# Assignment 3

### 9.12.2014

### Radu Ionita, Nicolae Mariuta

**Question 1: Recovery Concepts**

1. *In a system implementing force and no-steal, is it necessary to implement a scheme for redo? What about a scheme for undo? Explain why.*

Implementing force approach means that in the case of a transaction commit, we need to ensure that all the changes made to objects in the buffer pool are forced to disk.

When the no-steal approach is used, in the event of an aborted transaction we are not required to undo the changes, mainly because the changes have not been written to disk. This shows that implementing a undo scheme is not needed.

Moreover if force approach is used all the changes are written to disk at commit time. The Redo scheme is therefore not necessary to be implemented.

1. *What is the difference between nonvolatile and stable storage? What types of failures are survived by each type of storage?*

Nonvolatile Storage data is not lost in the event of power shortage such as the volatile Storage, but it is lost in the event of a crash, total destruction of the hardware or random decay. An example of Nonvolatile Storage is the disk or DB files.

The Stable Storage example is the DB log that is online and DB backup that is offline.

The Stable Storage data is never lost. In case of data being lost from the disk, a corrupted file during a crash, random decay or in case of a permanent failure, if the files were in stable storage it means that a mirrored disks or a multitude of copies were used in different locations, making the data never lost.

1. *In a system that implements Write-Ahead Logging, which are the two situations in which the log tail must be forced to stable storage? Explain why log forces are necessary in these situations and argue why they are sufficient for durability*.

It is critical to make sure that all the log entries describing a change in the database are written to stable storage before the change is made. If this will not be done then when a system crashes just after the change we will have no record of the changes. The recovery manager is enabled by the log to undo the aborted actions and eventually uncompleted transactions and redo the actions of the committed transactions.

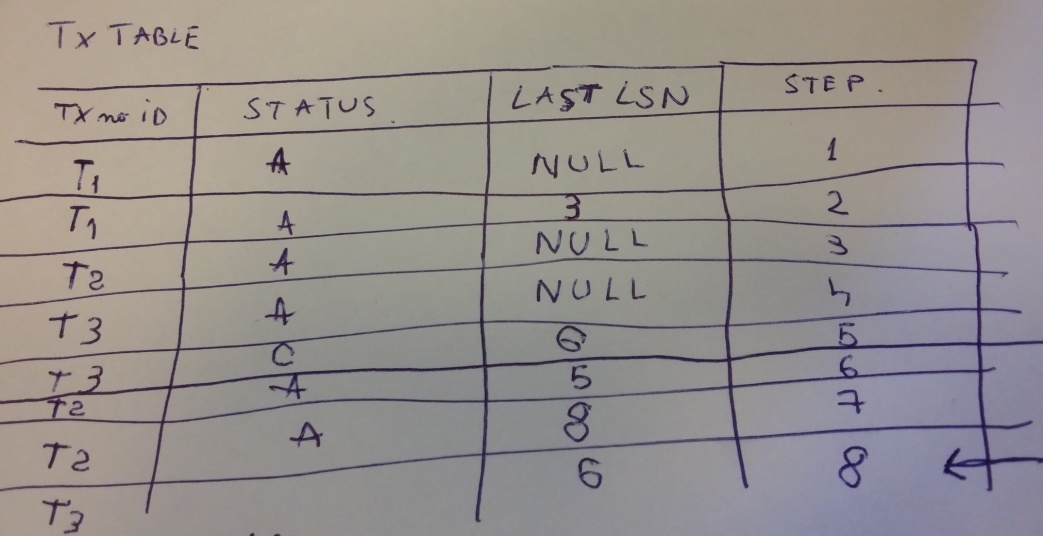
The log records every change to a DB object and the log tail must be forced to stable storage before any changes to the DB object are written to disk.

All writes to the log are sequential writes; the log is maintained as a sequential file. The cost of forcing the log tail is much smaller than the cost of writing all changed pages to disk.

They are sufficient for durability because maintaining one or more copies of the log on different disks and also on different locations makes the chance of all copies of the log to be lost very small.

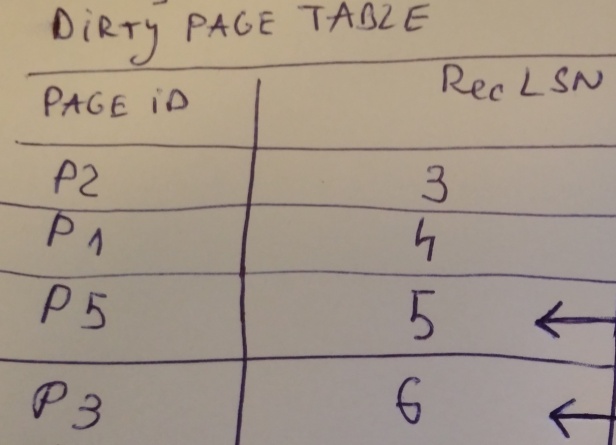
**Question 2: ARIES**

1. *Show the state of the transaction and dirty page tables after the analysis phase:*

**

*Picture 1. Tx Table*

T3 ended, so it has to be removed from the table. The steps column is a personal added column in order to show the table in one picture instead of removing and making a new table for update, commit and end.



Picture 2. Dirty Page Table

1. *Show the sets of winner and loser transactions:*

Based on the above pictures, the loser transactions are the ones that are still active at the time of the crash. The loser transactions are T1 and T2, while T3 managed to commit and end, becoming the winner transaction

1. *Show the values for the LSNs where the redo phase starts and where the undo phase ends:*

The REDO phase starts at the smallest recLSN in dirty page table after Analysis, based on the above picture it is **nr 3.** The UNDO phase starts at the oldest log rec. of Xact active at crash, and it ends at the first record that is **nr 3.**

1. *Show the set of log records that may cause pages to be rewritten during the redo phase:*

The log records that will be rewritten during the redo phase is the Dirty Page Table;

1. *Show the set of log records undone during the undo phase:*

All the transactions (T1 and T2) that made an update except the transaction that committed (T3), namely LSNnr: 9, 8, 5, 4;

1. *Show the contents of the log after the recovery procedure completes:*

## Questions for Discussion on the Performance Measurements

1. *Discuss in detail the setup you have created for your experiments. In particular, document your data generation procedures, hardware employed, measurement procedures (e.g., number of repetitions, statistics used such as average or deviation), and any other considerations you made. In the evaluation of this question, we will consider not only your thoroughness, but also whether you provide a brief justification/rationale for your decisions.*

For the *BookSetGenerator* we implemented the two functions as it follows:

* *sampleFromSetOfISBNs* at first we check if the number of ISBNs that are demanded is not greater than the total number of ISBNs and the whole list of ISBNs is return if it happens. After that we create a set of *Integers* for the ISBNs that will be returned. We used function in *Math* to generate random number for ISBN with maximum value same as the list size and add repeat the operation until the generated ISBN is not already in the list that has to be returned.
* *nextSetOfStockBooks* we created a *Set* of *StockBooks* which contains the books that will be returned and a *Set* of *Integers* which will contain the list of ISBNs for the books that have already been added into the *Set* that the function returns just to make sure that a second book with same ISBN is not added. All books have name “Book+ISBN” and author name “Author+ISBN” to make it easier to create the books. We also created that books to be randomly *EditorPick* or not by using *Random* for generating random Boolean. The initial number of books is randomly chosen between 1 and 10.
* *initializeBookStoreData* function from class *CertainWorkload* simply calls the function *nextSetOfStockBooks* for creating the initial set of books

For the workload interactions we implemented the functions:

* *runRareStockManagerInteraction* this method is implemented almost same as it is described in the assessment. We used the generator created previously to create a random set of new books to be added and create a second set of books where the books that are not already in the store and we return that list at the end
* *runFrequentStoreManagerInteraction* for this function we created a Comparator which sorts the books ascending by considering the number of copies that are available. This allowed us to sort the list of books in stock and take only the books with lowest number of copies.
* *runFrequentBookStoreInteraction* we implemente this method by calling the methods that are described in the assessment and called those functions from the *WorkloadConfiguration* as we did for all these 3 functions*.*

For the reporting of metrics we used parameters to count the successful frequent interactions, total frequent bookstore interactions and total time needed by each workload thread and then used these sums to calculate throughput, the goodput and the latency.

1. *Show and explain the plots for throughput and latency that you obtained. As described above, we expect two plots: one for throughput and one for latency. Each plot should include two curves: one for executions in the same address space, and one for executions across address spaces. Describe the trends observed and any other effects. Explain why you observe these trends and how much that matches your expectations.*
2. *How reliable are the metrics and the workloads for predicting the performance of the bookstore? Are the metrics well chosen? What additional metrics would you choose to demonstrate the performance of the book-store?*