2024年春季学期

数据库系统概论

An Introduction to Database Systems

第十章 数据库恢复技术

中国科学技术大学 大数据学院

黄振亚，huangzhy@ustc.edu.cn

复习：数据由DBMS统一管理和控制

- **DBMS提供的数据控制功能**
  - **(1)数据的安全性（Security）保护 （第4章）**

    保护数据，以防止不合法的使用造成的数据的泄密和破坏。
  - **(2)数据的完整性（Integrity）检查 （第5章）**

    将数据控制在有效的范围内，或保证数据之间满足一定的关系。
  - **(3)数据库恢复（Recovery） （第10章）**

    将数据库从错误状态恢复到某一已知的正确状态。
  - **(4)并发（Concurrency）控制 （第11章）**

    对多用户的并发操作加以控制和协调，防止相互干扰而得到错误的结果。

第十章 数据库恢复技术

10.1 事务的基本概念

一、事务定义

二、事务的特性

一、事务(Transaction)

- 定义
  - 事务(Transaction)是用户定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的工作单位
- 事务和程序比较
  - 在关系数据库中，一个事务可以是一条或多条SQL语句, 也可以包含一个或多个程序
  - 一个程序通常包含多个事务
- 事务是恢复和并发控制的基本单位

定义事务

- 显式定义方式

```
BEGIN TRANSACTION
    SQL 语句1
    SQL 语句2
    。 。 。 。 。
COMMIT
```

```
BEGIN TRANSACTION
    SQL 语句1
    SQL 语句2
    。 。 。 。 。
ROLLBACK
```

- 事务异常终止
- 事务运行的过程中发
- 系统将事务中对数据
  部撤销
- 事务滚回到开始时的

●事务正常结束
●提交事务的所有操作（读+更新）
●事务中所有对数据库的更新写回到磁盘上的物理数据库中

定义事务

- 显式定义方式

BEGIN TRANSACTION	BEGIN TRANSACTION
SQL 语句1	SQL 语句1
SQL 语句2	SQL 语句2
。 。 。 。 。	。 。 。 。 。
COMMIT	ROLLBACK

- 隐式方式

当用户没有显式地定义事务时，

DBMS按缺省规定自动划分事务

二、事务的特性(ACID特性)

事务的ACID特性

- 原子性（Atomicity）
  - 事务中的操作要么都做，要么都不做
- 一致性（Consistency）
  - 从一个一致性状态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
- 持续性（Durability ）
  - 一个事务一旦提交，它的结果不应当受其他操作或故障的影响

（1）原子性

- 事务是数据库的逻辑工作单位
  - 事务中包括的诸操作要么都做，要么都不做

（2）一致性

- 事务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态
- 一致性状态
  - 数据库中只包含成功事务提交的结果
- 不一致状态
  - 数据库系统运行中发生故障，有些事务尚未完成就被迫中断；
  - 这些未完成事务对数据库所做的修改有一部分已写入物理数据库，这时数据库就处于一种不正确的状态

一致性与原子性

银行转帐例子：从帐号A中取出1万元，存入帐号B。

- 定义一个事务，该事务包括两个操作

A	B
A=A-1w	B=B+1w

- 这两个操作要么全做，要么全不做
  - 全做或者全不做，数据库都处于一致性状态
  - 如果只做一个操作，用户逻辑上就会发生错误，少了一万元，数据库就处于不一致性状态

（3）隔离性

一个事务的执行不能被其他事务干扰

- 一个事务内部的操作及使用的数据对其他并发事务是隔离的
- 并发执行的各个事务之间不能互相干扰

假设数据D为16，A,B两个用户操作数据D，存在冲突

	A	B
1	读取D	
2		读取D
3	D=D-1, 写入	
4		D=D+1, 写入

（4）持续性

- 持续性也称永久性（Permanence）
  - 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。
  - 接下来的其他操作或故障不应该对其执行结果有任何影响。

事务的特性

- 保证事务ACID特性是事务处理的任务
- 破坏事务ACID特性的因素

1. 多个事务并行运行时，不同事务的操作交叉执行

   - 数据库管理系统必须保证多个事务的交叉运行不影响这些事务的隔离性

2. 事务在运行过程中被强行停止

   - 数据库管理系统必须保证被强行终止的事务对数据库和其他事务没有任何影响

第十章 数据库恢复技术

10.2 数据库恢复概述

- 故障是不可避免的
  - 系统故障：计算机软、硬件故障
  - 人为故障：操作员的失误、恶意的破坏等。
- 数据库的恢复
  - 把数据库从错误状态恢复到某一已知的正确状态(称为一致状态或完整状态)
- 恢复子系统是数据库管理系统的一个重要组成部分
- 恢复技术是衡量系统优劣的重要指标

第十章 数据库恢复技术

故障的种类

- 事务内部的故障
- 系统故障
- 介质故障
- 计算机病毒

一、事务内部的故障

- 事务内部的故障
  - 有的是可以通过事务程序本身发现的
    - 银行转账事务
  - 有的是非预期的

事务内部的故障（续）

- [例] 银行转账事务，这个事务把一笔金额从一个账户甲转给另一个账户乙

```
BEGIN TRANSACTION
    读账户甲的余额BALANCE;
BALANCE=BALANCE-AMOUNT;                (AMOUNT 为转账金额)
 写回BALANCE;
IF(BALANCE < 0 ) THEN
    { 打印‘金额不足，不能转账’;          (事务内部造成回滚，可预期)
      ROLLBACK;                        (撤销刚才的修改，恢复事务)
    }
 ELSE
    { 读账户乙的余额BALANCE1;
      BALANCE1=BALANCE1+AMOUNT;
      写回BALANCE1;
      COMMIT;
    }
```

事务内部的故障（续）

- 例子所包括的两个更新操作要么全部完成，要么全部不做

  否则就会使数据库处于不一致状态，

  - 例如，只把账户甲的余额减少了而没有把账户乙的余额增加
- 在这段程序中
  - 若产生账户甲余额不足的情况，应用程序可以发现并让事务滚回，撤销已作的修改，恢复数据库到正确状态。

事务内部的故障（续）

- 事务内部更多的故障是非预期的，是不能由应用程序处理的
  - 运算溢出
  - 并发事务发生死锁而被选中撤销该事务
  - 违反了某些完整性限制等

  以后，事务故障仅指这类非预期的故障
- 事务故障：指事务没有达到预期的终点 (COMMIT或者显式的ROLLBACK)，数据库可能处于不正确状态
  - 破坏原子性：一部分执行了
- 事务故障的恢复：事务撤消（UNDO）
  - 强行回滚（ROLLBACK）该事务
  - 撤销该事务已经作出的任何对数据库的修改，使得该事务象根本没有启动一样

二、系统故障

- 系统故障：软故障
  - 指造成系统停止运转的任何事件，使得系统要重新启动
    - 整个系统的正常运行突然被破坏
    - 所有正在运行的事务都非正常终止
    - 内存中数据库缓冲区的信息全部丢失
    - 不破坏数据库
- 系统故障的常见原因
  - 特定类型的硬件错误（如CPU故障）
  - 操作系统故障
  - DBMS代码错误
  - 系统断电

系统故障的恢复

- 发生系统故障时，事务未提交，一些尚未完成的事务的结果可能已送入物理数据库，造成数据库可能处于不正确状态
  - 恢复策略：强行撤消（UNDO）所有未完成事务
- 发生系统故障时，事务已提交，有些已完成的事务可能有一部分甚至全部留在缓冲区（在内存），尚未写回到磁盘上的物理数据库中，系统故障使得这些事务对数据库的修改部分或全部丢失
  - 恢复策略：重做（REDO）所有已提交的事务

三、介质故障

- 介质故障：硬故障，指外存故障
  - 磁盘损坏
  - 磁头碰撞
  - 操作系统的某种潜在错误
  - 瞬时强磁场干扰
- 介质故障恢复： 备份
  - 装入数据库发生介质故障前某个时刻的数据副本
  - 重做自此时开始的所有成功事务，将这些事务已提交的结果重新记入数据库
  - 比较前两种故障（事务内部的故障、系统故障），可能性小，但是破坏性大

四、计算机病毒

- 计算机病毒
  - 一种人为的故障或破坏，是一些恶作剧者研制的一种计算机程序
  - 可以繁殖和传播
- 危害
  - 破坏、盗窃系统中的数据
  - 破坏系统文件
- 破坏计算机系统 —— 破坏数据库系统

故障小结

- 各类故障，对数据库的影响有两种可能性
  - 一是数据库本身被破坏（硬盘）
  - 二是数据库没有被破坏，但数据可能不正确（不一致），这是由于事务的运行被非正常终止造成的。

第十章 数据库恢复技术

10.4 恢复的实现技术

- 恢复操作的基本原理：冗余

  利用存储在系统其它地方的冗余数据来重建数据库中已被破坏，或不正确的那部分数据

- 恢复机制涉及的关键问题
  - 1. 如何建立冗余数据
    - 数据转储（backup）
    - 登录日志文件（logging）
  - 2. 如何利用这些冗余数据实施数据库恢复

10.4.1 数据转储

一、什么是数据转储

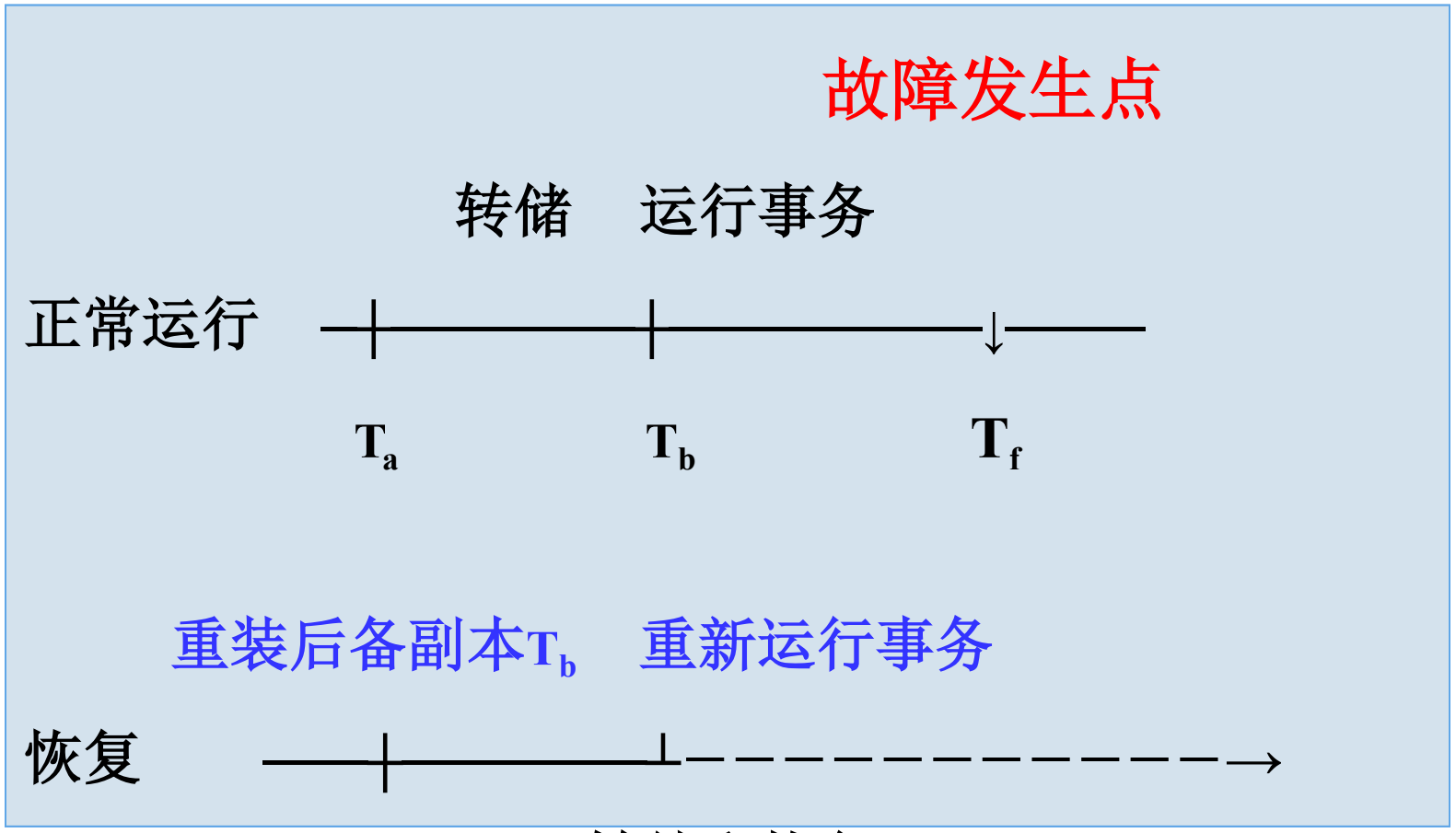
二、转储方法

一、什么是数据转储

- 转储：指DBA将整个数据库复制到磁带或另一个磁盘上保存起来的过程，备用的数据文本称为后备副本
- 如何使用
  - 数据库遭到破坏后可以将后备副本重新装入
  - 重装后备副本只能将数据库恢复到转储时的状态
  - 要想恢复到故障发生时的状态，必须重新运行自转储以后的所有更新事务
- 转储非常耗时，不能频繁进行

数据转储（续）

[例]



转储和恢复

数据转储（续）

上图中：

- 系统在T_a时刻停止运行事务，进行数据库转储
- 在T_b时刻转储完毕，得到T_b时刻的数据库一致性副本
- 系统运行到T_f时刻发生故障
- 为恢复数据库，首先由数据库管理员重装数据库后备副本，将数据库恢复至T_b时刻的状态
- 重新运行自T_b～T_f时刻的所有更新事务，把数据库恢复到故障发生前的一致状态

二、转储方法

1. 静态转储与动态转储
2. 海量转储与增量转储
3. 转储方法小结

静态转储

- 在系统中无运行事务时进行的转储操作
- 转储开始时数据库处于一致性状态
- 转储期间不允许对数据库的任何存取、修改活动
- 得到的一定是一个数据一致性的副本
- 优点：实现简单
- 缺点：降低了数据库的可用性
  - 转储必须等待正运行的用户事务结束
  - 新的事务必须等转储结束

动态转储

- 转储操作与用户事务并发进行
- 转储期间允许对数据库进行存取或修改
- 优点
  - 不用等待正在运行的用户事务结束
  - 不会影响新事务的运行
- 动态转储的缺点
  - 不能保证副本中的数据正确有效

[例]在转储期间的某个时刻Tc，系统把数据A=100转储到磁带上，而在下一时刻Td，某一事务将A改为200。转储结束后，后备副本上的A已是过时的数据了

动态转储

- 利用动态转储得到的副本进行故障恢复
  - 需要把动态转储期间各事务对数据库的修改活动登记下来，建立日志文件
  - 后备副本加上日志文件才能把数据库恢复到某一时刻的正确状态

2. 海量转储与增量转储

- 海量转储: 每次转储全部数据库
- 增量转储: 只转储上次转储后更新过的数据
- 海量转储与增量转储比较
  - 从恢复角度看，使用海量转储得到的后备副本进行恢复往往更方便
  - 但如果数据库很大，事务处理又十分频繁，则增量转储方式更实用更有效

3．转储方法小结

- 转储方法分类

		转储状态	
		动态转储	静态转储
转储方式	海量转储	动态海量转储	静态海量转储
	增量转储	动态增量转储	静态增量转储

10.4 恢复的实现技术

10.4.1 数据转储

10.4.2 登记日志文件

10.4.2 登记日志文件

一、日志文件的格式和内容

二、日志文件的作用

三、登记日志文件

一、日志文件的格式和内容

- 什么是日志文件

  日志文件(log)是用来记录事务对数据库的更新操作的文件

- 日志文件的格式
  - 以记录为单位的日志文件
  - 以数据块为单位的日志文件

日志文件的格式和内容（续）

- 以记录为单位的日志文件内容
  - 各个事务的开始标记(BEGIN TRANSACTION)
  - 各个事务的结束标记(COMMIT或ROLLBACK)
  - 各个事务的所有更新操作

  以上均作为日志文件中的一个日志记录 (log record)
- 以记录为单位的日志文件，每条日志记录的内容
  - 事务标识（标明是哪个事务）
  - 操作类型（插入、删除或修改）
  - 操作对象（记录内部标识）
  - 更新前数据的旧值（对插入操作而言，此项为空值）
  - 更新后数据的新值（对删除操作而言,此项为空值）

日志文件的格式和内容（续）

- 以数据块为单位的日志文件，每条日志记录的内容
  - 事务标识（标明是那个事务）
  - 被更新的数据块

二、日志文件的作用

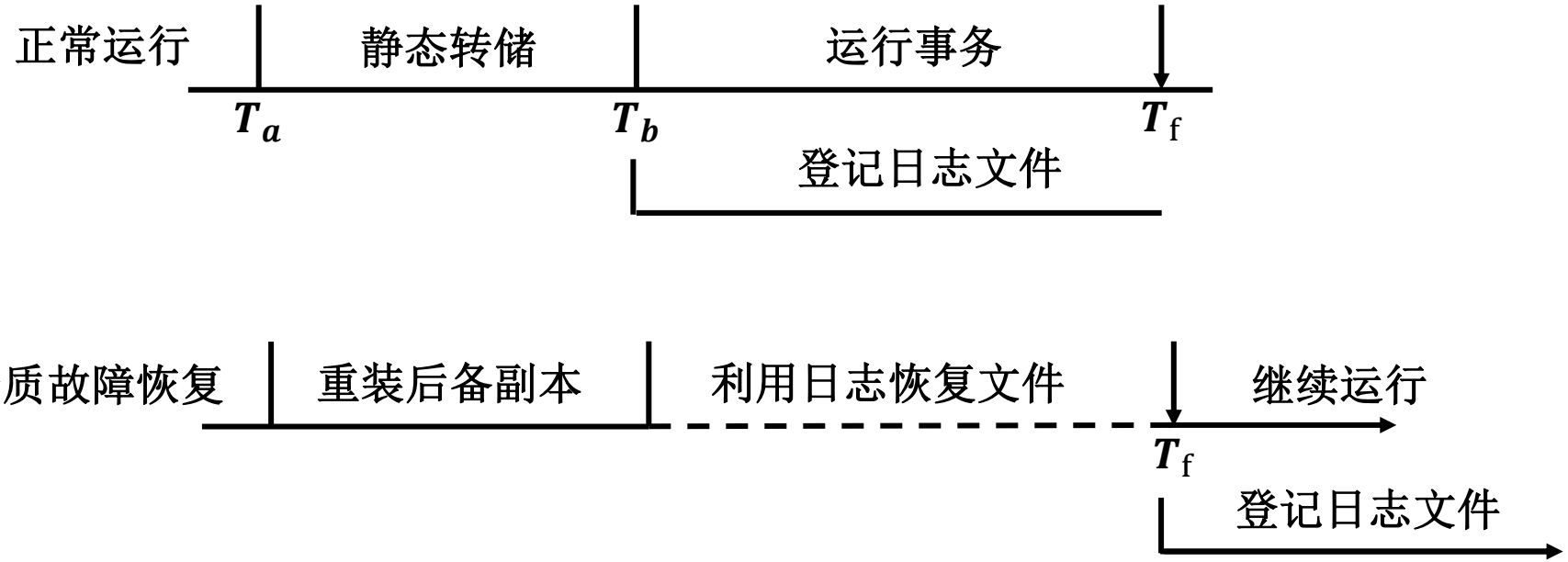
- 进行事务故障恢复
- 进行系统故障恢复
- 协助后备副本进行介质故障恢复
- 具体地：
  - 事务故障恢复与系统故障恢复必须用日志文件
    - **UNDO，REDO**
  - 介质：动态转储必须建立日志文件
  - 介质：静态转储可以建立日志文件

二、日志文件的作用

- 在静态转储方式中，也可以建立日志文件。
  - 当数据库毁坏后可重新装入后备副本把数据库恢复到转储结束时刻的正确状态
  - 利用日志文件，把已完成的事务进行重做处理，尚未完成的事务进行撤销处理
  - 不必重新运行那些已完成的事务程序就可把数据库恢复到故障前某一时刻的正确状态

日志文件的作用

- 利用静态转储副本和日志文件进行恢复



UNDO，REDO

三、登记日志文件

- 两条基本原则
  - 1. 登记的次序严格按并行事务执行的时间次序
  - 2. 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
- 为什么要先写日志文件
  - 写数据库和写日志文件是两个不同的操作（两个写）
  - 在这两个操作之间可能发生故障
  - 如果先写了数据库修改，而在日志文件中没有登记下这个修改，则以后就无法恢复这个修改了
  - 如果先写日志，但没有修改数据库，按日志文件恢复时只不过是多执行一次不必要的UNDO操作，并不会影响数据库的正确性

第十章 数据库恢复技术

10.5 恢复策略

10.5.1 事务故障的恢复

10.5.2 系统故障的恢复

10.5.3 介质故障的恢复

10.5.1 事务故障的恢复

- 事务故障：事务在运行至正常终止点前被终止
  - 一部分完成，一部分未完成
- 恢复方法
  - 由恢复子系统应利用日志文件撤消（UNDO）此事务已对数据库进行的修改
- 事务故障的恢复由系统自动完成，对用户是透明的，不需要用户干预

事务故障的恢复步骤（UNDO）

1. 反向扫描文件日志（即从最后向前扫描日志文件），查找该事务的更新操作
2. 对该事务的更新操作执行逆操作。即将日志记录中“更新前的值” 写入数据库
   - 插入操作，“更新前的值”为空，则相当于做删除操作
   - 删除操作，“更新后的值”为空，则相当于做插入操作
   - 修改操作，则相当于用修改前值代替修改后值
3. 继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理
4. 如此处理下去，直至读到此事务的开始标记，事务故障恢复就完成了

10.5 恢复策略

10.5.1 事务故障的恢复

10.5.2 系统故障的恢复

10.5.3 介质故障的恢复

10.5.2 系统故障的恢复

- 系统故障造成数据库不一致状态的原因
  - 未完成事务对数据库的更新已写入数据库
  - 已提交事务对数据库的更新还留在缓冲区没来得及写入数据库
- 恢复方法
  - **1. Undo 故障发生时未完成的事务**
  - **2. Redo 已完成的事务**
- 系统故障的恢复由系统在重新启动时自动完成，不需要用户干预

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）
   - 重做(REDO) 队列: 在故障发生前已经提交的事务
     - 这些事务既有BEGIN TRANSACTION记录，也有COMMIT记录
   - 撤销 (Undo)队列:故障发生时尚未完成的事务
     - 这些事务只有BEGIN TRANSACTION记录，无相应的COMMIT记录
   - 指针到日志尾部

系统故障的恢复步骤

2. 对撤销(Undo)队列事务进行撤销(UNDO)处理

- 反向扫描日志文件，对每个UNDO事务的更新操作执行逆操作（按事务发生顺序）
- 即将日志记录中“更新前的值”写入数据库

3. 对重做(Redo)队列事务进行重做(REDO)处理

- 正向扫描日志文件，对每个REDO事务重新执行登记的操作
- 即将日志记录中“更新后的值”写入数据库

10.5 恢复策略

10.5.1 事务故障的恢复

10.5.2 系统故障的恢复

10.5.3 介质故障的恢复

10.5.3 介质故障的恢复

1.重装数据库

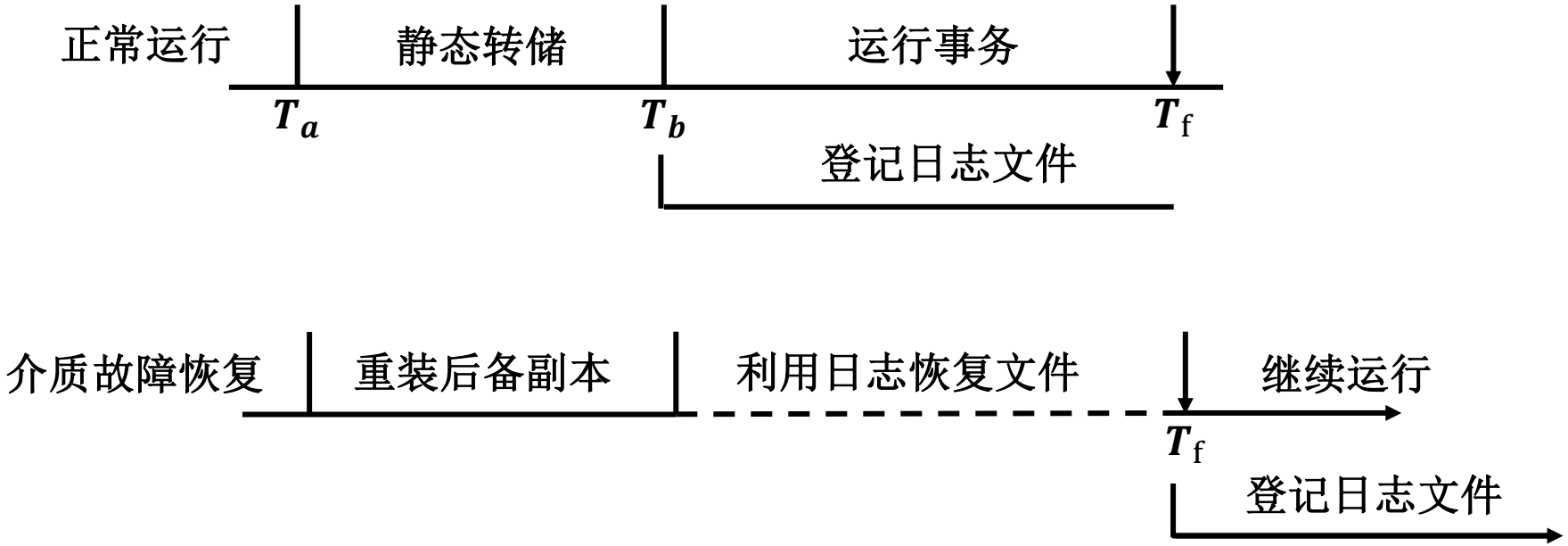
2.重做已完成的事务

介质故障的恢复（续）

- 恢复步骤
  - 1. 装入最新的后备数据库副本(离故障发生时刻最近的转储副本)，使数据库恢复到最近一次转储时的一致性状态
    - 对于静态转储的数据库副本，装入后数据库即处于一致性状态
    - 对于动态转储的数据库副本，还须同时装入转储时刻的日志文件副本，利用与恢复系统故障的方法（即REDO+UNDO），才能将数据库恢复到一致性状态
  - 2. 装入有关的日志文件副本(转储结束时刻的日志文件副本)，重做已完成的事务
    - 首先扫描日志文件，找出故障发生时已提交的事务的标识，将其记入重做队列。
    - 然后正向扫描日志文件，对重做队列中的所有事务进行重做处理。即将日志记录中“更新后的值”写入数据库

日志文件的作用

- 利用静态转储副本和日志文件进行恢复



UNDO，REDO

介质故障的恢复（续）

介质故障的恢复需要DBA介入

- DBA的工作
  - 重装最近转储的数据库副本和有关的各日志文件副本
  - 执行系统提供的恢复命令
- 具体的恢复操作仍由DBMS完成

第十章 数据库恢复技术

10.6 具有检查点的恢复技术

一、问题的提出

二、检查点技术

三、利用检查点的恢复策略

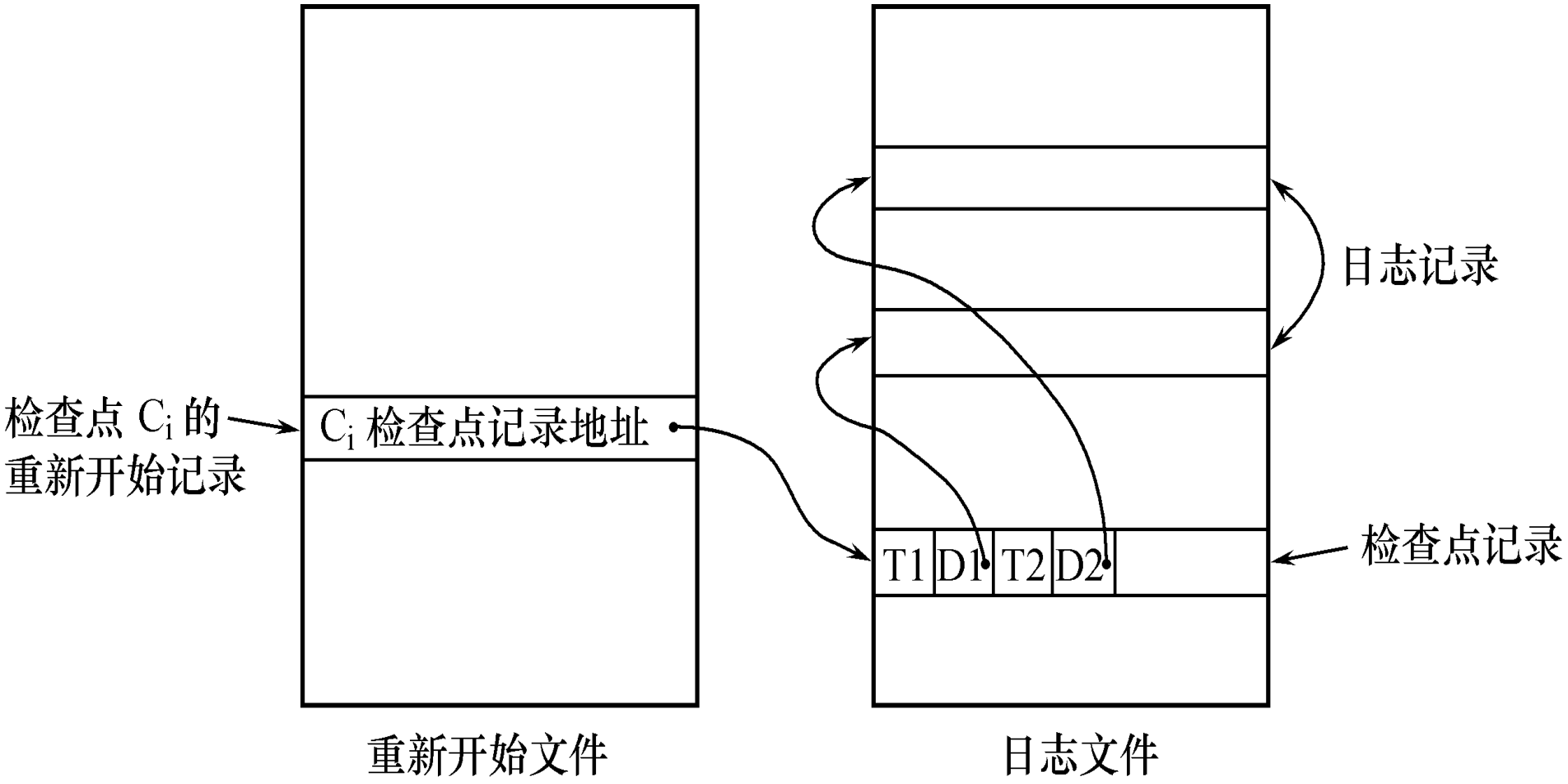
一、问题的提出

- 两个问题
  - 搜索整个日志将耗费大量的时间
  - REDO处理：重新执行，浪费了大量时间
- 具有检查点（checkpoint）的恢复技术
  - 在日志文件中增加检查点记录（checkpoint）
  - 增加重新开始文件（快速找到检查点）
  - 恢复子系统在登录日志文件期间动态地维护日志

二、检查点技术

- 检查点记录的内容
  - 1. 建立检查点时刻所有正在执行的事务清单
  - 2. 这些事务最近一个日志记录的地址
- 重新开始文件的内容
  - 记录各个检查点记录在日志文件中的地址

检查点技术（续）



具有检查点的日志文件和重新开始文件

动态维护日志文件的方法

- 动态维护日志文件的方法

  周期性地执行如下操作：建立检查点，保存数据库状态

  具体步骤是：

  - 1.刷日志缓冲区：将当前日志缓冲区中的所有日志记录写入磁盘的日志文件上
  - 2.记录检查点：在日志文件中写入一个检查点记录
  - 3.刷数据：将当前数据缓冲区的所有数据记录写入磁盘的数据库中
  - 4.记重新开始文件：把检查点记录在日志文件中的地址写入一个重新开始文件

建立检查点

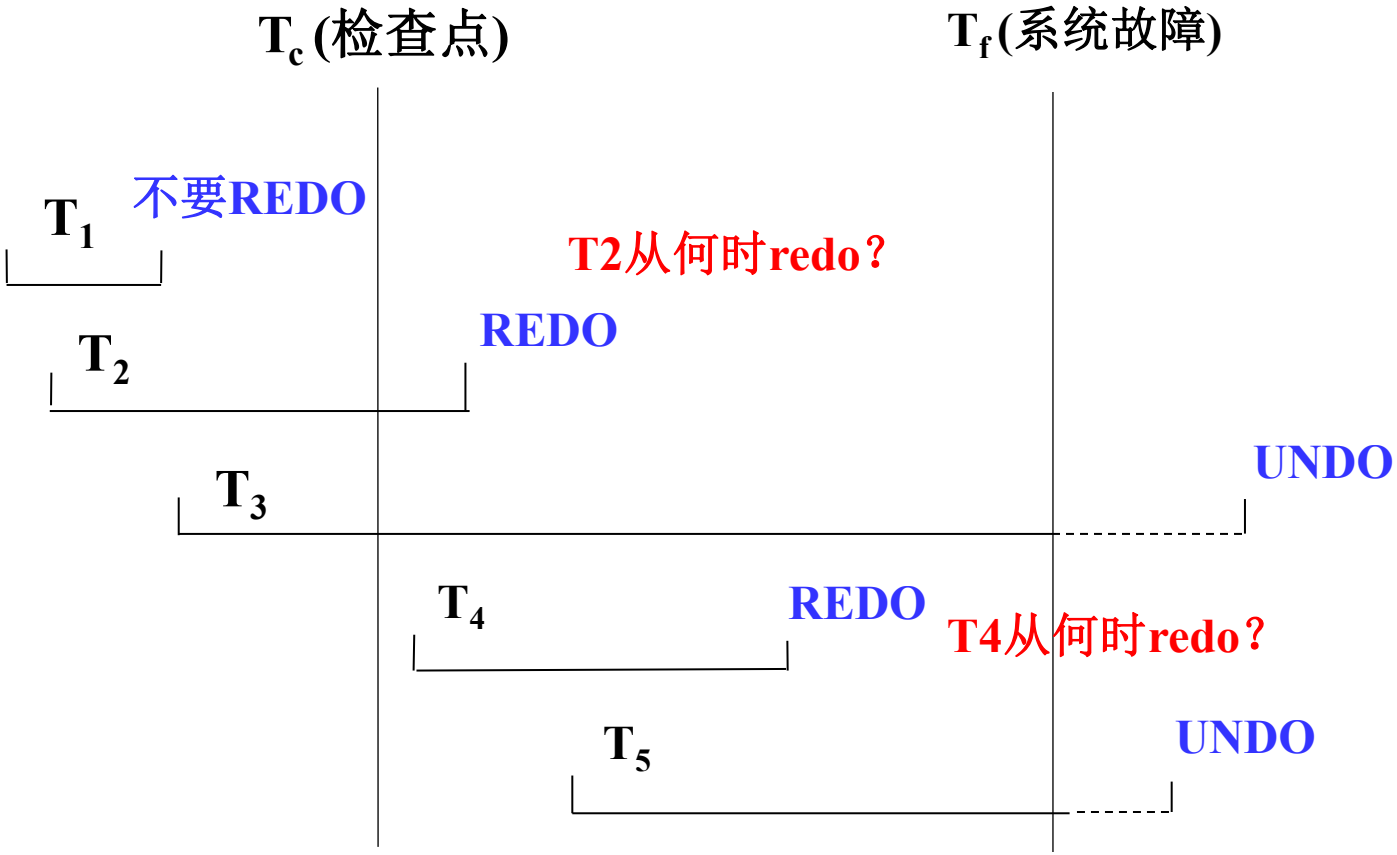
- 恢复子系统可以定期或不定期地建立检查点,保存数据库状态 （通常是DBMS自动做）
  - 定期
    - 按照预定的一个时间间隔，如每隔一小时建立一个检查点
  - 不定期
    - 按照某种规则，如日志文件已写满一半建立一个检查点

三、利用检查点的恢复策略

- 使用检查点方法可以改善恢复效率
  - 当事务T在一个检查点之前提交
    - T对数据库所做的修改已写入数据库
    - 写入时间是在这个检查点建立之前或在这个检查点建立之时
    - 在进行恢复处理时，没有必要对事务T执行REDO操作
  - 当事务T在检查点时未完成
    - T对DB的操作不确定是否写入数据库
    - 恢复处理时，如果需要REDO T，重做的起始点是检查点

利用检查点的恢复策略（续）

系统出现故障时，恢复子系统将根据事务的不同状态采取不同的恢复策略



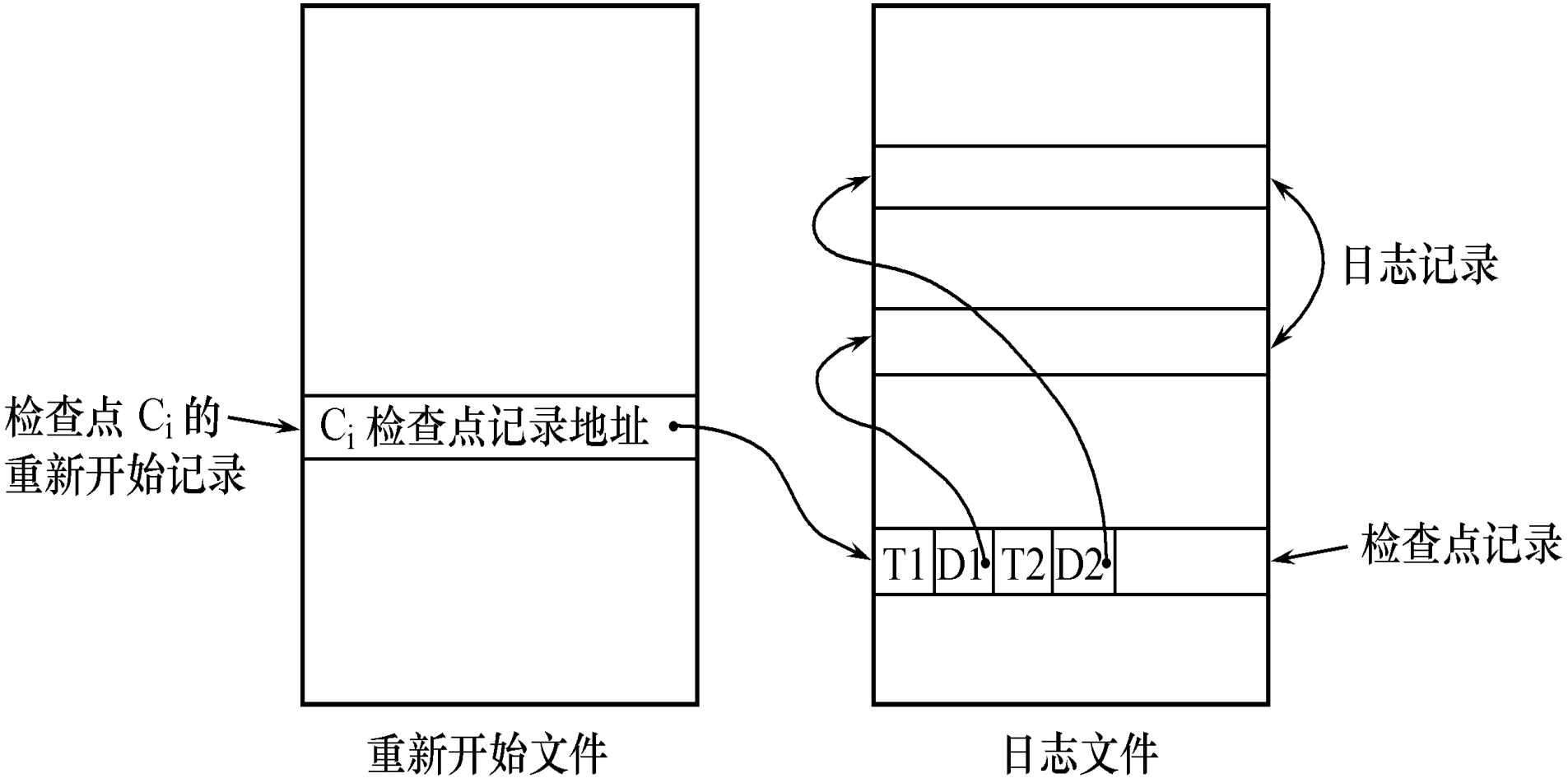
利用检查点的恢复策略（续）

- T1：在检查点之前提交
- T2：在检查点之前开始执行，在检查点之后故障点之前提交
- T3：在检查点之前开始执行，在故障点时还未完成
- T4：在检查点之后开始执行，在故障点之前提交
- T5：在检查点之后开始执行，在故障点时还未完成

恢复策略：

- T3和T5在故障发生时还未完成，所以予以撤销UNDO
- T2和T4在检查点之后才提交，它们对数据库所做的修改在故障发生时可能还在缓冲区中，尚未写入数据库，所以要REDO
- T1在检查点之前已提交，所以不必执行REDO操作

利用检查点的恢复步骤



具有检查点的日志文件和重新开始文件

利用检查点的恢复步骤

- 1.从重新开始文件中找到最后一个检查点记录在日志文件中的地址，由该地址在日志文件中找到最后一个检查点记录
- 2.由该检查点记录得到检查点建立时刻所有正在执行的事务清单ACTIVE-LIST
  - 建立两个事务队列
    - UNDO-LIST
    - REDO-LIST
  - 把ACTIVE-LIST暂时放入UNDO-LIST队列，REDO队列暂为空

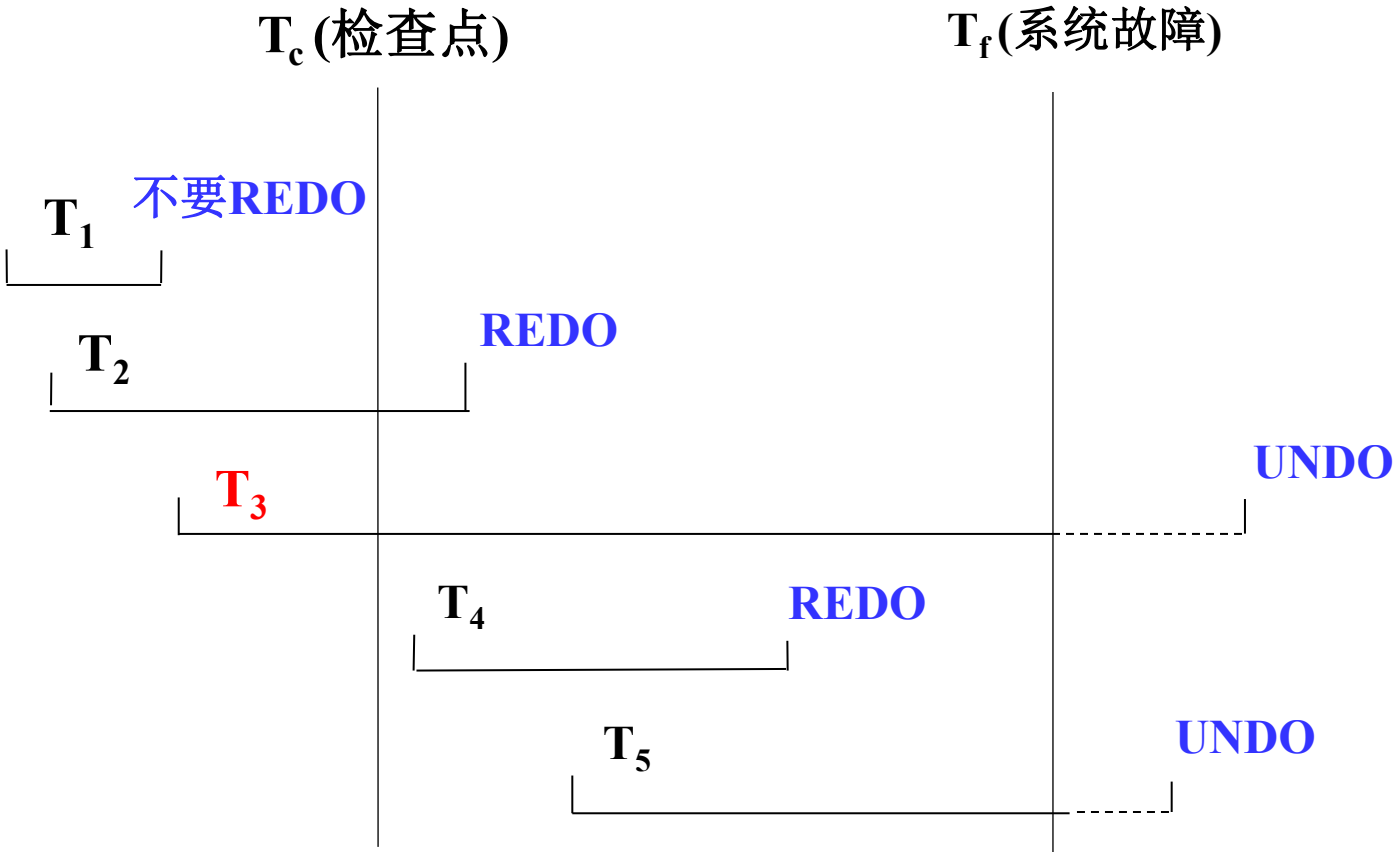
利用检查点的恢复策略（续）

- 3.从检查点开始正向扫描日志文件，直到日志文件结束
  - 如有新开始的事务T_i，把T_i暂时放入UNDO-LIST队列
  - 如有提交的事务T_j，把T_j从UNDO-LIST队列移到REDO-LIST队列
- 4.反向扫: 对UNDO-LIST中的每个事务执行UNDO操作

  正向扫：对REDO-LIST中的每个事务执行REDO操作

利用检查点的恢复策略（续）

系统出现故障时，恢复子系统将根据事务的不同状态采取不同的恢复策略

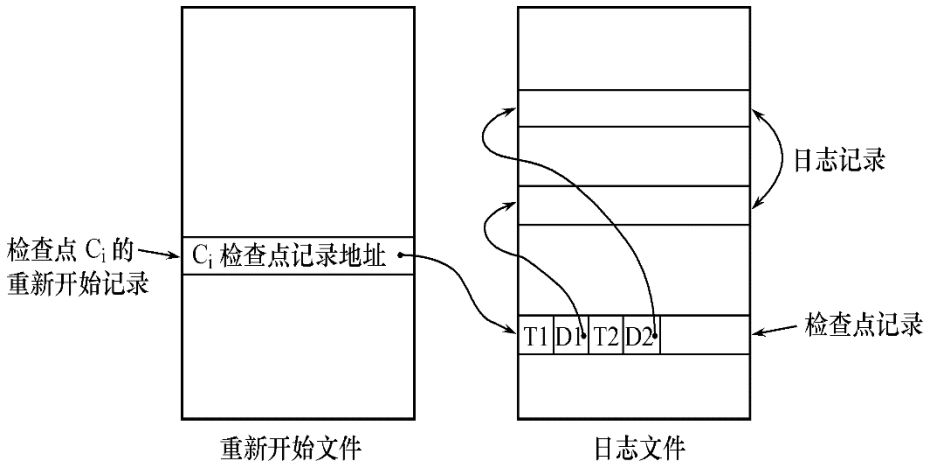


思考一：动态维护日志文件的方法

建立检查点

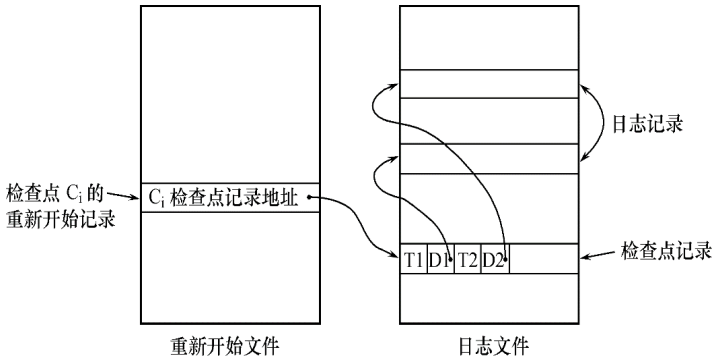
- 1.刷日志缓冲区：将当前日志缓冲区中的所有日志记录写入磁盘的日志文件上
- 2.记录检查点：在日志文件中写入一个检查点记录
- 3.刷数据：将当前数据缓冲区的所有数据记录写入磁盘的数据库中
- 4.记重新开始文件：把检查点记录在日志文件中的地址写入一个重新开始文件

问，如果在2-3之间出故障了？
如果在3-4之间出故障了？



思考二：建立检查点的作用

- 恢复子系统可以定期或不定期地建立检查点,保存数据库状态 （通常是DBMS自动做）
  - 定期
    - 按照预定的一个时间间隔，如每隔一小时建立一个检查点
  - 不定期（如何选择规则？）
    - 按照某种规则，如日志文件已写满一半建立一个检查点
- 扔日志文件？怎么扔？



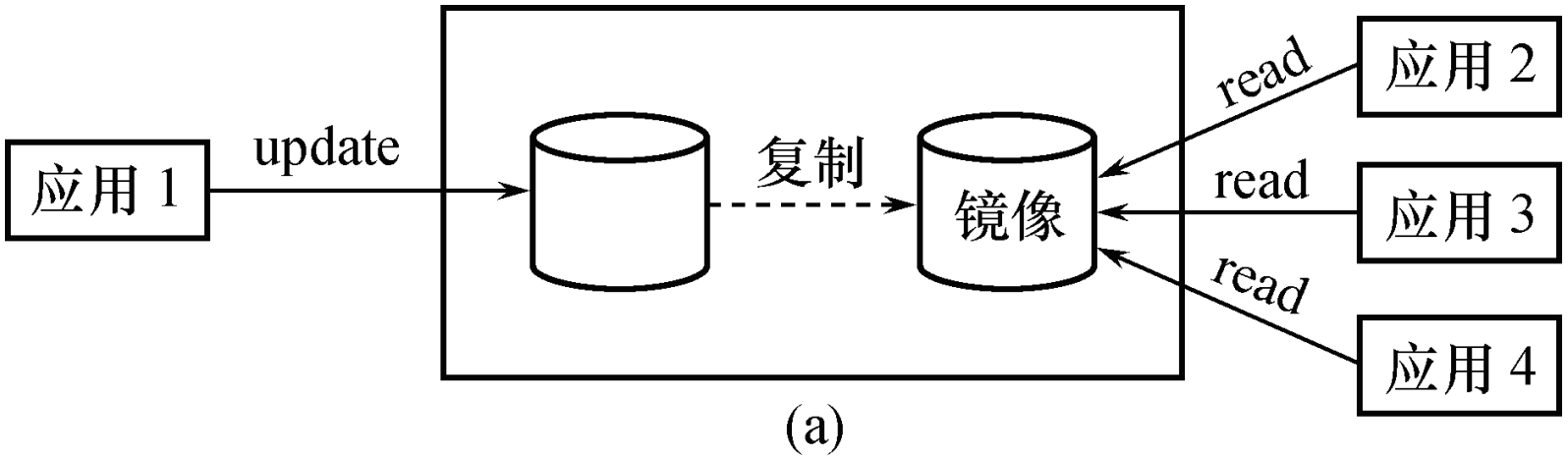
第十章 数据库恢复技术

10.7 数据库镜像

- 介质故障是对系统影响最为严重的一种故障，严重影响数据库的可用性
  - 介质故障恢复比较费时
  - 为预防介质故障，DBA必须周期性地转储数据库
- 提高数据库可用性的解决方案
  - 数据库镜像（Mirror）

数据库镜像（续）

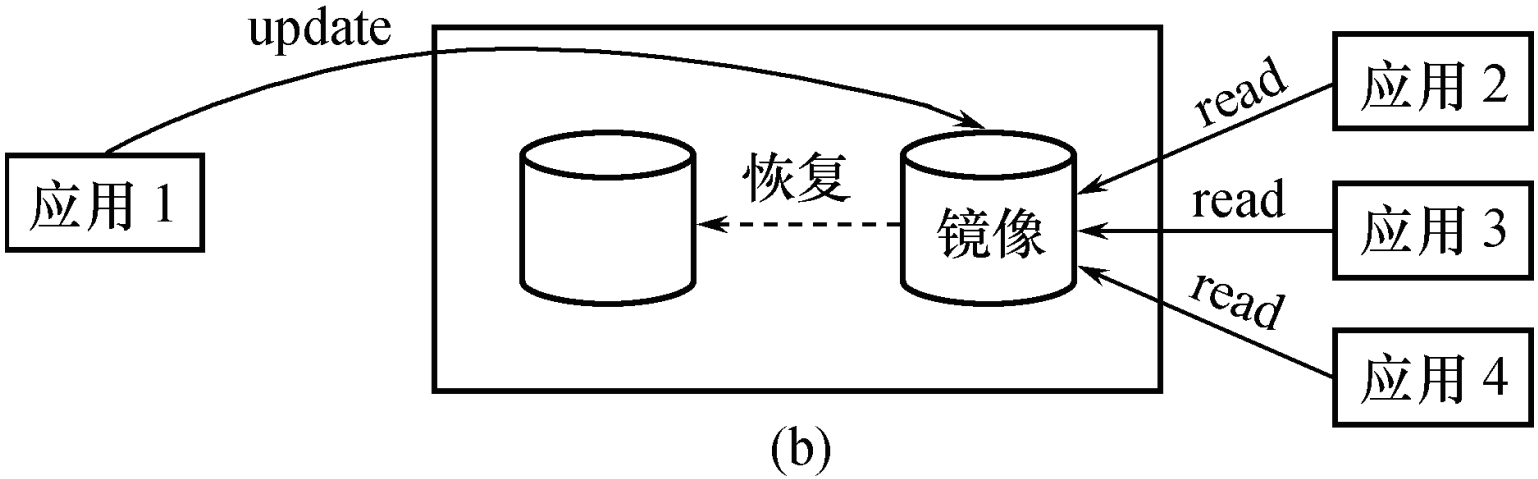
- 数据库镜像
  - DBMS自动把整个数据库或其中的关键数据复制到另一个磁盘上
  - DBMS自动保证镜像数据与主数据库的一致性
- 每当主数据库更新时，DBMS自动把更新后的数据复制过去



(a)

数据库镜像的用途

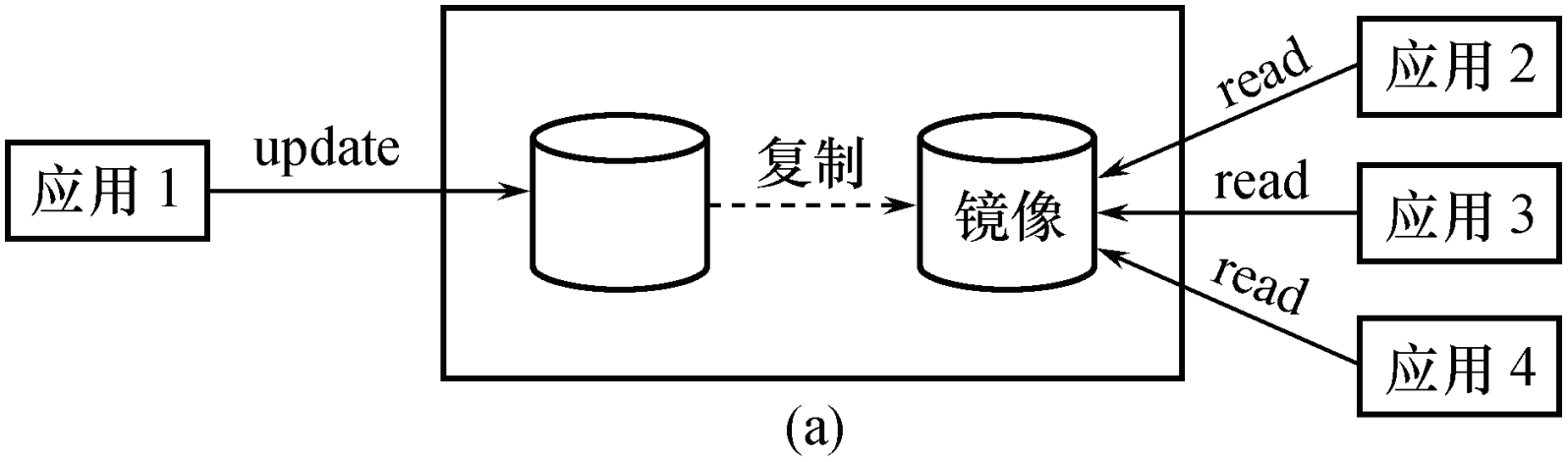
- 出现介质故障时
  - 可由镜像磁盘继续提供使用
  - 同时DBMS自动利用镜像磁盘数据进行数据库的恢复
  - 不需要关闭系统和重装数据库副本(如下图所示)



(b)

数据库镜像（续）

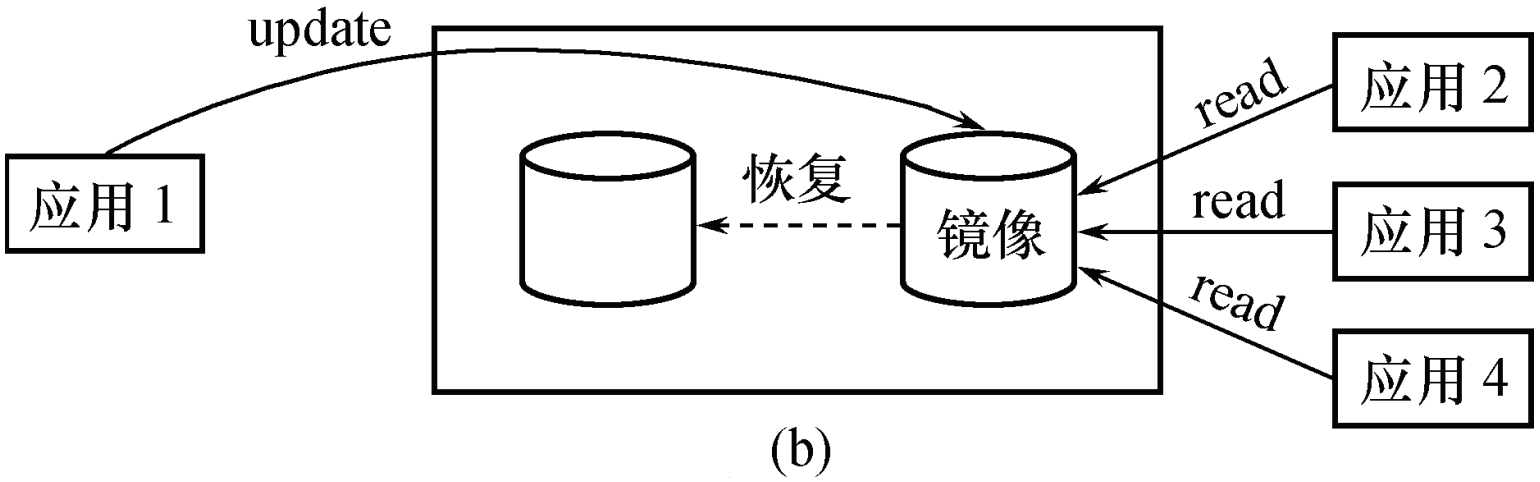
- 没有出现故障时
  - 可用于并发操作
  - 一个用户对数据加排他锁修改数据，其他用户可以读镜像数据库上的数据，而不必等待该用户释放锁



(a)

数据库镜像（续）

- 频繁地复制数据自然会降低系统运行效率
  - 在实际应用中用户往往只选择对关键数据和日志文件镜像，而不是对整个数据库进行镜像



(b)

第十章 数据库恢复技术

10.8 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

小结（续）

- 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

小结（续）

- 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**

小结（续）

- 提高恢复效率的技术
  - 检查点技术
    - 可以提高系统故障的恢复效率
    - 可以在一定程度上提高利用动态转储备份进行介质故障恢复的效率
  - 镜像技术
    - 镜像技术可以改善介质故障的恢复效率