Lecture Notes

Recovery Algorithm

Date : 09/11/11

Submitted by:

Palash Goyal 08012310

Satbir Singh 08012321

A system is susceptible to crash which can be caused by various factors such as hardware failure, power failure, software errors, viruses etc. If there is an ongoing transaction, this leads to loss of information. Such situations are prevented by using some sort of recovery system which ensures that the system can be restored to the old state before the crash happened.

Types and severity of crashes based on causes:

Type 1: The crashes that are caused when storage goes down.

Type 2: The crashes which are caused when a deadlock is created or due some invalid I/O value.

The crashes of Type 1 are more severe than the crashes of Type 2.

Crash recovery from Type 1 crashes can be done using:

a. Store the data in multiple copies in the hard disk and the query is made to the backup data set rather than the local one.
b. Different type of storage device can be used. This method is more reliable than the one stated above but comes at the cost of slower transactions.

Lock-based recovery: A transaction basically has 3 stages which should be considered while doing lock files for system recovery namely:

Stage 1: *Transaction beginning*

Stage 2: *Writing the variables*

Stage 3: *Commit operation*

A lock file can be written either at the beginning or the end of above mentioned stages. The corresponding cases are discussed below:

Stage 1: **Transaction beginning**

a. If the lock file is written before the transaction begins and the system crashes after the transaction begins but before it ends, the system is restored to the point where transaction hasn't yet started. This case will have an overhead because if the system crashes just before the transaction was going to complete, it will have to do the whole process all over again.
b. If the lock file is written after the transaction begins, while system restore after a crash, it won't be known whether the transaction actually took place or the system crashed before the transaction was completed which makes the case of writing the lock file before more favorable.

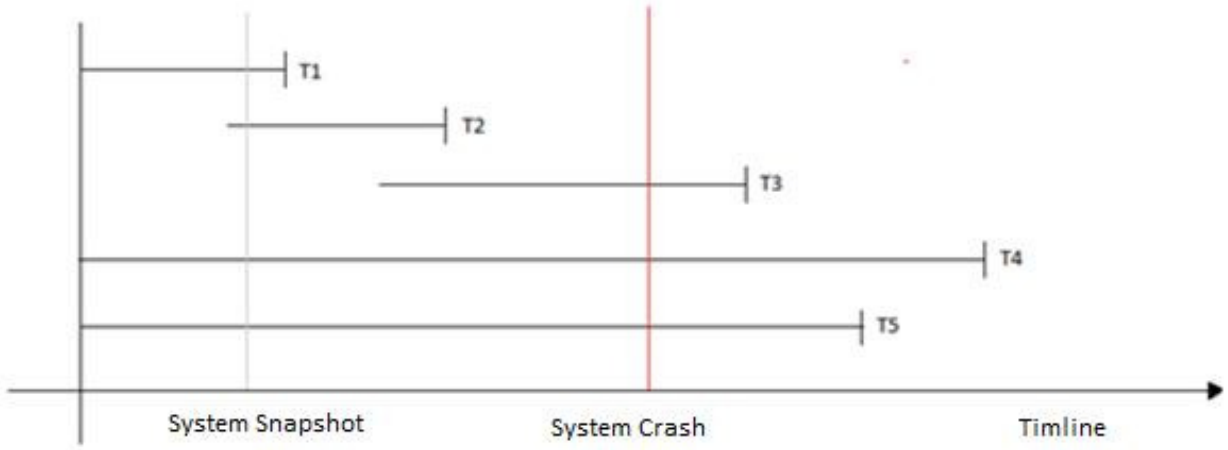
Stage 2: **Writing the variables**

a. If the lock files are written before writing the variables, then the system is restored to the point where the process of variable change has not yet started. This makes it more suitable because if the system crashes while writing to the variables, we will restore it to the point where it hasn't started and can be done again.
b. If the lock files are written after we started writing the variables and the system crashes during this process, there is no way of knowing whether the write to variables was complete.

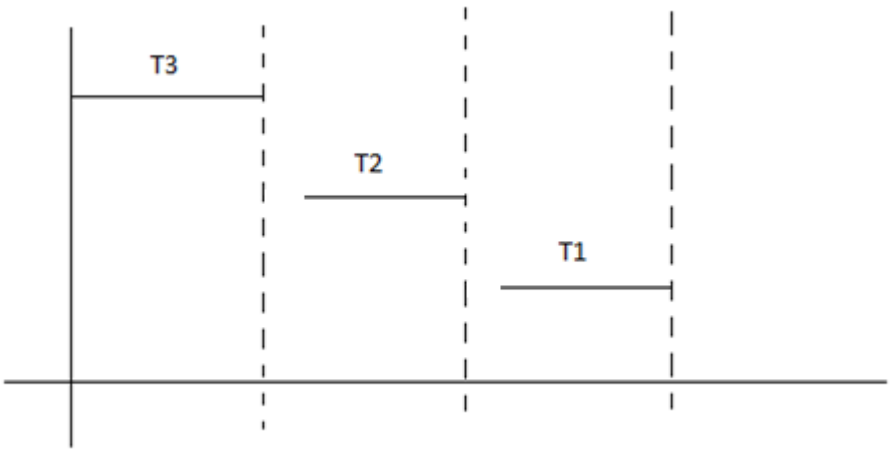
Stage 3: **Commit operation**

a. If the lock files are written before the commit operation and the system crashes before the commit actually took place, the system will assume that the commit was completed. So, this scenario is not adequate.
b. If the lock files are written after the commit operation takes place, the system is restored to the point where data has been actually saved into the disk making this case more favorable.

Consider the following scenario:



- No Dependency scenario: T1 and T2 are safe at crash because the snapshot was taken and both of them are independent of the transactions that are in progress.
- T2 is dependent upon T3: during crash recovery both T2 and T3 have to be undone.
- T1 is dependent on T2 and T2 is dependent upon T3: The right case is to start T2 after T3 is finished and T1 after T2 is done as given in next figure.



Note: Snapshots can be taken periodically and are helpful in recovery but more storage space is required and system might slow down depending on the frequency on which the snapshots are taken.