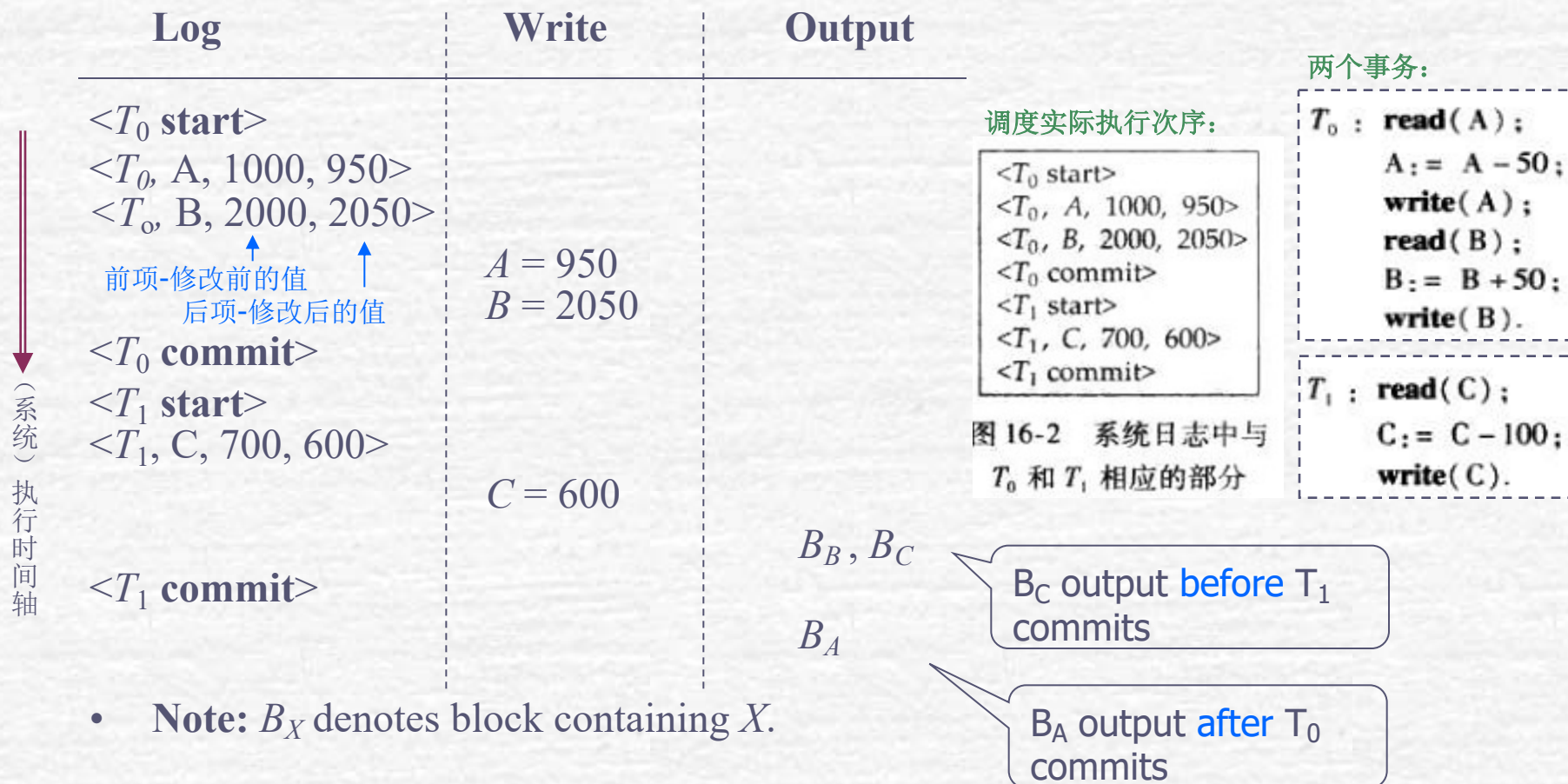
“日志-内存-数据库”三者的更新情况:

事务在修改数据库前创建日志

记录在稳定存储器中

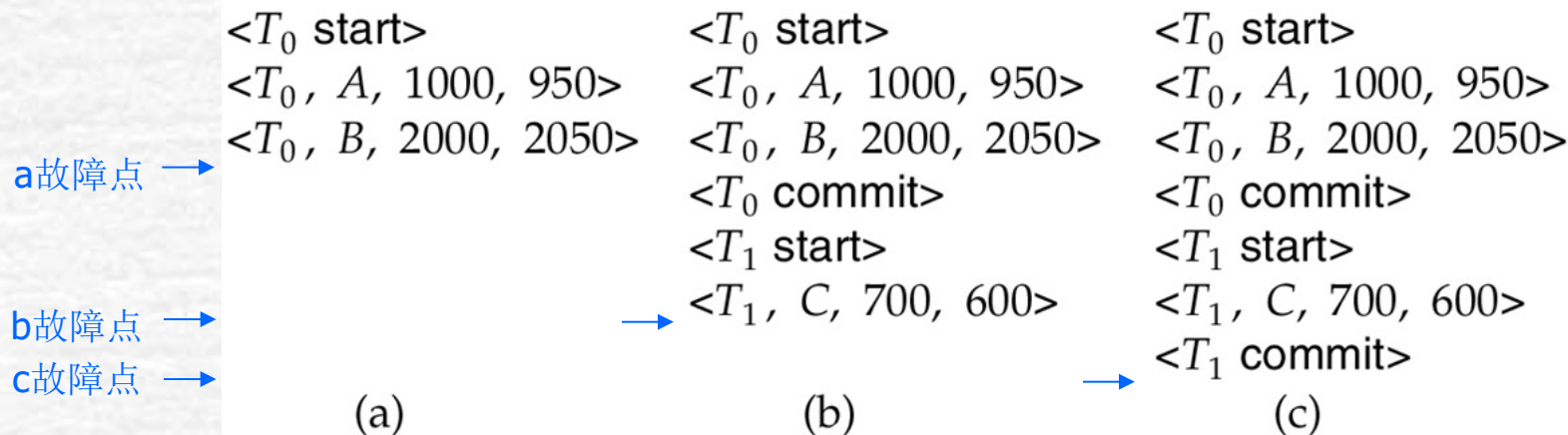
在内存中进行

对数据库的修改(记录在磁盘上)



基于日志的恢复机制

下面我们观察三种情况的日志。



账户A=1000
账户B=2000
账号C= 700

T_0 -账户A转
账50到B;

T_1 -账号C取
出100.

图 16-4 在三个不同时间显示的同一个日志

每一种情况的恢复行为如下: 三种不同故障情形, 分别应当如何恢复?

a故障 (a) undo (T_0): B 恢复为 2000 , A 恢复为 1000, 并且增加日志

$\langle T_0, B, 2000 \rangle, \langle T_0, A, 1000 \rangle, \langle T_0, \text{abort} \rangle$

均未提交!

b故障 (b) redo (T_0) 和 undo (T_1): A 和 B 设置为 950 和 2050 , C 恢复为 700,


增加日志记录 $\langle T_1, C, 700 \rangle, \langle T_1, \text{abort} \rangle$.

仅 T_0 提交!

c故障 (c) redo (T_0) 和 redo (T_1): A 和 B 设置为 950 和 2050.

C 设置为 600

均已提交!



➤ 当系统故障发生时，必须检查日志来决定哪些事务需要重做，哪些需要撤销。如果搜索整个日志来确定该信息，那么开销太大。

➤ 如何能够降低开销？

- **引入检查点**。在运行过程中，DBMS一般每隔一定的时间在日志记录中设置一个检查点，在日志中加入<Checkpoint L>记录。
- 在系统崩溃后，系统检查日志以找到最后一条检查点记录，只需要对L中的事务以及检查点记录写到日志中之后才开始执行的事务进行撤销或重做操作；
- 考虑在检查点前完成的事务Ti。<Ti commit>记录（或<Ti abort>记录）在日志中出现检查点记录之前，Ti所作的任何数据库修改都必然已在检查点之前或作为检查点本身的一部分写入数据库。因此，在恢复时就不必再对Ti执行重做操作。
- 因此，引入检查点可以提高恢复过程的效率。

检查点的执行过程

- 将当前位于主存的所有日志记录输出到稳定存储器。
- 将所有修改的缓冲块输出到磁盘。
- 将一个日志记录<Checkpoint L> 输出到稳定存储器，其中L是执行检查点时正活跃的事务的列表。
- 通常，在执行检查点操作的过程中不允许执行任何更新，且将所有更新过的缓冲块都输出到磁盘。

基于检查点的数据恢复过程

崩溃发生后当数据库系统重启时，恢复动作分两阶段进行：

1. 在重做阶段，系统通过从最后一个检查点开始正向地扫描日志来重放所有事务的更新。重放的扫描日志的过程中所采取的具体步骤如下：

- a. 将要回滚的事务的列表 undo-list 初始设定为 $\langle \text{checkpoint } L \rangle$ 日志记录中的 L 列表。
- b. 一旦遇到形为 $\langle T_i, X_j, V_1, V_2 \rangle$ 的正常日志记录或形为 $\langle T_i, X_j, V_2 \rangle$ 的 redo-only 日志记录，就重做这个操作；也就是说，将值 V_2 写给数据项 X_j 。
- c. 一旦发现形为 $\langle T_i, \text{start} \rangle$ 的日志记录，就把 T_i 加到 undo-list 中。
- d. 一旦发现形为 $\langle T_i, \text{abort} \rangle$ 或 $\langle T_i, \text{commit} \rangle$ 的日志记录，就把 T_i 从 undo-list 中去掉。

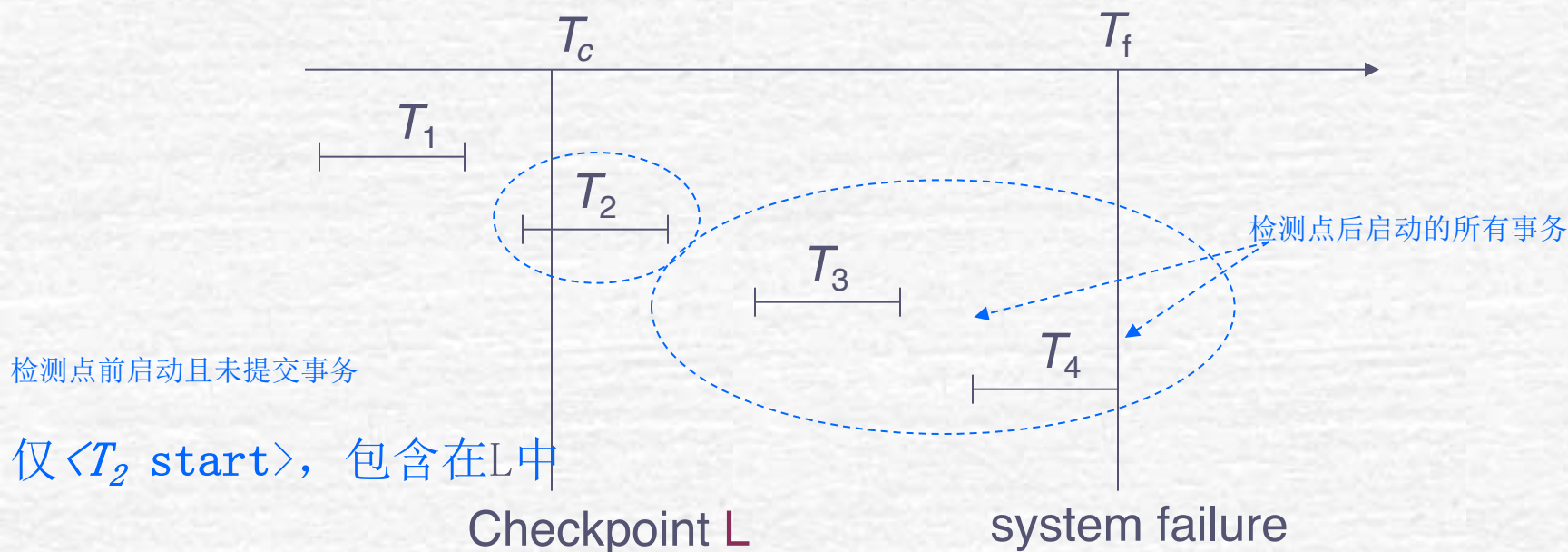
在 redo 阶段的末尾，undo-list 包括在系统崩溃之前尚未完成的所有事务，即，既没有提交也没有完成回滚的那些事务。

2. 在撤销阶段，系统回滚 undo-list 中的所有事务。它通过从尾端开始反向扫描日志来执行回滚。

- a. 一旦发现属于 undo-list 中的事务的日志记录，就执行 undo 操作，就像在一个失败事务的回滚过程中发现了该日志记录一样。
- b. 当系统发现 undo-list 中事务 T_i 的 $\langle T_i, \text{start} \rangle$ 日志记录，它就往日志中写一个 $\langle T_i, \text{abort} \rangle$ 日志记录，并且把 T_i 从 undo-list 中去掉。
- c. 一旦 undo-list 变为空表，即系统已经找到了开始时位于 undo-list 中的所有事务的 $\langle T_i, \text{start} \rangle$ 日志记录，则撤销阶段结束。

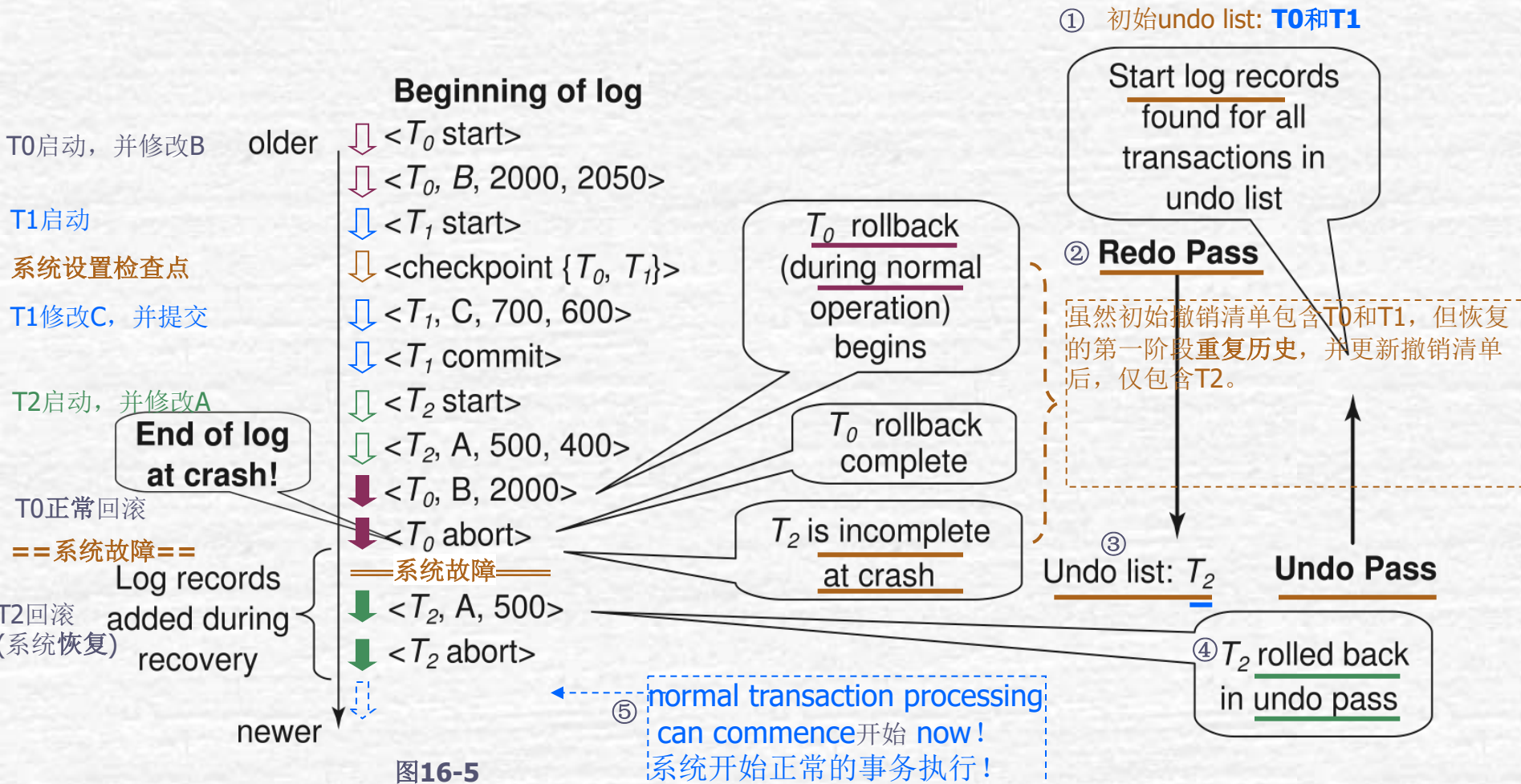
当恢复过程的撤销阶段结束之后，就可以重新开始正常的事务处理了。

基于检查点进行恢复的直观示例



- T_1 可以忽略 (因设置检查点时已完成对数据库的修改)
- T_2 and T_3 redo. (因已经完成提交操作)
- T_4 undo (因没有完成提交操作)

基于检查点进行恢复的详细过程示例



备 份

- 备份是数据库运行过程中必要的规划及操作，但还原通常只在发生问题时才会用到，或者想要找回旧数据时才执行还原。
- 总之，管理人员必定要有备份规划，并定期执行备份。

数据库备份

- 周期性地将磁盘上的数据库转储到磁带上进行备份。由于磁带是脱机存放，可以不受系统故障的影响。数据库的数据量一般比较大，备份耗时久，数据库状态一般须冻结，这样也很影响数据库的正常工作，因此一般**数据库备份不能太频繁**。当数据库发生故障时，就可以用最近的备份来恢复数据库。
- 数据库中的数据一般只部分更新，很少全部更新。**增量备份**就是指转储修改过的物理块，那么转储的数据量显著减少，耗时减少，因此备份频率可以提高，从而减少发生故障时数据更新的丢失。

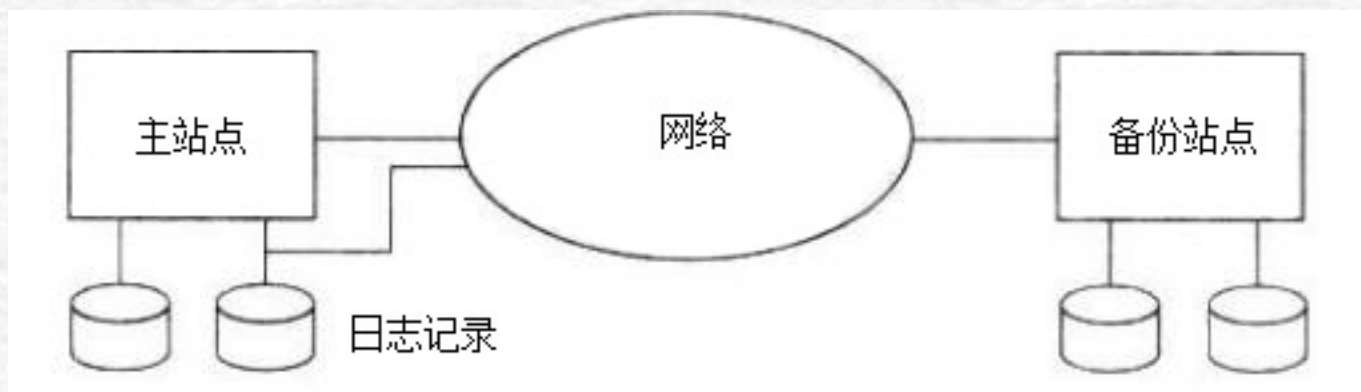
日志文件备份

- 日志文件是供恢复用的数据库运行情况的记录。日志文件备份本身不能用来还原数据库，因为日志文件只记录自上次备份后，对数据库所做的异常更新。
- 故日志文件是在数据库还原之后，用来将数据库还原到最初的备份点上。

远程备份

- 传统的事务处理系统是集中式或客户/服务器模式的系统，这样的系统易受自然灾害（如火灾、洪水和地震）的攻击，系统可通过远程备份提供高可用性。
- 在一个站点执行事务处理，称为主站点，使用一个远程备份站点，这里有主站点所有的数据备份。
- 远程备份站点有时也叫辅助站点。随着更新在主站点上执行，远程站点必须保持与主站点同步。我们通过发送主站点的所有日志记录到远程备份站点来达到同步。
- 远程备份站点必须物理的与主站点分离

- 当主站发生故障，远程备份站点就接管处理。
- 它首先使用源于主站点的数据拷贝，以及收到的来自主站点的日志记录执行恢复。
- 事实上，远程备份站点执行的恢复动作就是主站点要恢复时需要执行的恢复动作。
- 一旦恢复执行完成，远程备份站点就开始处理事务。





➤ 在设计一个远程备份系统时，有几个问题必须考虑。

➤ 故障检测。

➤ 控制权的移交。

➤ 恢复时间。

➤ 提交时间。



故障检测

- 对于远程备份系统而言，检测什么时候主站点发生故障是很重要的。
- 通信线路故障会使远程备份站点误以为主站点已发生故障。为避免这个问题，我们在主站点和备份站点之间维持几条具有独立故障模式的通信线路。

控制权的移交

- 当主站点发生故障时，备份站点就接管处理并成为新的主站点，当原来的主站点恢复后，它可以为远程备份站点工作，抑或再次接管作为主站点。
- 在任意情况下，原主站点都必须收到一份在它故障期间备份站点上所执行更新的日志。

恢复时间

- 如果远程备份站点上的日志增长到很大，恢复就会花很长时间。
- 远程备份站点可以周期性处理它收到的redo日志，并执行一个检查点，从而日期中早期的部分可以删除。这样，远程备份站点接管的延迟显著缩短。
- 采用热备份配置可使用备份站点几乎能在一瞬间接管。在该配置中，远程备份站点不断处理到达的redo日志记录，在本地进行更新。一旦检测到主站点发生故障，备份站点就通过回滚未完成的事务来完成恢复；然后做好处理新事务的准备。

提交时间

为保证已提交事务的更新是持久的，只有在其日志记录到达备份站点之后才能宣布该事务已提交。该延迟会导致等待事务提交的时间变长，因此某些系统允许较低程度的持久性。持久性的程度可以按如下分类：

- **一方保险：**事务的提交日志记录一写入主站点的稳定存储器，事务就提交。
 - 当备份站点接管处理时，已提交事务的更新可能还没有在备份站点执行，好像丢失了更新。当主站点恢复后，丢失的更新不能直接并入，因为它可能与后来在备份站点上执行的更新冲突。需人工干预恢复数据库一致状态。
- **两方强保险：**事务的提交日志记录一写入主站点和备份站点的稳定存储器，事务就提交。
 - 如果其中一个站点停工，事务处理就无法进行，因此，更新丢失可能性很小，但是可用性比单站点还低。
- **两方保险：**如果主站点和备份站点都是活跃的，该机制与两方强保险机制相同。如果只有主站点的活跃的，事务的提交日志记录一写入主站点的稳定存储器，就允许事务提交。
 - 可用性较好，避免了更新丢失，提交稍慢，综合较佳方案。

恢复策略

➤针对不同故障类型，可采用不同的恢复策略。

1. 事务故障恢复策略

事务故障是经常发生的，其一定发生在事务提交之前。事务一旦提交，即使要撤销也不可能了。对于事务故障，一般采取的恢复策略就是撤销该事务，并释放其占有的资源。

2. 系统崩溃恢复策略

系统崩溃不像事务故障这么频繁，但发生的可能性还是很大。对于系统崩溃，一般采取的恢复策略是重新启动操作系统和DBMS，恢复数据库到一致状态（对未提交的事务进行撤销操作，对已提交的事务进行重做操作）。只有当数据库恢复到一致状态，才允许用户访问数据库。

3. 磁盘故障恢复策略

在正常情况下，磁盘故障是很少发生的。对于磁盘故障，一般采用的恢复策略是修复系统，必要时更换磁盘，加载最近备份复本，再根据日志文件中的日志记录重做最近备份复本以后提交的所有事务。

数据库恢复小结

- 如果数据库只包含成功事务提交的结果，就说数据库处于一致性状态。保证数据一致性是对数据库的最基本的要求。
- 事务是数据库的逻辑工作单位
 - **DBMS**保证系统中一切事务的原子性、一致性、隔离性和持续性

- DBMS必须对事务故障、系统故障和介质故障进行恢复
- 恢复中最经常使用的技术：数据库转储和登记日志文件
- 恢复的基本原理：利用存储在后备副本、日志文件和数据库镜像中的冗余数据来重建数据库

- 常用恢复技术
 - 事务故障的恢复
 - UNDO
 - 系统故障的恢复
 - UNDO + REDO
 - 介质故障的恢复
 - 重装备份并恢复到一致性状态 + REDO

openGauss数据库的日志

- 在数据库运行过程中，会出现大量日志，既有保证数据库安全可靠的WAL日志（预写式日志，也称为Xlog），也有用于数据库日常维护的运行和操作日志等。在数据库发生故障时，可以参考这些日志进行问题定位和数据库恢复的操作。

检查项	异常状态
系统日志	数据库系统进程运行时产生的日志，记录系统进程的异常信息。
操作日志	通过客户端工具（例如gs_guc）操作数据库时产生的日志。
Trace日志	打开数据库的调试开关后，会记录大量的Trace日志。这些日志可以用来分析数据库的异常信息。
黑匣子日志	数据库系统崩溃的时候，通过故障现场堆、栈信息可以分析出故障发生时的进程上下文，方便故障定位。黑匣子具有在系统崩溃时，dump出进程和线程的堆、栈、寄存器信息的功能。

openGauss数据库的日志

检查项	异常状态
审计日志	开启数据库审计功能后，将数据库用户的某些操作记录在日志中，这些日志称为审计日志。
WAL日志	又称为REDO日志，在数据库异常损坏时，可以利用WAL日志进行恢复。由于WAL日志的重要性，所以需要经常备份这些日志。
性能日志	数据库系统在运行时检测物理资源的运行状态的日志，在对外部资源进行访问时的性能检测，包括磁盘、OBS、HadoopopenGauss等外部资源的访问检测信息。



课堂小测试

$\langle T_0, \text{start} \rangle$

$\langle T_0, A, 1000, 800 \rangle$

$\langle T_1, \text{start} \rangle$

$\langle T_0, B, 2000, 1600 \rangle$

$\langle T_1, C, 700, 600 \rangle$

$\langle T_0, \text{commit} \rangle$

如何进行恢复？

课程总结与作业安排

- 基本知识：
 - 故障的种类
 - 日志
 - redo、undo
 - 远程备份系统
- 扩展学习：
 - 学习openGauss的日志系统
- 作业

第16章习题： 16. 2, 16. 4.