Observations AND PROBLEM:

In the proposed speculative locking protocols, the waiting transaction is allowed to access the locked data object whenever the lock-holding transaction produces corresponding after-images during execution. By accessing both before and after images, the waiting transaction carries out speculative transactions and retains one execution based on the termination decisions of the preceding transactions. This paper extends speculation to 2PL to improve the transaction processing performance of Distributed DB Systems. Under naive SL, the number of speculative transactions drastically increases with the amount of data. In a database system, a transaction is more likely to commit than abort. The paper proposes SL(1) and SL(2) that process transactions efficiently by reducing the number of speculations. Even with manageable extra resources, SL improves the performance over 2PL in DDBS where processing takes longer and aborts occur frequently.

Also, time to commit in a WAN vs LAN is much greater. This means that 2PL has lower performance in these conditions. Thus the introduction of speculative locking under distributed database systems will bring about huge increases in performance levels.

Performance and Implementation Issues

Performance

- Even the transaction produces multiple data object versions only one is force-written. Thus there is no extra disk write cost.
- The logging process is delayed until the transaction confirms the single version. In case of a failure all these data objects in main memory may be lost. This overhead is negligible.During execution, after images are sent to object sites whenever the transaction produces them. This overhead can be reduced by 1) piggybacking them on 2PC prepare messages 2)immediate transfer only for remote hotspot data objects.

Implementation

- To implement the locks under SL, the system should know when the transaction completes the work with the data object. Putting lock conversion markers for each data object from EW to SPW is not difficult.Lock management needs to be changed to recognise three kinds of locks: R, EW, SPW.
- Object tree and speculative execution management: for both, the existing implementation can be extended to corresponding trees and multiple executions.

- Index modification: there is minimal change to the existing index implementations.Commit processing: There are two ways to implement 2PC. Based on the type of implementation we can either dynamically set the timeout periods in the existing 2PC implementations based on the number of transactions in the depend_set or we can directly adopt 2PC under 2PL without modifications.

Key contributions and description:

Speculative locking for DDBSs

System model and assumptions

A database system consists of a set of data objects. A data object is the smallest accessible unit of data. Each data object is stored at one site only. Database sites are connected by a computer network. Each site supports a transaction manager and a data manager.The transaction manager supervises transaction processing and data managers manage individual databases.

A transaction is a set of atomic operations, either a read or a write, on data objects. When two transactions are not ordered relative to each other, they can be executed in any order. However, a read and write on the same element must be ordered. The read and write sets contain the objects to be locked by the transaction.

A transaction copies data objects through read operations into private working space and issues a series of updates.

This paper employs centralised 2PC although any variant of 2PC or 3PC can be used.

Lock compatibility

In the SL approach, the W-lock is partitioned into the execution-write (EW) lock and the speculative-write (SPW) lock. Transactions request only R and EW-locks. A transaction requests the R-lock to read and the EW-lock to read and write. Consider that the transaction obtains EW-lock and later produces the after-images of the data object.

The EW-lock is changed to the SPW-lock after including the after-images in the respective data object tree. The paper assumes no lock conversion from R to EW-lock.

Under SL, only one transaction holds an EW-lock on the data object at any point in time. However, multiple transactions can hold the R and SPW-locks simultaneously.

SL ensures consistency by forming a commit dependency among transactions. If Ti forms a commit dependency with Tj , Ti is committed only after the termination of Tj. The commit dependency rules are as follows: 1) If Ti obtains the EW-lock while Tj is holding either an R-lock or an SPW-lock on the data object, Ti forms a commit dependency with Tj , and 2) if Ti obtains the R-lock while Tj is holding an SPW-lock on the data object, Ti forms a commit dependency with Tj.

SL(r)

If a transaction conflicts with n transactions, it is robust against r transaction aborts. Every data object X is organised as a tree. Locks are requested dynamically one at a time.

Lock acquisition

- The home site of transaction Ti sends lock request of Ti to X that resides at Sk.
- If the lock request is in conflict with previous lock request that is waiting for X, the depend_set is initialised to empty and the tuple <depend_set, waiting> is appended to queueX.
- When the previous lock request acquires an R/SPW lock, i.e. status not waiting, the R/EW lock is granted to the new lock request. If Ti forms a commit dependency with Tj on X then depend_set is updated with Tj. Status of new lock request is changed to not waiting. Both treeX and depend_set are sent with a lock reply message to the home site of Ti.

Execution :- At the home site, on receiving the lock reply, processing is carried out according to the given algorithm. When Ti issues a write operation on X, for all speculative executions the set depend_set is copied to depend_set(Xp) where Xp is an after image value of X produced by the speculation. All after images with their depend_sets are sent to the object site. Each after image value along with its depend set is included as a child to the corresponding before image node of treeX.

2PC processing

- The coordinator sends PREPARE messages to the participants. If transactions have not been completed or are aborted, then they send VOTE_ABORT, otherwise VOTE_COMMIT.
- When the coordinator gets a VOTE_COMMIT from all participants, it selects the speculation to be performed and sends the identity of

after images with a GLOBAL_COMMIT to all participants, otherwise it sends a GLOBAL_ABORT.

- When a participant receives a GLOBAL_COMMIT, the tree of the data oject is replaced by the subtree with the received after image as the root node.
- If Ti is an element of the depend_set, then it is deleted. If Ti is not in depend_set of any speculation also it is dropped.
- If any Tw accesses the data object that was accessed by Ti at Sk, then Ti is deleted from the lock request.
- When a participant receives a GLOBAL_ABORT, the afterimages included by the transaction are deleted along with the subtrees.
- Ti is deleted from depend_set is Ti is in Tw depend_set and depend_abort is incremented.If depend_abort>r, abort the transaction Tw. Otherwise for all speculations, if Ti is in depend_set Tw then Ti is dropped.
- If any Tw accesses the data object that was accessed by Ti at Sk, then Ti is deleted from the lock request.

Deadlocks :- They are removed using a dependency graph. If a cycle is found then the transaction causing the cycle is removed.

SL variants processing

SL(n)

- extreme pessimistic
- Robust against n transaction-aborts
- When Ti conflicts with n transactions there are 2^n termination possibilities. Thus the total number of speculation executions carried out is 2^n.
- For a transaction, if p is the probability of commit and q for abort, then the probability of k transactions committing and n-k aborting is nCkp^kq^n-k.
- Under SL(n), since it carries out all the possibilities, there is no chance of aborting.
- SL(0):- Extreme optimistic. It is robust against zero transaction aborts, because it does only one execution based on the after images. Given p, the probability of commit, then the probability that at least one transaction aborts is 1-p^n. Unless p is very close to one, its performance decreases with the amount of data assertions.
- SL(1) :-In a database system, since a submitted transaction is more likely to commit than to abort, we introduce SL(1) and SL(2) by

processing transactions with moderate robustness against cascading aborts. Each transaction is robust against one transaction abort. Thus only n+1 speculation executions need to be supported. The probability that n-1 transactions commit is nCn-1 p^n-1q. The probability that at least n-1 transactions commit is p^n + np^n-1q. The probability of abort when k preceding transactions abort is 1 - p^n – np^n-1(1-p).

- SL(2) :- A transaction is robust against two transaction aborts. It is thus sufficient to support Σn + 1 executions for a transaction. Given the numerical value of the probability that at least n-2 transactions commit and the probability of abort when k preceding transactions abort, SL(2) can improve performance under high abortive situations.
- SL(3), SL(4) variants have too many executions to be viable in high abort conditions.

Multiple conflicts

Object dependency between the data objects accessed by the transaction affects the number of speculative executions of the subsequent transactions. If the object dependency exists between any two objects accessed by the transaction, the subsequent transaction may form indirect dependencies. If object dependency does not exist among the data objects accessed by a transaction, a transaction carries out 2^n, n + 1, and Σn+ 1 executions in case of SL(n), SL(1), and SL(2), respectively. However, if object dependency exists among the data objects accessed by the transaction, the SL protocols have to carry out additional executions to cover indirect dependencies. This paper carries out simulation experiments assuming that object dependency exists among all the data objects accessed by a transaction.

Extension under limited resources

Since each speculative execution need a separate workspace, the size of main memory limits the number of executions. Assuming that all transactions require the same amount of main memory and all of the speculative transactions require the same amount of main memory, based on the active transactions and the number of memory units available at a site, we can calculate executions_limit. A simulation study under a wide variety of transaction workloads indicate that even manageable extra resources both SL(1) and SL(2) significantly improve the performance

over 2PL in the DBMS environments where a transaction spends longer in processing and transaction aborts occur frequently.