CONTENTS

恢复系统

911, 灾难已经给人们留下了巨大的伤痛，但这远远没有结束，当重建工作遭遇数据灾难恢复难题时，数据丢失带来的二次灾难正在上演。据统计，“9.11”事故一年后，重返世贸大厦的企业由原先的 350 家变成 150 家，另外 200 家企业由于重要信息系统的破坏，关键数据的丢失而永远的消失了。再来看看国外一些数据灾难恢复研究机构的统计：金融业在灾难停机两天内所受损失为日营业额的 50%；如果在两星期内无法进行数据灾难恢复方案，75% 的公司将业务停顿，43% 的公司将再也无法开业；没有实施数据灾难恢复方案的公司 60% 将在灾难后 2-3 年间破产。由此可见，数据灾难恢复方案对灾难后社会的正常运行起到了非常关键的作用。

恢复系统

■要点：

- 故障是不可避免会发生的。
- 任何大型的数据库管理系统都预先采取了措施（恢复机制）保证系统发生故障后仍能保持事务的原子性和持久性。
- 恢复机制负责将数据库恢复到故障发生前的某个一致性状态。

■事务故障

■系统故障

■介质故障

■其它故障

故障分类、特征及恢复策略

■对于不同类型的故障在恢复时应做不同的恢复处理。

■恢复的本质是利用存储的冗余数据（如日志、影子、备份副本等）来重建数据库中已经被破坏或已经不正确的那部分数据。

■DBMS 中的恢复管理模块由两部分组成:

- 正常事务处理过程中：系统需记录冗余的恢复信息，以保证故障发生后有足够的信息进行数据库恢复;
- 故障发生后：利用冗余信息进行 UNDO 或 REDO 等操作，将数据库恢复到一致性状态。

日志及特点

■日志是 DBMS 记录数据库全部更新操作的序列文件。下面介绍日志特点、记录类型，记录格式，撤销及重做操作。

■主要特点有：

- 日志文件记录了数据库的全部更新顺序。
- 日志文件是一个追加（append-only）文件。
- DBMS 允许事务的并发执行导致日志文件是“交错的”。
- 属于单个事务的日志顺序与该事务更新操作的执行顺序是一致的。

■为了保证数据库能运用日志进行恢复，要求日志文件必须放到稳固存储器（如磁盘阵列）上，并且要求每条日志记录必须在其所包含数据记录的更新值写到外存储器之前先写到稳固存储器上，即先写 (write-ahead) 日志规则

日志记录类型

■数据库中的日志记录有两种类型:

- 记录数据更新操作的日志记录，包括 UPDATE 、 INSERT 和 DELETE 操作;
- 记录事务操作的日志记录，包括 START 、 COMMIT 和 ABORT 操作。

日志记录格式

- <T_i, A, V_1, V_2> 表示事务 T_i 对数据元素 A 执行了更新操作，V_1 表示 A 更新前的值（前映像），V_2 表示 A 更新后的值（后映像）。对于插入操作，V_1 为空；对于删除操作，V_2 为空。
- <T_i, START> 表示事务 T_i 已经开始。此时 DBMS 完成对事务的初始化工作，如分配事务工作区等。
- <T_i, COMMIT> 表示事务 T_i 已经提交，即事务 T_i 已经执行成功（该事务对数据库的修改必须永久化）。事务提交时其更新的数据都写到了数据缓冲区中，但是由于不能控制缓冲区管理器何时将缓冲块从内存写到磁盘。因此当看到该日志记录时，通常不能确定更新是否已经写到磁盘上。
- <T_i, ABORT> 表示事务已经中止，即事务执行失败。此时，如果 T_i 所做的更新已反映到磁盘上，DBMS 必须通过 UNDO 操作来消除 T_i 对磁盘数据库的影响。

UNDO 操作

■对于要 UNDO 的事务 T，日志中记录有 <T, START> 以及 T 对数据库的所有更新操作的日志记录。

■UNDO 过程为：从 T 的最后一条更新日志记录开始，从日志尾向日志头 (反向) 依次将 T 更新的数据元素值恢复为旧值 (V_1) 。

REDO 操作

■与 UNDO 相反，其是对已提交事务进行重做。

■对于要 REDO 的事务 T，日志中已经记录了 <T, START> 、 T 的所有更新操作日志以及 <T, COMMIT> 。

■REDO 过程为：从 T 的第一条更新日志记录开始，从日志头向日志尾 (顺向) 依次将 T 更新的数据元素值恢复为新值 (V_2) 。

基于日志恢复策略举例

[例 10] 考虑订票事务 T_1 和 T_2，除更新航班的剩余票数 A 外，还分别需更新售票点的售出票数 X 和 Y。假设先执行 T_1，再执行 T_2，几种可能的日志记录如图 -19 所示。

<T_1, START>
<T_1, X, 0, 2>
<T_1, A, 10, 8>

(a)

图 -19 T_1 和 T_2 串行执行的三种日志情形

基于日志恢复策略举例

■情形 (a)： T_1 完成 WRITE(A) 后系统发生崩溃。当系统重启动时，检查到有 <T_1, START>，但没有 <T_1, COMMIT>。因此恢复时执行 UNDO(T_1)，将 X 和 A 的值分别恢复为 0 和 10。

<T_1, START>
<T_1, X, 0, 2>
<T_1, A, 10, 8>

(a)

图 -19 T_1 和 T_2 串行执行的三种日志情形

基于日志恢复策略举例

■情形 (b) ：T_2 完成 WRITE(A) 后系统发生崩溃。这时需分别执行两个恢复操作 UNDO(T_2) 和 REDO(T_1)，因为 T_1 既有 <T_1, START>，又有 <T_1, COMMIT> 日志，而 T_2 只有 <T_2, START> 日志。恢复完成后 X、Y 和 A 的值分别为 2、0 和 8。

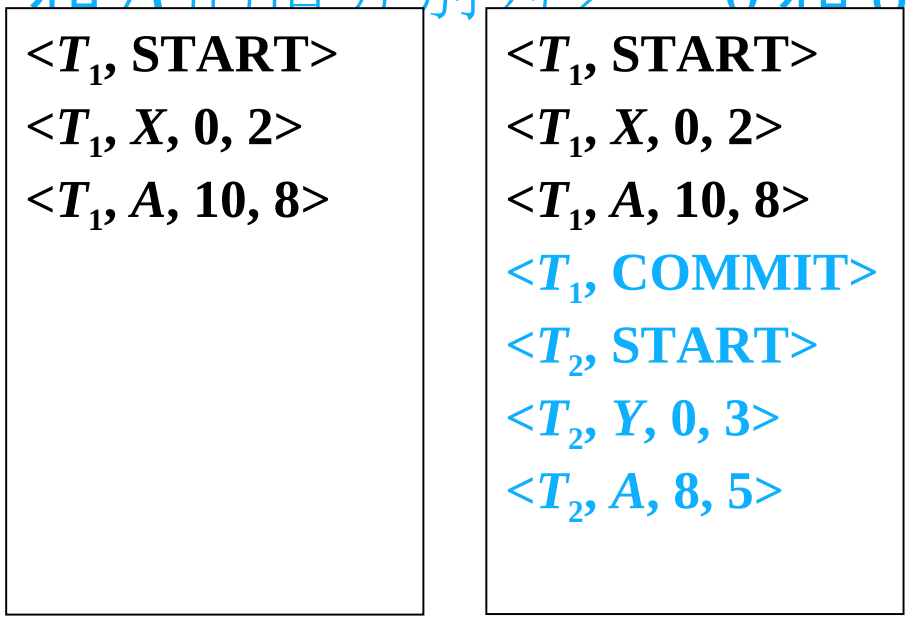

图 -19 T_1 和 T_2 串行执行的三种日志情形

基于日志恢复策略举例

■情形 (c) ： T_2 完成提交后系统发生崩溃。这时也需执行两个恢复操作 REDO(T_1) 和 REDO(T_2)， 因为 T_1 和 T_2 都有 START 和 COMMIT 日志。恢复完成后 X 、 Y 和 A 的值分别为 2 、 3 和 5 。

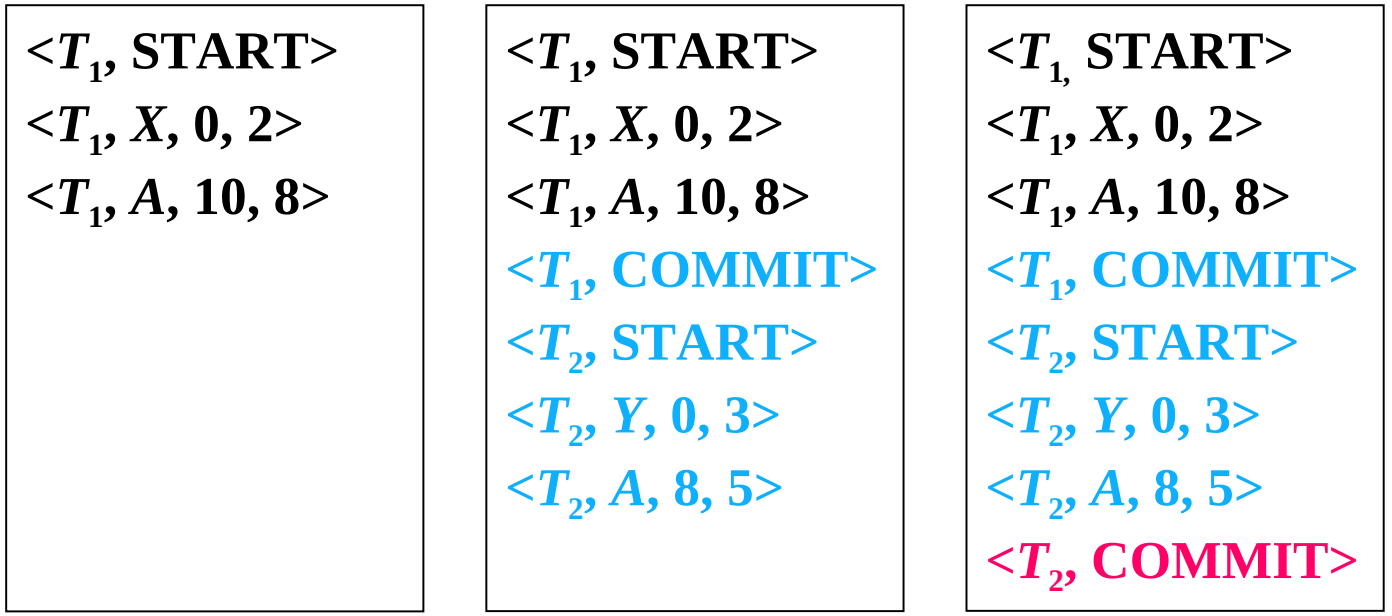

图 -19 T_1 和 T_2 串行执行的三种日志情形

并发执行事务的基本恢复过程?

并发执行事务的基本恢复过程

■三个阶段：

- 分析阶段：从日志头开始顺向扫描日志，确定重做事务集（REDO-set）和撤销事务集（UNDO-set）。将既有 <*T*, START> 又有 <*T*, COMMIT> 日志记录的事务 *T* 加入 REDO-set；将只有 <*T*, START> 没有 <*T*, COMMIT> 日志记录的事务 *T* 加入 UNDO-set。
- 撤销阶段：从日志尾反向扫描日志，对每一条属于 UNDO-set 中事务的更新操作日志依次执行 UNDO 操作。
- 重做阶段：从日志头顺向扫描日志，对每一条属于 REDO-set 中事务的更新操作日志依次执行 REDO 操作。

■UNDO 与 REDO 必须是幂等的，即重复执行任意次的结果与执行一次的结果是一样的。

日志太大，如何办?

引入检查点目的

■利用日志文件恢复主要有两个问题：

- 日志扫描过程太耗时。因为日志文件必须保存在磁盘中，而且随着时间的不断推进，日志文件在不断扩大，扫描的时间也就变得越来越长。
- 许多要求 REDO 事务的更新实际上在恢复时都写入了磁盘的物理数据库中。尽管对它们做 REDO 操作不会造成不良后果，但会使恢复过程变得更长，导致数据库系统停止服务延长，从而降低了数据库的可用性。

■为了减少扫描开销和提高恢复效率，引入了检查点技术。

检查点技术

■检查点是周期性地向日志中写一条检查点记录并记录所有当前活跃的事务，为恢复管理器提供信息，以决定从日志的何处开始恢复。

■检查点工作主要包括:

- 将当前位于日志缓冲区的所有日志记录输出到磁盘上;
- 将当前位于数据缓冲区的所有更新数据块输出到磁盘上;
- 记录日志记录 <Checkpoint *L*> 并输出到磁盘上，其中 *L* 是做检查点时活跃事务的列表。

根据系统是否运行等，检查点分类有哪些?

检查点技术分类

- 静态检查点技术：在检查点执行过程中，不允许事务执行任何更新动作，如写缓冲块或写日志记录，称其为静态检查点技术。
- 如果事务 *T* 在做检查点之前就已提交，那么它的 <*T*, COMMIT> 记录一定出现在 <Checkpoint *L*> 记录前，并且其更新在做 Checkpoint 时都已写到磁盘中，因此不需要对 *T* 做任何恢复操作，这样可大大减少恢复工作量。
- 模糊检查点技术：如果数据缓冲区及日志缓冲区中缓存的更新数据很多时，就会导致系统长时间不能接受事务处理，这对响应时间要求较严格的系统来说是不可忍受的。为避免这种中断，可使用模糊检查点 (fuzzy checkpoint) 技术，允许在做检查点的同时接受数据库更新操作。

基于检验点恢复方法

■图 -20 是系统崩溃时的不同事务状态类型，其中 t_c 为完成最近检查点时刻，t_f 为故障发生时刻。

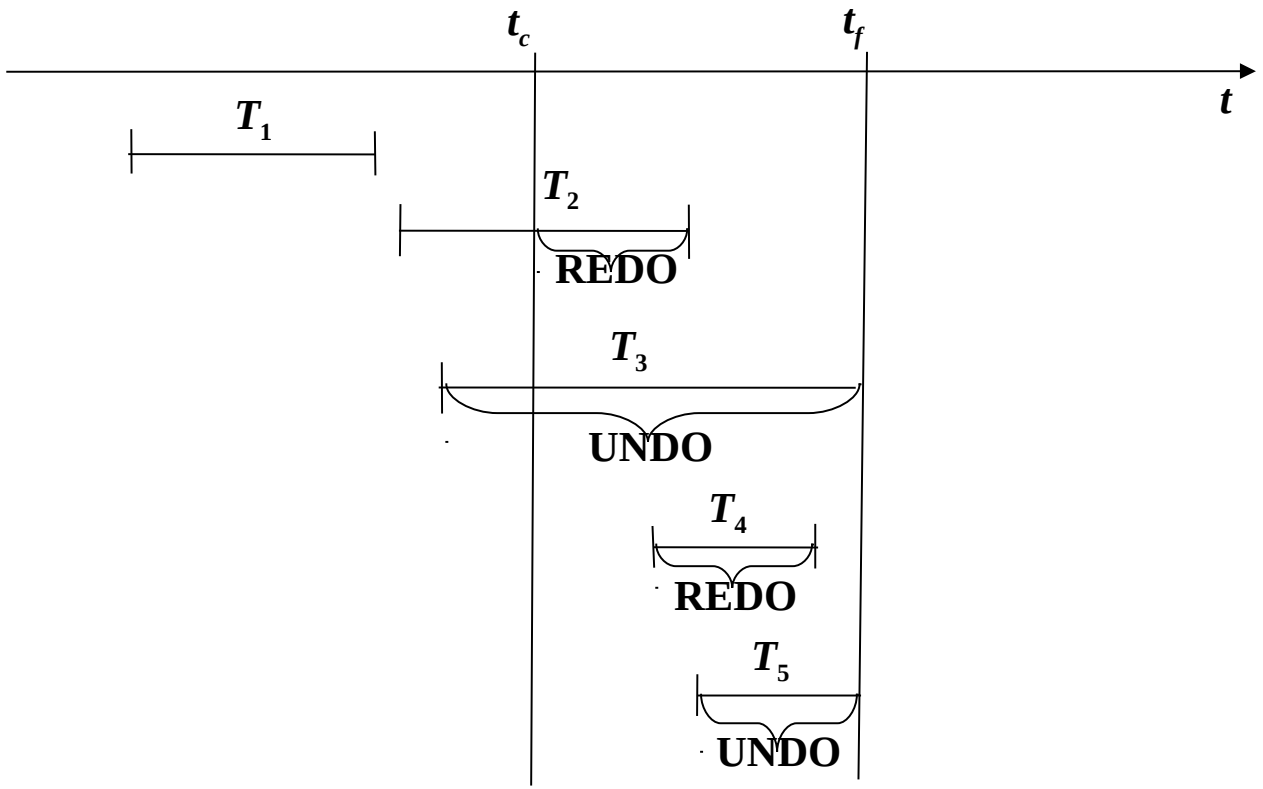

图 -20　系统崩溃时不同状态事务的不同恢复处理

基于检验点恢复方法

- T_1类事务的更新在做检查点之前已经写到磁盘上，故不用做任何恢复操作。

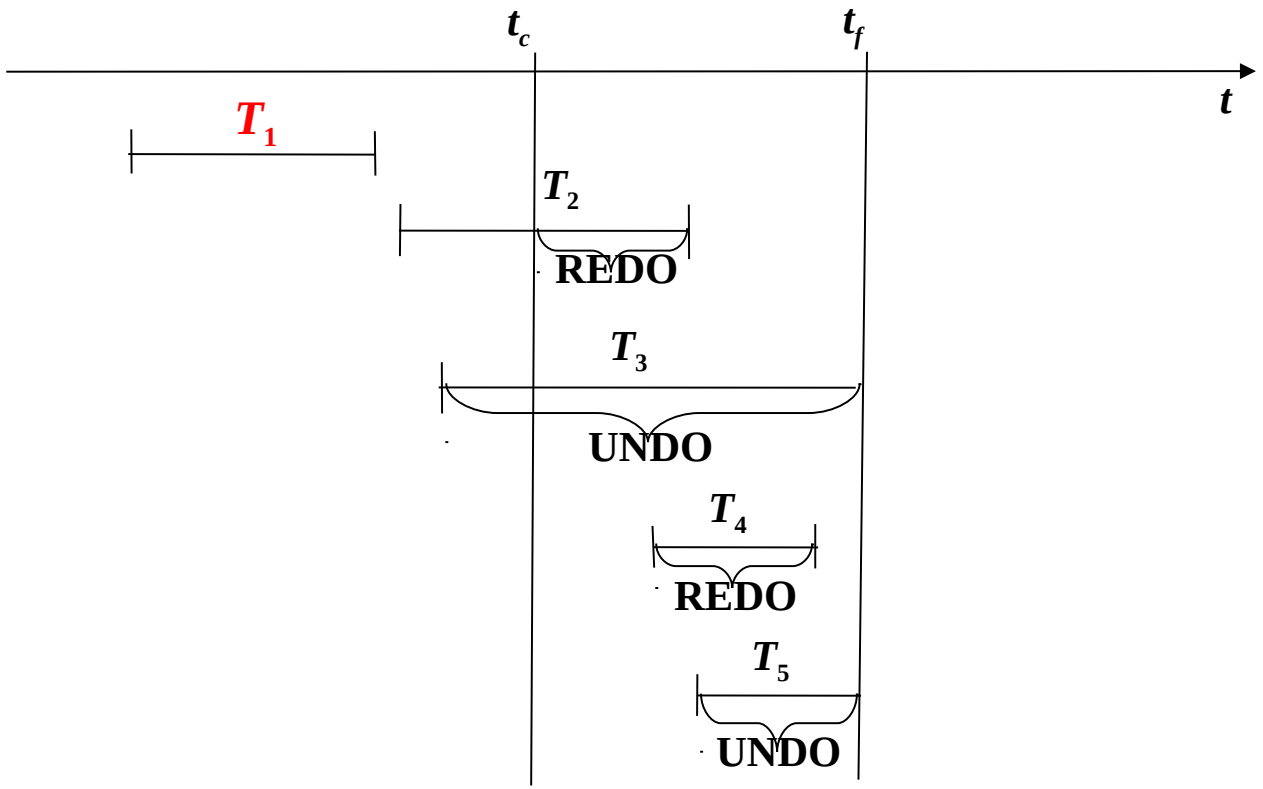

图 -20　系统崩溃时不同状态事务的不同恢复处理

基于检验点恢复方法

- T_2 类事务在 t_c 前的更新已写到磁盘，故重做时只需根据 t_c 之后的日志记录进行 REDO 即可。

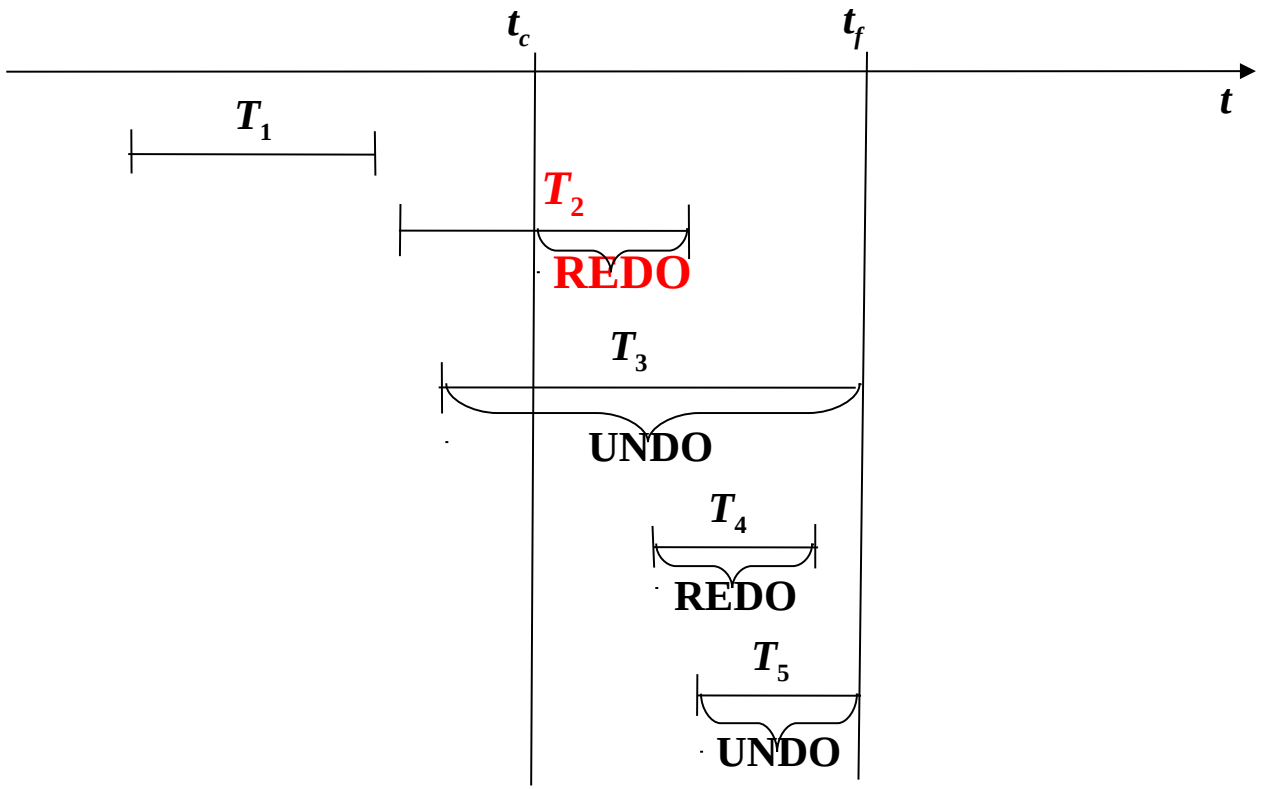

图 -20　系统崩溃时不同状态事务的不同恢复处理

基于检验点恢复方法

- T_3类事务为在 t_c 之前开始且在 t_f 之前仍未结束的事务，这类事务需要全部撤销。

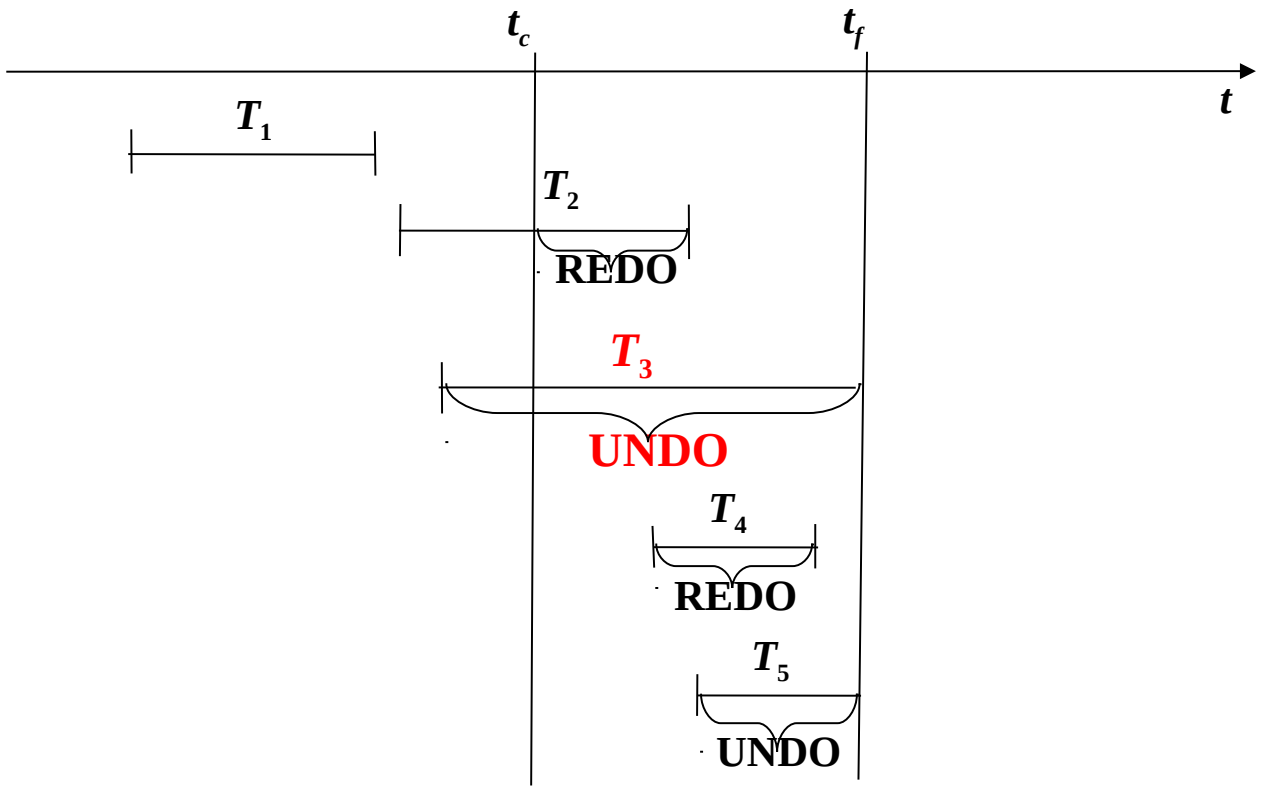

图 -20　系统崩溃时不同状态事务的不同恢复处理

基于检验点恢复方法

- T_4类事务为在 t_c 之后开始且在 t_f 之前已完成的事务，这类事务需要全部重做。

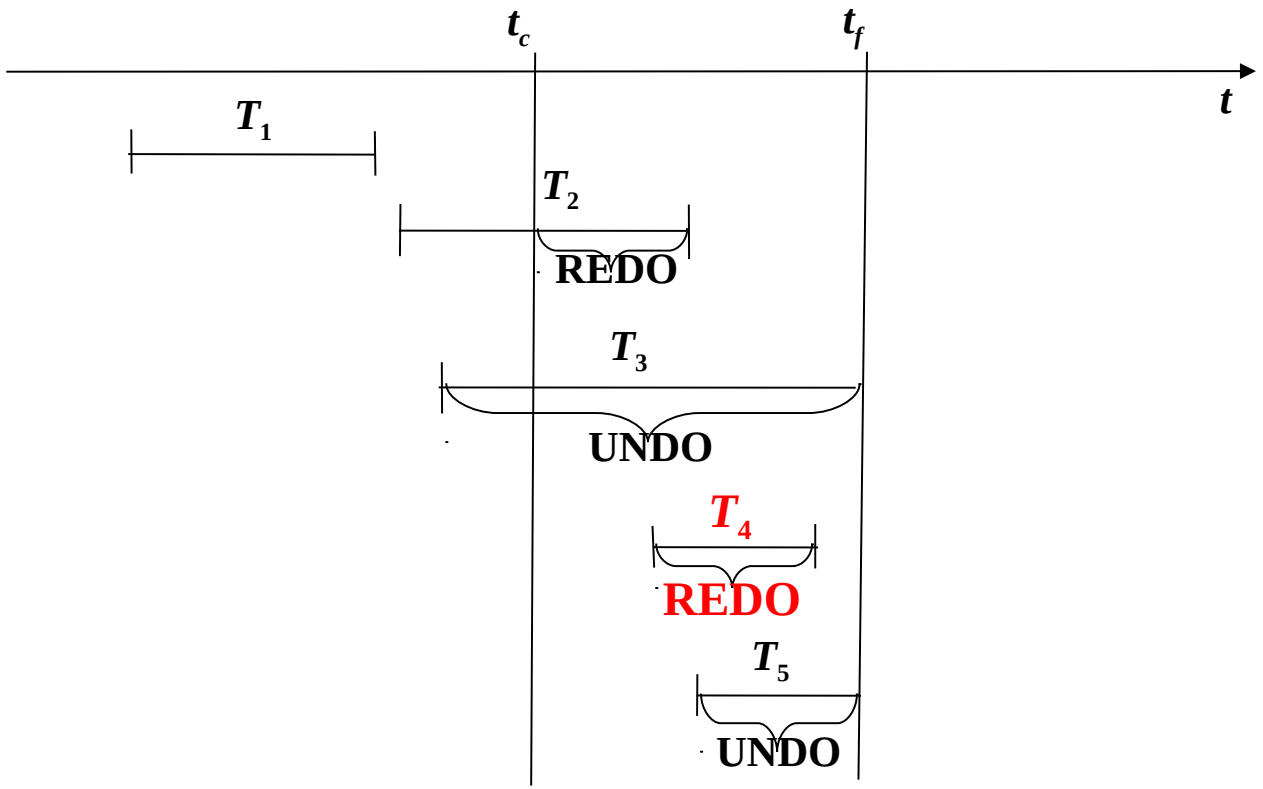

图 -20　系统崩溃时不同状态事务的不同恢复处理

基于检验点恢复方法

- T_5类事务为在 t_c之后开始且在 t_f之前仍未完成的事务，这类事务应全部撤销。

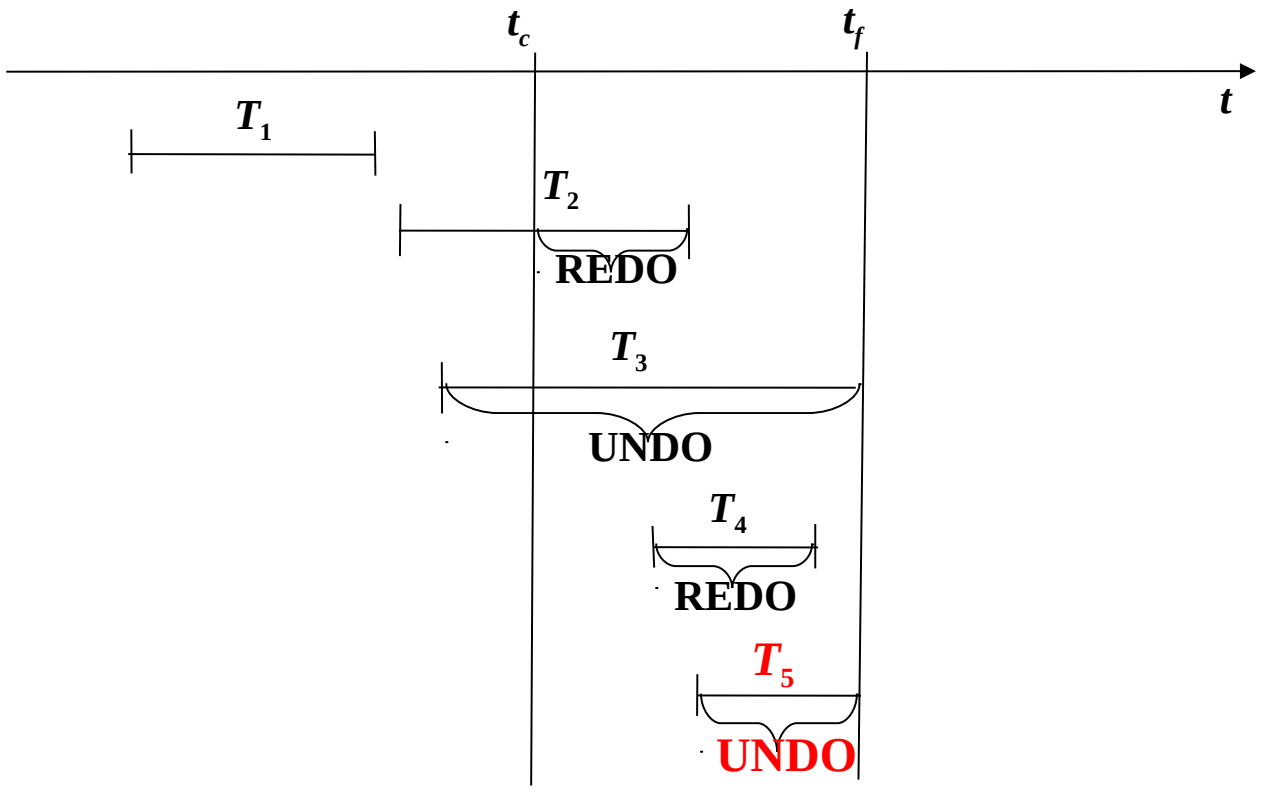

图 -20　系统崩溃时不同状态事务的不同恢复处理

检查点技术

■ [例 11] 系统崩溃时日志文件记录内容如图 -21 所示，试写出系统重启后恢复处理的步骤及恢复操作 (指 UNDO, REDO 操作) ，并指明 *A*, *B*, *C*, *D* 恢复后的值分别是多少?

■ 分析阶段：从最后一次检查点开始顺向扫描日志，确定重做事务集 REDO-set 和撤销事务集 UNDO-set 。

- 将既有 <*T*, START> 又有 <*T*, COMMIT> 日志记录的事务 *T* 加入 REDO-set ;
- 将只有 <*T*, START> 没有 <*T*, COMMIT> 日志记录的事务 *T* 加入 UNDO-set 。

<T_0 , START>
<T_1 , START>
<T_0 , *A*, 2, 12>
<T_2 , START>
<T_0 , COMMIT>
<T_1 , *C*, 6, 16>
<T_2 , *B*, 4, 14>
<Checkpoint {T_1, T_2}>
<T_1 , *A*, 12, 20>
<T_3 , START>
<T_1 , COMMIT>
<T_2 , *B*, 14, 40>
<T_3 , *A*, 20, 60>
<T_4 , START>
<T_3 , *D*, 8, 18>
<T_3 , COMMIT>
<T_4 , *C*, 16, 30>

图 -21 日志文件

■ 分析过程如下：

	UNDO-set	REDO-set
➢ <Checkpoint {T_1, T_2}>	{T_1, T_2}	{ }
➢ <T_3, START>	{T_1, T_2, T_3}	{ }
➢ <T_1, COMMIT>	{T_2, T_3}	{T_1}
➢ <T_4, START>	{T_2, T_3, T_4}	{T_1}
➢ <T_3, COMMIT>	{T_2, T_4}	{T_1, T_3}

<T_0 , START>
<T_1 , START>
<T_0 , A, 2, 12>
<T_2 , START>
<T_0 , COMMIT>
<T_1 , C, 6, 16>
<T_2 , B, 4, 14>
<Checkpoint {T_1, T_2}>
<T_1 , A, 12, 20>
<T_3 , START>
<T_1 , COMMIT>
<T_2 , B, 14, 40>
<T_3 , A, 20, 60>
<T_4 , START>
<T_3 , D, 8, 18>
<T_3 , COMMIT>
<T_4 , C, 16, 30>

图 -21 日志文件

检查点技术

■ 撤销过程如下：

<$T4$，C, 16, 30> ： C=16

<$T2$，B, 14, 40> ： B=14

<$T2$，B, 4, 14> ： B=4

■ 撤销后的结果为： B=4 ， C=16 。

■ [… 1 所
示 … 操作
(… 复后
的 …

■ 扌 … 至遇
到最后一次检查点止，对每一条属于 UNDO-set={T_2, T_4} 中事务的更新操作日志依次执行 UNDO 操作。

● 其次，对于同时出现在 UNDO-set={T_2, T_4} 与 <Checkpoint {T_1, T_2}> 列表中的事务集 {T_2}，从最后一次检查点开始继续反向扫描日志至遇到这些事务的 START 止，对属于 {T_2} 中事务的更新操作日志依次执行 UNDO 操作。

<T_0 ， START>
<T_1 ， START>
<T_0 ， A, 2, 12>
<T_2 ， START>
<T_0 ， COMMIT>
<T_1 ， C, 6, 16>
<T_2 ， B, 4, 14>
<Checkpoint {T_1, T_2}>
<T_1 ， A, 12, 20>
<T_3 ， START>
<T_1 ， COMMIT>
<T_2 ， B, 14, 40>
<T_3 ， A, 20, 60>
<T_4 ， START>
<T_3 ， D, 8, 18>
<T_3 ， COMMIT>
<T_4 ， C, 16, 30>

图 -21 日志文件

检查点技术

■ [例　11] 系统崩溃时日志文件记录内容如图 -21 所示，试写出系统重启后恢复处理的步骤，并指明 *A*, *B*, *C*, *D* 恢复后的值分别是多少？

恢复完成后的结果为：*A*=60 ， *B*=4 ， *C*=16 ， *D*=18 。

■ 重做阶段：从最后一次检查点开始顺向扫描日志，对每一条属于 REDO-set={T_1, T_3} 中事务的更新操作日志依次执行 REDO 操作。

■ 重做过程如下：

<T_1 ， *A*, 12, 20> ： *A*=20

<T_3 ， *A*, 20, 60> ： *A*=60

<T_3 ， *D*, 8, 18> ： *D*=18

■ 重做后的结果为：*A*=60 ， *D*=18 。

<T_0 ， START>
<T_1 ， START>
<T　　*A*, 2, 12>
<T_1 ， *C*, 6, 16>
<T_2 ， *B*, 4, 14>
<Checkpoint {T_1, T_2}>
<T_1 ， *A*, 12, 20>
<T_3 ， START>
<T_1 ， COMMIT>
<T_2 ， *B*, 14, 40>
<T_3 ， *A*, 20, 60>
<T_4 ， START>
<T_3 ， *D*, 8, 18>
<T_3 ， COMMIT>
<T_4 ， *C*, 16, 30>

图 -21　日志文件

小结

- 故障种类，和各种恢复措施以及检查点技术。