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问题？
- 如果出现问题，一般有什么问题？
- 计算机系统呢？
- 你的文件如何更安全？

故障的种类

1）故障是如何产生的?

1）软硬件故障：

- **事务故障**(transaction failure)。有两种错误可能造成事务执行失败：
  - □ **逻辑错误**(logical error)。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。
  - □ **系统错误**(system error)。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。
- **系统崩溃**(system crash)。硬件故障，或者是数据库软件或操作系统的漏洞，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致使系统终止，而不破坏非易失性存储器内容的假设称为**故障－停止假设**(fail-stop assumption)。设计良好的系统在硬件和软件层有大量的内部检查，一旦有错误发生就会将系统停止。因此，故障－停止假设是合理的。
- **磁盘故障**(disk failure)。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。

2）其它事故：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…

稳定存储器

2)如何实现稳定存储器？

- 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

存储器分类

- **易失性存储器**(volatile storage)？ 内存
- **非易失性存储器**(nonvolatile storage)？硬盘，磁带
- **稳定存储器**(stable storage)？

稳定存储器，或更准确地说是接近稳定的存储器，在恢复算法中起到至关重要的作用。

要实现稳定存储器，我们需要在多个非易失性存储介质(通常是磁盘)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。

前面(第 10 章)讲到 RAID 系统保证了单个磁盘的故障(即使发生在数据传送过程中)不会导致数据丢失。最简单并且最快的 RAID 形式是磁盘镜像，即在不同的磁盘上为每个磁盘块保存两个拷贝。RAID 的其他形式代价低一些，但性能也差一些。

具有容错能力的磁盘阵列

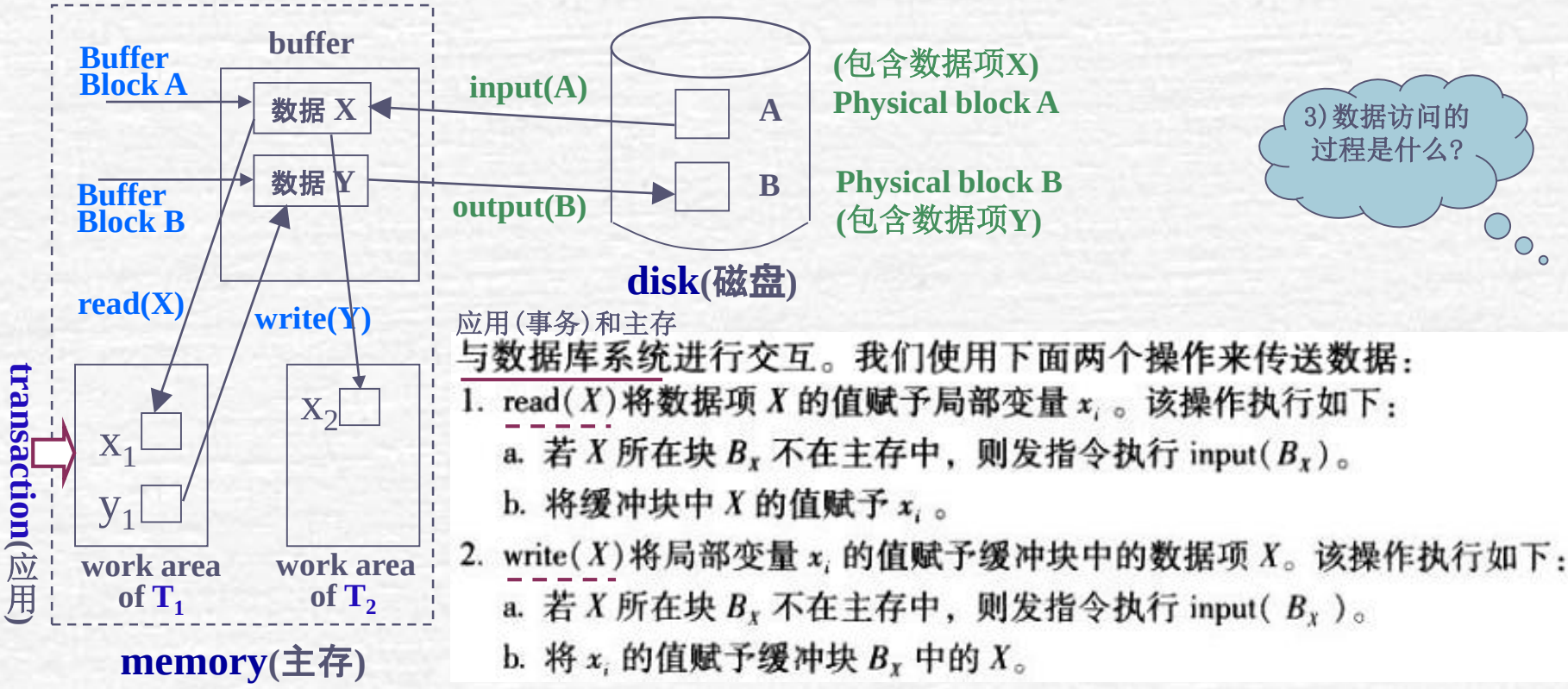
但是，RAID 系统不能防止由于灾难(如火灾或洪水)而导致的数据丢失。许多系统通过将归档备份存储在磁带上并转移到其他地方来防止这种灾难。但是，由于磁带不能被连续不断地移至其他地方，最后一次磁带被移至其他地方以后所做的更新可能会在这样的灾难中丢失。更安全的系统远程为稳定存储器的每一个块保存一份拷贝，除在本地磁盘系统进行块存储外，还通过计算机网络写到远程去。由于在往本地存储器输出块的同时也要输出到远程系统，一旦输出操作完成，即使发生火灾或洪水这样的灾难，输出结果也不会丢失。我们在 16.9 节学习这种远程备份系统。

数据访问

事务由磁盘向主存输入信息，然后再将信息输出回磁盘。输入和输出操作以块为单位完成。位于磁盘上的块称为**物理块**(physical block)，临时位于主存的块称为**缓冲块**(buffer block)。内存中用于临时存放块的区域称为**磁盘缓冲区**(disk buffer)。

磁盘和主存间的块移动是由下面两个操作引发的：

1. input(B)传送物理块 B 至主存。
2. output(B)传送缓冲块 B 至磁盘，并替换磁盘上相应的物理块。

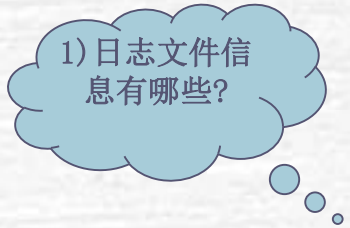


应用(事务)和主存与数据库系统进行交互。我们使用下面两个操作来传送数据：

1. read(X)将数据项 X 的值赋予局部变量 x_i。该操作执行如下：
   a. 若 X 所在块 B_X 不在主存中，则发指令执行 input(B_X)。
   b. 将缓冲块中 X 的值赋予 x_i。
2. write(X)将局部变量 x_i 的值赋予缓冲块中的数据项 X。该操作执行如下：
   a. 若 X 所在块 B_X 不在主存中，则发指令执行 input(B_X)。
   b. 将 x_i 的值赋予缓冲块 B_X 中的 X。

基于日志的恢复机制

<T_0 **start**>
<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>



<T_0 **commit**>
<T_1 **start**>
<T_1, C, 700, 600>

<T_1 **commit**>

Deferred database modification
延迟数据库修改(技术)
Immediate database modification
立即数据库修改(技术)

基于日志的恢复机制

事务执行的日志是什么？

两个事务：

T_0 : **read**(A);
A := A − 50;
write(A);
read(B);
B := B + 50;
write(B).

T_1 : **read**(C);
C := C − 100;
write(C).

T_0	T_1
Read(A);	
A:=A-50;	
Write(A);	
	Read(C);
	C:=C-100;
	Write(C);
Read(B);	
B:=B+50;	
Write(B).	

基于日志的恢复机制

2)事务执行和日志的关系是什么？

“日志-内存-数据库”三者的更新情况：

记录在稳定存储器中 Log	在内存中进行 Write	对数据库的修改(记录在磁盘上) Output
$<T_0$ **start**$>$		
$<T_0$, A, 1000, 950$>$		
$<T_0$, B, 2000, 2050$>$		
	$A = 950$	
	$B = 2050$	
$<T_0$ **commit**$>$		
$<T_1$ **start**$>$		
$<T_1$, C, 700, 600$>$		
	$C = 600$	
		B_B, B_C
$<T_1$ **commit**$>$		
		B_A

前项-修改前的值

后项-修改后的值

（系统）执行时间轴

B_C output before T_1 commits

B_A output after T_0 commits

- **Note:** B_X denotes block containing X.

调度实际执行次序：

```
<T0 start>
<T0, A, 1000, 950>
<T0, B, 2000, 2050>
<T0 commit>
<T1 start>
<T1, C, 700, 600>
<T1 commit>
```

图 16-2 系统日志中与 T_0 和 T_1 相应的部分

两个事务：

```
T0 : read(A);
     A := A - 50;
     write(A);
     read(B);
     B := B + 50;
     write(B).
```

```
T1 : read(C);
     C := C - 100;
     write(C).
```

案例3

基于日志的恢复机制

3)基于日志的恢复策略是什么？

下面我们观察三种情况的日志.

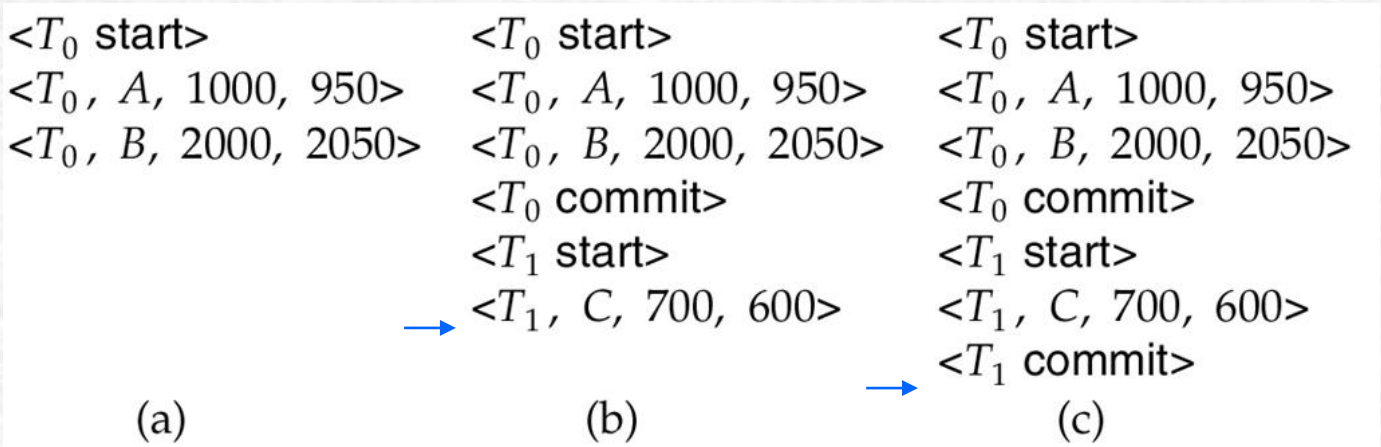


图 16-4　在三个不同时间显示的同一个日志

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.

案例4

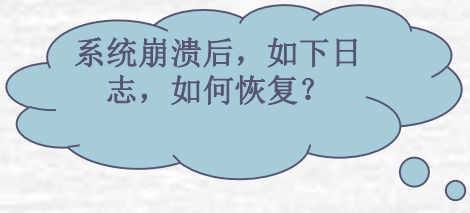
每一种情况的恢复行为如下: **三种不同故障情形，分别应当如何恢复？**

a故障 (a) undo (T_0): B 恢复为 2000， A 恢复为 1000, 并且增加日志 <T_0, B, 2000>, <T_0, A, 1000>, <T_0, **abort**>　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 <T_1, C, 700>, <T_1, **abort**>.　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！

基于日志的恢复机制



<T0 start>
<T0, A, 800, 700>
<T0, B, 1000, 1100>
<T1 start>
<T1, C, 800, 600>
<T0, B, 1000>
<T1, D, 600, 800>
<T0, A, 800>
<T0, abort>

检查点

4)什么是检查点，主要作用?

不足

当系统故障发生时，我们必须检查日志，决定哪些事务需要重做，哪些需要撤销。原则上，我们需要搜索整个日志来确定该信息。这样做有两个主要的困难：

1. 搜索过程太耗时。
2. 根据我们的算法，大多数需要重做的事务已把其更新写入数据库中。尽管对它们重做不会造成不良后果，但会使恢复过程变得更长。

为降低这种开销，引入检查点。

检查点

检查点的执行过程如下：

4)如何利用检查点，进行数据恢复?

1. 将当前位于主存的所有日志记录输出到稳定存储器。
2. 将所有修改的缓冲块输出到磁盘。
3. 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。

在检查点执行过程中，不允许事务执行任何更新动作，如往缓冲块中写入或写日志记录。16.5.2

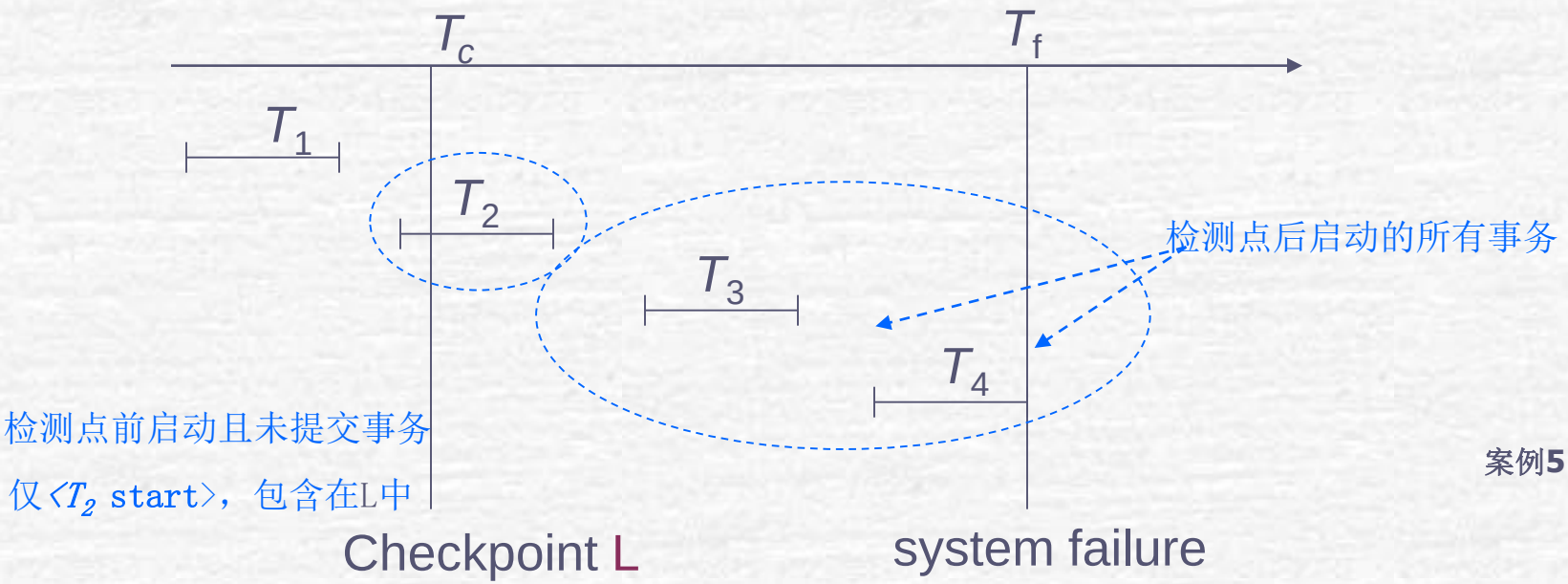
数据恢复

在系统崩溃发生之后，系统检查日志以找到最后一条 <**checkpoint** L> 记录（这可以通过从尾端开始反向搜索日志来进行，直至遇到第一条 <**checkpoint** L> 记录）。

只需要对 L 中的事务，以及 <**checkpoint** L> 记录写到日志中之后才开始执行的事务进行 undo 或 redo 操作。让我们把这个事务集合记为 T。

- 对 T 中所有事务 T_k，若日志中既没有 <T_k **commit**> 记录，也没有 <T_k **abort**> 记录，则执行 undo(T_k)。
- 对 T 中所有事务 T_k，若日志中有 <T_k **commit**> 记录或 <T_k **abort**> 记录，则执行 redo(T_k)。

基于检查点进行恢复



案例5

- T1 可以 忽略 (更新操作已经写入硬盘 due to checkpoint)
- T2 和 T3 redone. (因设置检查点时已完成对数据库的修改)
- T4 undone (因已完成提交动作)

(未已完成提交动作)

基于检查点进行恢复的详细过程示例

5）基于日志的数据恢复过程（有检查点）是什么？

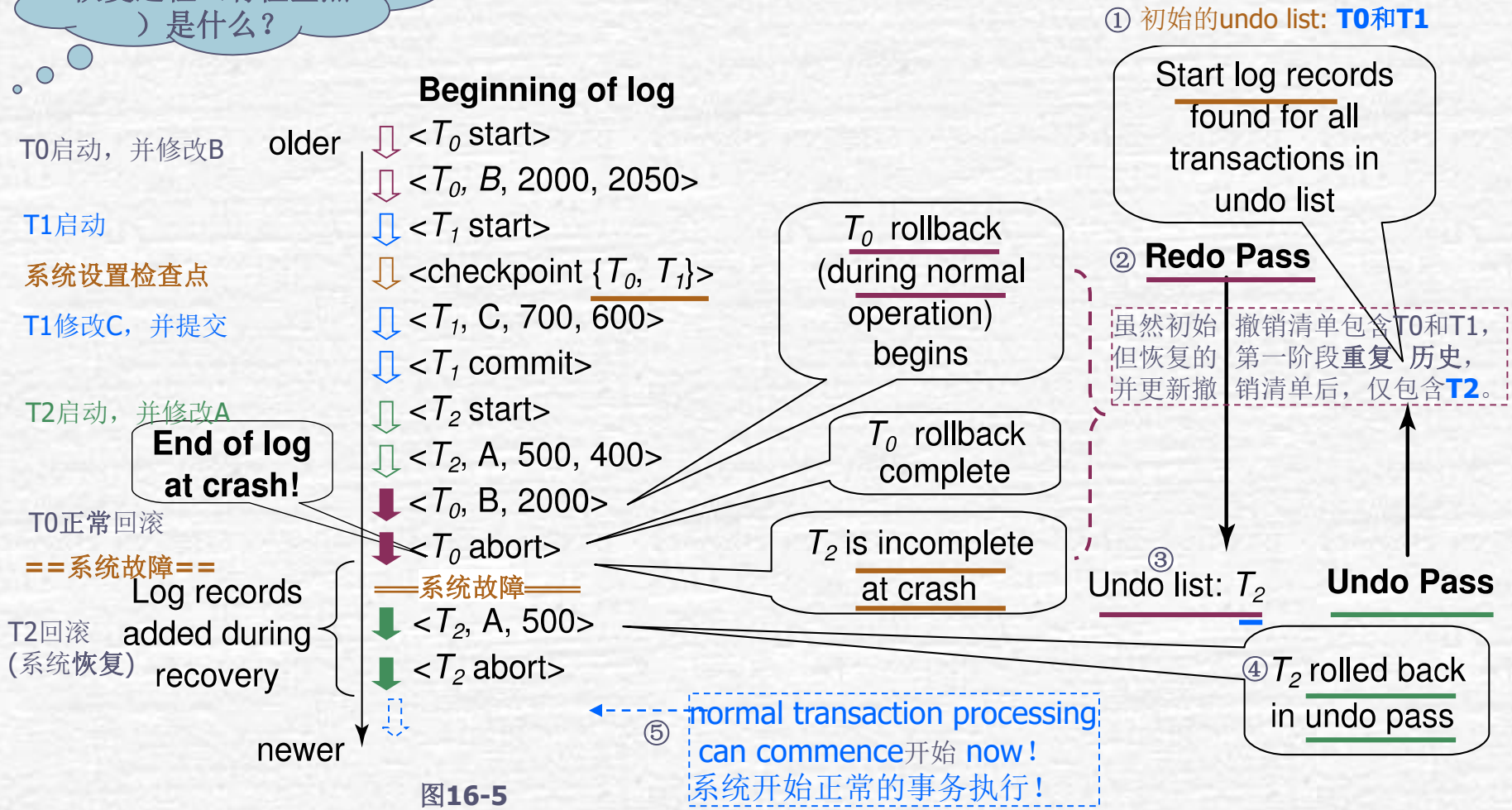


图16-5

案例6

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 <**checkpoint** L> 日志记录中的 L 列表。

   b. 一旦遇到形为 <T_i，X_j，V_1，V_2> 的正常日志记录或形为 <T_i，X_j，V_2> 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 <T_i，**start**> 的日志记录，就把 T_i 加到 undo-list 中。

   d. 一旦发现形为 <T_i，**abort**> 或 <T_i，**commit**> 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。 → 需要恢复的事务集合

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 <T_i，**start**> 日志记录，它就往日志中写一个 <T_i，**abort**> 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 <T_i，**start**> 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。

远程备份系统

1. 基本工作原理

1)远程备份系统的工作原理是什么？

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难(如火灾、洪水和地震)的攻击。要求事务处理系统无论系统故障还是自然灾难都能运行的需求日益高涨。这种系统必须提供**高可用性**(high availability)；即系统不能使用的时间必须非常地短。

我们可以这样来获得高可用性，在一个站点执行事务处理，称为**主站点**(primary site)，使用一个**远程备份**(remote backup)站点，这里有主站点所有数据的备份。远程备份站点有时也叫**辅助站点**(secondary site)，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。

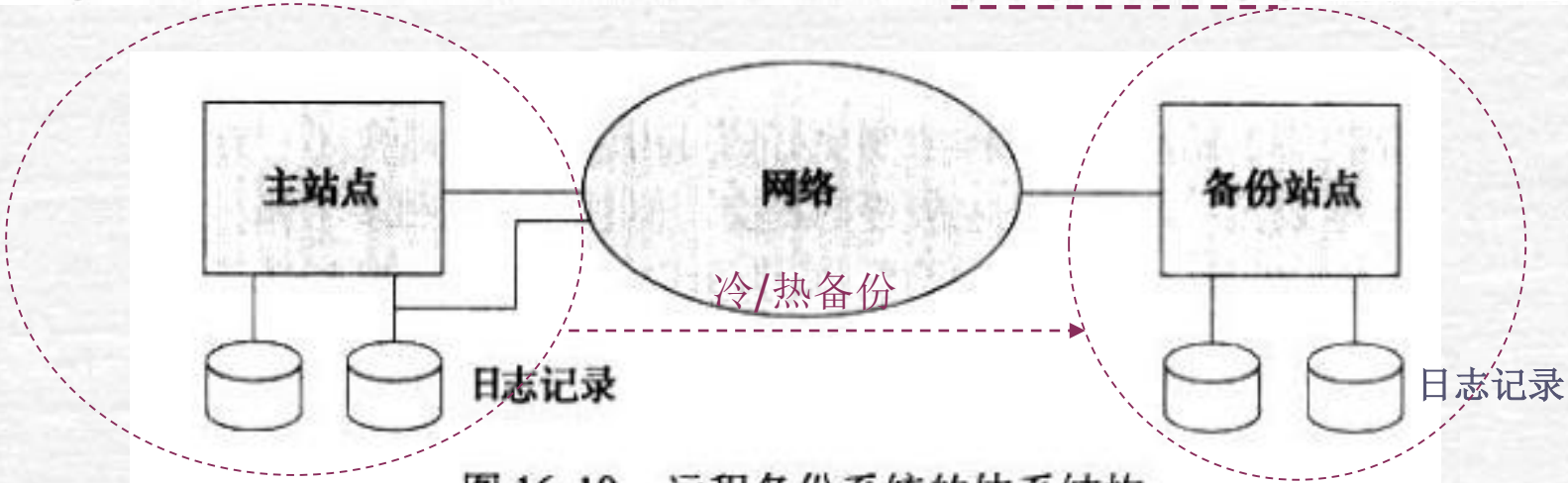


图 16-10　远程备份系统的体系结构

2)远程备份系统设计需要考虑的问题？

远程备份系统的关键技术

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**(detection of failure)。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，例如，可以使用几种互相独立的网络连接，或许包括通过电话线路的调制解调器连接。这些连接可能由那些能通过电话系统进行通信的操作人员的手工干预来提供支持。
- **控制权的移交**(transfer of control)。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，它可以作为远程备份站点工作，抑或再次接管并作为主站点。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。

  移交控制权最简单的办法是原主站点从原备份站点收到 redo 日志，并将它们应用到本地以赶上更新。然后原主站点就可以作为远程备份站点工作，如果控制权必须回传，原备份站点可以假装发生故障，导致原主站点重新接管。
- **恢复时间**(time to recovery)。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地处理它收到的 redo 日志，并执行一个检查点，从而日志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**(hot-spare)配置可使备份站点几乎能在一瞬间接管，在该配置中，远程备份站点不断地处理到达的 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；然后就做好处理新事务的准备。

远程备份系统的关键技术（续）

- **提交时间**(time to commit)。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称该事务已提交。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**(one-safe)。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。

    这种机制的问题是，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，这样，该更新好像丢失了。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相冲突。因此，可能需要人工干预来使数据库回到一致状态。
  - □ **两方强保险**(two-very-safe)。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。

    这种机制的问题是，如果主站点或备份站点其中的一个停工，事务处理就无法进行。因此，虽然丢失数据的可能性很小，但其可用性实际上比单站点的情况还低。
  - □ **两方保险**(two-safe)。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。如果只有主站点是活跃的，事务的提交日志记录一写入主站点的稳定存储器事务，就允许它提交。

    这一机制提供了比两方强保险机制更好的可用性，同时避免了一方保险机制面临的事务丢失问题。它导致比一方保险机制较慢的提交，但总的来说利多于弊。

(应只是多少情况下不需人工干预，但并非所有情况)

指前面的好像丢失？

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

<T0, start>
<T0, A, 1000, 800>
<T1, start>
<T0, B, 2000, 1600>
<T1, C, 700, 600>
<T0, commit>

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.