Assignment 4
Group 5: Qi Lin, Xiaoying Li, Ziying Yu
5/25/2022

Question 1

The system uses shadow paging as a mechanism for ensuring action consistency of logical operators, so each file (Employee file, name file, sal file, Departments file, mname file, and dname file) is managed via a page table pointing to pages in the file.

The transaction T begins, creates a copy of the public page table, and maintains it as a private page table during its life. Initially, both the public and the transaction T's page table are identical.

The insertion of the Employee table causes insertion on page 7 of the Employee file, and insertion into the corresponding B-trees causes updates to pages 9 and 11 of the name file and pages 13, 15, and 17 of the sal file. Also, the update of the Departments table causes page 19 of the Departments file to be updated and an insertion of the record on page 21 of the mname file corresponding to the B-tree. The transaction does not update in place on stable storage, but rather creates a new copy of the page and makes its private page table point to the new page. All updates are performed on the copy of the page. So transaction T creates copies of all the public pages it made changes to, which are page 7 of the Employee file, pages 9 and 11 of the name file, pages 13, 15, and 17 of the sal file, page 19 of the Departments file, and page 21 of the mname file. And then it makes its private page table point to the new pages and performed all updates on copies of those pages.

Then the transaction T requests commit. To maximize concurrency and prevent different transactions from writing the same page, page level locks will be used. The private page table of T is copied atomically to be the public page table. First, the transaction T's page table for each updated file is copied to stable storage. While transferring the page table to the disk, the transaction T acquires a lock on the page table, copies only the pages it modified from its page table and the remainder from the original public page table, and then releases the lock on the page table. Second, the master directory that maintains pointers to the copies of the page tables of the files is copied to a new location. Finally, a pointer stored in the disk is then switched to atomically point to the new master page table.

Since the system uses logical logging with logical undo, it stores the name of the operator and corresponding parameters from which UNDO/REDO can be derived, also a single log will be written for each operator. The logs of transaction T's operations to be written are as follows:

<T begin>
<T, insert, {Jill, 120k}, Employee>
<T, update, {Jill, toy}, Departments>
<T commit>

Question 2

Transaction T begins. Since the system does physical logging for action consistency, to maximize concurrency, and to prevent freed space from being used by some other transaction, the transaction T acquires page level locks on all the public pages it made changes to, which are page 7 of the Employee file, pages 9 and 11 of the name file, pages 13, 15, and 17 of the sal file, page 19 of the Departments file, and page 21 of the mname file. Then it first does the operation of inserting {Jill, 120k} to the Employee table, which causes insertion on page 7 of the Employee file. And insertion into the corresponding B-trees causes updates to pages 9 and 11 of the name file and pages 13, 15, and 17 of the sal file. Thus, the transaction successfully completes the insert operation. Then it does the operation of updating the Departments table, which causes page 19 of the Departments file to be updated and an insertion of the record on page 21 of the mname file corresponding to the B-tree. But the transaction aborts during the execution of the update operation. It fails after inserting the tuple on the Departments file on page 19, and before the corresponding insertion on page 21 of the mname B-tree. So, the system needs to do undo operations, which reinstate the old state of all the records transaction T changed. When all undo operations are finished, page level locks on all the public pages transaction T made changes to will be released.

Since the system does physical logging for action consistency, it stores before and after images of objects. And the update part of each page will be written to the log, which is written before updating each data page since if the data page is partially updated, the log can be used to reinstate the before value in case of UNDO and after value in case of REDO. The logs of transaction T's operations to be written are as follows:

<T begin>
<T, Employee, b(page 7,O), a(page 7,O)>
<T, name, b(page 9,O), a(page 9,O)>
<T, name, b(page 11,O), a(page 11,O)>
<T, sal, b(page 13,O), a(page 13,O)>
<T, sal, b(page 15,O), a(page 15,O)>
<T, sal, b(page 17,O), a(page 17,O)>
<T, Departments, b(page 19,O), a(page 19,O)>
<T abort>
<T, Departments, b(page 19,O)>
<T, sal, b(page 17,O)>
<T, sal, b(page 15,O)>
<T, sal, b(page 13,O)>
<T, name, b(page 11,O)>
<T, name, b(page 9,O)>
<T, name, Employee, b(page 7,O)>

Question 3

1.

CLRs are very important for logical undo operations, and they help us to avoid some more complex system failure cases, such as system failures again when in recovery processing.

Suppose pageLSN is 20, and the log record that modified the page are three log LSN, 10, 12, and 20. Now we want to abort some operations on this page corresponding to LSN 12. If we reset the pageLSN of this page to the LSN of the log record before LSN 12, which is LSN 10 in this example, this pageLSN will no longer capture the effect of log 20 on this page. Therefore, this transaction rollback will go wrong. CLRs can help us avoid this situation. Such as CLR(12) means that the effect of log corresponding to LSN 12 has been undone, and sets the pageLSN of this page to the LSN of this CLR.

2.

CLRs ensure the logical undo operations, if we want to replace the CLRs, we should ensure that transaction rollback and restart algorithms can support physical undo operations, which means we need to execute the redo and undo operations physically. This is physical logging. In physical logging, the rollback and restart algorithms can work correctly without CLRs under the assumption of page level locking.

3.

If we perform the selective redo operations, the pageLSN is no longer a true indicator of the current state of the page.

Suppose the pageLSN is 10 before the failure, this page is modified first by the transaction T1 at LSN 10, then subsequently modified by transaction T2 at LSN 20. In the end, this page is modified by transaction T1 at LSN 30. Now, the system crashed, and we need to restart, here we suppose T1 commits and T2 aborts. In selective redos, we should redo the transaction T1, Redo operations will update at LSN 30. Undo operations will try to undo LSN 20, although this LSN 20 update is not on this page, since T2 has been aborted, we don't need to redo the T2 at LSN 20. This attempt to undo the effect of T2 will go wrong since it is not present on the page, the pageLSN is no longer a true indicator of the state of the page.

Reference link: https://www.cs.umd.edu/class/spring2015/cmsc724/acid.pdf

Question 4

1.

The modification here is that we do not have to only release locks held by a transaction after the transaction commits. Similar to operation logging, we create an operation-begin when an operation starts, and operation-end when an operation completes. However, rather than release locks held by a transaction after it commits, the lock can be released once the operation completes, that is after the operation-end. In this way, we save the time overhead in the group commit for releasing all the locks.

Operation logging is done as follows:

1) When operation starts, log <Ti, Oj, operation-begin>. Here Oj is a unique identifier of the operation instance.
2) While the operation is executing, normal log records with physical redo and physical undo information are logged.
3) When the operation completes, <Ti, Oj, operation-end, U> is logged, where U contains information needed to perform a logical undo information.

If crash/rollback occurs before the operation completes, the operation-end log record is not found, and the physical undo information is used to undo the operation.

If crash/rollback occurs after the operation completes, the operation-end log record is found, and in this case, logical undo is performed using U; the physical undo information for the operation is ignored.

Since we use the operation-begin and operation-end in the log for operations, we can still perform the recovery process correctly. If we see an operation-end, we can use U to do the logical undo operations for the abort operation, and then write an operation abort log. If we cannot see an operation-end, we should do a physical undo operation based on the normal log records.

Thus, the modification works correctly with the early lock release which is used in conjunction with group commit as we described above.

2.

Inspired by distributed 2PC, we need to first design how to synchronize with other sites about group commits in several rounds (request-to-prepare, prepared, commit, and done). Each site has an independent log. When the coordinator wants to write a group commit log, it sends a REQUEST-TO-PREPARE message to cohorts. If a cohortite ‘s log record is consistent with the coordinator, it then writes a PREPARED log record before responding back with a PREPARED message. Else, the cohort responds back with a NO. After the coordinator has received all the PREPARED messages from cohorts, it then can write into the commit log, and mark cohorts as PREPARED. If all cohorts are PREPARED, the coordinator then changes the status to COMMITTED and sends COMMIT messages to other cohorts. When the cohort receives a COMMIT message from the coordinator, it starts to write its own commit log and sends ACK to the coordinator. On receipt of ACK from the cohort, the coordinator marks the cohort as ACKED.

The cohort performs the following actions to recover. If the log does not contain a completion log record, it acquires all locks for the group transaction and asks the coordinator for the status of the group transaction. As for the coordinator, if it has no information about the transaction in memory, it returns an ABORT message. If it has information and group transactions committed, it sends back a COMMIT. Else, if it has information but the transaction is not yet committed, it sends back a WAIT. The cohort will execute abort, commit, and wait for operations based on the response. If the log contains a completion log record, just do nothing.

Coordinate performs the following actions to recover. If the log does not contain a commit. Simply abort the group transaction. If a cohort asks for status in the future, its status is not in the database, and it will be considered aborted. If there is a commit log record, but no completion log record, the coordinator recreates group transactions entry committed in the database and the recovery process will ask all the cohort if they are still waiting for a commit message. If no one is waiting, the completion entry will be written. If the commit log record and completion log record, for the group commit, the coordinator just does nothing.

When a system crashes, we can recover in the following cases. If the coordinator crashes before writing a commit log for these group transactions, it cannot commit these transactions. So, the current group transactions should be aborted since the coordinator hasn't successfully written a commit log for these transactions. During a recovery process, we should undo these transactions. If the coordinator crashes after writing the commit log of these group transactions, which means all cohorts have reached a consensus. Every cohort wrote the commit log of the group transactions and committed them successfully. In this case, we do not have to undo the current group transactions because the corresponding effect has been reflected on the disk of the coordinator and cohort

Question 5

1.

The bound is the LSN before Ci.lsn1. Ci.lsn1 is the LSN of the last checkpoint starts, since log records before that LSN have their updates reflected in the database on disk and need not be redone.

2.

The bound is the LSN before min (Ti.lsn)(i=1,2, .., n), since these transactions were active at the time of failure, we need to undo the effect of these transactions, once we finished undo work for these transactions, which means we roll back to the LSN of the first log record for these active transactions, the undo work will stop at that LSN.

1.

No. (If the checkpoint cannot provide any active transaction information at that time)

Suppose there is a transaction T1 that starts before the checkpoint but has no update log record after the checkpoint. This transaction was active prior to the system failure, however, it will not present in the active list at the end of the analysis pass, since this restart algorithm only adds the transaction to the active list when sees an update log record for the transaction, this transaction keeps silence after the checkpoint, therefore, there is no update log record for this transaction.

Yes. (If the checkpoint can provide any active transaction information at that time)

The question is not very clear, here we want to analyze another case, we assume that the checkpoint provides information for active transactions at that time, which means the restart algorithm contains the necessary information about these active transactions that start before the checkpoint and thus add them into the active list.

In this case, every transaction that was active prior to the system failure was present in the active list at the end of the analysis pass. Here are two cases for this argument. The transaction starts before and after the checkpoint.

Suppose T1 starts before the checkpoint, it will add to the active list based on the information which is provided by the checkpoint. Since this transaction was active prior to the failure, this transaction has no commit or abort log record after the checkpoint, it will present in the active list at the end of the analysis pass.

Suppose T2 starts after the checkpoint, it will be added to the active list when the restart algorithm sees the update log record, that is the start log record. Since this transaction was active prior to the failure, this transaction has no commit or abort log record after the start log record, it will be present in the active list at the end of the analysis pass. Thus, the answer is Yes.

2.

Yes.

In this case, every transaction that had been committed prior to the system failure but whose effects may need to be redone using log records present in the commit list at the end of the analysis pass. Since we need to redo the effect of the transaction, the commit log record for the transaction must happen after the checkpoint, otherwise, the effect of the transaction is reflected in the database on disk and need not be redone. Therefore, the restart algorithm can see this commit log record, and the algorithm can successfully insert this transaction into the commit list and delete the transaction from the active list (in case the transaction is present in the active list).

Question 6

1.

The paper covers multiple recovery techniques by first introducing the Write-Ahead Logging (WAL) protocol. WAL provides atomicity and durability by in-place updating. It is widely implemented in most database systems for recovery purposes. It then briefly discusses Algorithms for Recovery and Isolation Exploiting Semantics (ARIES), which is based on WAL. It not only supports WAL but many other features, such as steal and no-force buffer approaches, etc. Some novelty of ARIES is its extension, including ARIES/NT, ARIES-RRH, ARIES/IM, ARIES/LHS, and ARIES/CSA. Despite the fact that ARIES is a well-known recovery method, recent technology advancements have prompted the creation of new software designs that can better leverage contemporary hardware. Thus, the paper introduces a usage of log-structured files and write-optimized B-tree in the recovery process. The novelty of single-page repair which uses a B-tree mapping is that it handles individual pages' recovery instead of the whole media device's recovery.

MMDB durability and recovery are generally like disk-based database systems, but they are different in several ways. The way MMDB recovers from a system failure is different from the disk-resident database. Most MMDBs perform redo-only logging and do not undo transaction updates. A logical transaction-level logging and an asynchronous transaction-consistent checkpoint are also commonly performed in most MMDBs, the novelty of them compared to ARIES recovery is that they are very lightweight for transaction processing.

Disk and main memory-resident DBMSs use data propagation to secondary memory to guarantee durability and duplication strategies to ensure recovery. However, DBMs running in NVM perform Write-behind Logging (WBL) for recovery and Dirty Tuple Table(DTT) to track transaction updates. The novelty of WBL is that it recovers from system failures almost instantaneously since it stores in the log the location of updated databases instead of how they were updated. Another novelty of WBL is that it does not require periodic checkpoint generation to speed up recovery because any transaction modifications made prior to the current group commit interval have already been saved.

At last, the paper focused on different recovery techniques for some MMDB other than relational MMDB. Hekaton is an SQL Server's in-memory engine. It uses checkpoints to ensure recovery after a failure. The Hekaton recovery method locates the most recent checkpoint file, which monitors all data and delta files, after a failure. VoltDB is an OLTP MMDB. The novelty of its checkpoint to recover failure is that it is asynchronous transaction-consistent. The recovery schema works even if the site topology changes. HyPer can manage both OLTP and OLAP workloads in the same database. Following a failure, the recovery procedure begins by copying the most recent fully written snapshot archive from secondary storage into main memory. The log is then replayed from the checkpoint record onward. SAP HANA loads the savepoint(snapshot) and performs REDO to recovery. The novelty of SiloR is it adds a concurrent recovery schema in Silo, it performs a fuzzy checkpoint. TimesTen is an OLTP in-memory database system. FineLine performs an instant recovery schema after a system failure.

In the end, the paper briefly talked about two recovery techniques, PACMAN, and Adaptive Logging. PACMAN is a recovery mechanism for transaction-level logging. It aims to parallelize the recovery and reduce transaction processing runtime overhead. Using a cost model, the adaptive logging technique dynamically detects bottlenecks and uses the same amount of I/Os as ARIES logging if all transactions are identified as bottlenecks.

2.

The paper has several comparisons regarding different databases. It first mainly compares the current popular MMDB with the traditional Disk-based database, as illustrated in the following:

databases / metrics	MMDB	Disk-based Database (MySQL, Oracle…)
data storage	physical pointers indexing for memory and cache utilization	buffer pool indirection secondary memory access
concurrency control	MVCC (optimistic / pessimist)	PCC
query processing	compiles queries directly to machine code logical transaction-level logging asynchronous transaction-consistent checkpoint	Volcano-style processing ARIES-style
other	higher throughput prefer logical logging (store fewer items on a log) multi-versioning of data	perform physical logging in-place updating

As the paper goes on, it then compares some different or the same kind of databases by specific metrics, such as the following comparison:

SQL Server	Hekaton
experiment (comparison strategy) in running typical customer workload	
MMDB (H-Store, VoltBD, and Calvin)	Hekaton, SAP HANA, MemSQL, and Oracle TimesTen
partition database	No-partitioned in-memory system
Within DBMs	
NVM-Resident Database Systems	disk and main memory resident DBMSs
Write-behind Logging (WBL)	data propagation to secondary memory + duplication

At last, the paper mainly compares some MMDBs' recovery other than other relational MMDB recovery. It uses a table, Table 1, as a comparison strategy to compare the recovery approaches above, which also includes important metrics such as concurrent control, logging, checkpoint, and instant recovery. As also compared in the above question.

Other metrics can be used to compare between databases such as message complexity, the level of code realization difficulty, fault-tolerance level, contention level, scabillity, etc.

3.

I really like the design of section 2, Database Recovery Overview. It focuses on disk-resident systems to convey the core ideas of DBMS recovery, which gives a suitable theoretical foundation for the following parts. It also offers readers a choice of whether to skip this section or not, depending on their awareness.

There are things that I wish to cover rather than certainly did not like something in the paper. As more metrics are used to compare databases mentioned in the above question. It would compose a limitation or future work part for different databases as reviewing and presenting recovery techniques. For example, what are some limitations of any database or recovery mechanism, or what future work people are working on them right now.