Terminology:

read-only transactions (ROT): transactions that only read data record

UT: update transactions

2PL: two phase locking

2PC: two phase commit in distributed database system

SI-based: Snapshot-Isolation based

SL-based: Speculation-based

Purpose of the Protocol:

In reference [1], the SL-based protocol is presented and a way to accommodate it to distributed database management system is proposed by Mohit Goyal, T. Ragunathan and P. Krishna Reddy. The proposal aims at providing a solution that could improve the performance and accuracy when executing read-only transactions in DDBMS. The authors choose to tackle only ROTs because of the large portion of information access requests in modern DDBMS. They suggest that current existing methods have drawbacks. 2PL’s performance is not ideal with data contention while SI-based protocol sacrifice accuracy for better performance. The following protocols are a paraphrase of protocols presented in reference [1].

Basic SL-based ROT protocol:

Synchronous SL-based protocol for ROTs:

In this protocol, instead of usual R and W locks in 2PL, there are four different type of locks. The lock compatibility matrix is shown in Figure 2. When a ROT wants to read a data record, it requests an RR lock on the object. When a UT wants to wants to read a data record, it requests an RU lock on the object. Executive write(EW) lock and speculative write(SPW) lock are for writing. When a UT wants to write a data object x, it first requests an EW lock of that object. After the write is done, the EW lock is converted into SPW lock. The SPW lock is hold by UT until it commits/aborts.

When a ROT have a conflict with a UT, that is, the ROT requests a RR lock on an data object that has its EW lock or SPW lock hold by an UT, it should follows the following steps:

1. If the UT holds the EW lock, ROT wait for the lock to be converted to SPW lock.
2. If the UT holds the SPW lock, ROT accesses both the before image and after image of the data object.
3. ROT synchronously carries out speculative executions based on both the before image and the after image of the data object respectively.
4. When the ROT commit, the choice of appropriate speculative execution depends on the state of conflicting UT. If the UT have committed successfully, retain the speculative execution based on the after image of the data object. If the UT is active or have aborted, retain the speculative execution based on the before image of the data object.

Asynchronous SL-based protocol for ROTs:

The locks used with ASLR are the same as SSLR. The lock compatibility matrix is shown in Figure 3. We can see that in ASLR, difference from in SSLR, RR lock are compatible with both EW lock and SPW lock. So in ASLR, when a ROT have a conflict with a UT, it should follows the following steps:

1. ROT accesses the before image of data object and carries out the corresponding speculative execution
2. When UT’s EW lock on the data object is converted to a SPW lock, ROT accesses the after image of data object and carries out the corresponding speculative execution
3. When one of the speculative executions finished, check whether any of the completed speculative executions is valid depending on the state of the conflicting UT. If it does, ROT retain this speculative execution, abort all other speculative executions and commit. Else, keep going.

Comparison between SSLR and ASLR with examples:

The examples in Figure 4 and Figure 5 demonstrate different behaviours between SSLR and ASLR under the same situation. In the examples, T1 and T3 are UTs and T2 is ROT. T21 and T22 are speculative executions of T2. Obviously, T21 is retained as the appropriate speculative execution in both examples.

Extend SL-based ROT protocol to DDBMS:

In the DDBMS scenario, the processing of transaction is divided into two phases, execution phase and commit phase.

Protocol for UTs:

Execution phase: Distributed 2PL is used with the modification of locks as we described in the previous section. Also, whenever after-image is produced, it is communicated to the object site and target sites where ROTs are waiting.

Protocols for ROTs:  
Like basic SL-based protocols, there are two different protocols for ROTs, the distributed synchronous speculative locking protocol for ROTs (DSSLR) and the distributed asynchronous speculative locking protocol for ROTs (DASLR). In both DSSLR and DASLR, two additional data structures are added. A “dependent\_set” is maintained for each ROT, which contains IDs of the conflicting UTs. A “dependent\_list” is maintained for each speculative execution, which contains IDs of UTs from which it has used an after-image when performing the speculative execution.

DSSLR:

Execution phase steps:

1. ROT send lock requests to object sites. For each requests, do step 2 to 4.
2. If the lock request conflict with UTs, ROT wait for the after-image of the data object to be produced.
3. Both before image and after image of the data object are sent to ROT’s home site.
4. ROT synchronously carries out speculative executions based on both the before image and after image of the data object. Add UT’s ID to ROT’s “dependent\_set”. Add UT’s ID to “dependent\_list” of speculative executions which used the after image.
5. When all the requested locks are granted, ROT complete the executions and turn into commit phase.

Commit phase steps:

1. When all the speculative executions complete, ROT asks the home sites of the UTs in the “dependent\_set” for their status to see whether they are committed or aborted/active
2. Base on the statuses of UTs, ROT select an appropriate speculative execution using “dependent\_list”.

DASLR:

Execution phase steps:

1. ROT send lock requests to object sites. For each requests, do step 2 to 4.
2. If the lock request conflict with UTs, ROT gets the before-image of the data object and carries out corresponding speculative execution. Add UT’s ID to ROT’s “dependent\_set”.
3. When after image is ready, it will be sent to ROT. ROT carries out additional speculative executions based on the after image of the data object. Add UT’s ID to “dependent\_list” of speculative executions which used the after image.
4. When all the requested locks are granted, ROT complete the executions and turn into commit phase.

Commit phase steps:

1. Whenever one of the speculative executions complete, ROT asks the home sites of the UTs in the “dependent\_set” for their status to see whether they are committed or aborted/active.
2. Base on the statuses of UTs, ROT decide if this is an appropriate speculative execution using “dependent\_list”. If it is, the transaction commit. If it is not, wait for the next speculative execution to complete.

Commentary on this solution:

This proposal is important because it improve the performance of the ROTs while not sacrificing the accuracy of data. It does so by interpret all the possible outcomes of ROTs with conflicting UTs and choose the appropriate one at the time of commit. It reduces the amount of time a ROT needs to wait for UT and use that time to precompute the results. The improvement of performance can also be observed on the experiment results in the paper shown in Figure 6. However, in this approach, the CPU usage would be very inefficient if there are multiple conflicting UTs for a ROT. The number of speculative executions will be growing exponentially. Another problem is that this protocol won’t work with replicated database, it is designed to work with DDBMS with fragment structure, so in the future, this could be a direction to develop. In addition, there is a lot of messages communicated in the process. Since the latency in network is not negligible, it is time-consuming to exchange these messages. It would be beneficial to optimize the number of messages need during the execution process.

Abstract:

Introduction:

This surveyed paper is aimed at audiences with basic understanding of DBMS and are interested in the concurrency control problem. To understand this paper, one need to know background knowledges including how basic concurrency control algorithms, such as Two-Phase Locking (2PL) and Optimistic Concurrency Control (OCC), works.

Unlike in Centralized Database Management Systems, in Distributed Database Management Systems, the storage devices are not attached to a common processor, instead they form an interconnected network while appeared as a unit to users. Some basic structure of distributing data are Fragmentation or Replication. Notice that the storage devices in a DDBMS don’t have to locate in the same physical location.

There are many advantages in implementing a distributed Database, for instance: protection of valuable data, performance improvement, higher reliability, easier to scale, etc. Transparency is one of the major properties of a distributed database management system. Users of the distributed system are not required to aware the complexities of implementing the distributed database management system.

We choose to discuss concurrency control in distributed database management system because we believe it is an important area in database. With better concurrency control algorithms, we could encourage more parallelism in execution and increase the throughput of databases. As DDBMS are widely used nowadays, any improvement of performance on the concurrency control protocols in DDBMS could bring large impacts. Some examples of popular distributed database include MongoDB, Hadoop and more. And idea of distributed ledger in Blockchain is an implementation of DDBMS.

Although distributed database management systems are widely used, we are still facing many difficult challenges in designing them and implementing them. Concurrency control is one of the major problems that a distributed database management system environment has. Most of these issues only exist in distributed database environment, but not in a centralized database environment, so we need specialized algorithms to tackle these problems. For example: Failure of Individual Sites, Failure of Communication Links, Distributed Deadlock, Distributed commit, Dealing with multiple copies of the data items, sync between different sites.

In our surveyed paper, we discuss three different approaches proposed in three papers to solve various problems in the concurrency control in DDBMS: Speculation-Based Protocols, Adaptive and Speculative Concurrency Control, Efficient Concurrency control mechanism. The first two algorithms focus on improving the throughput and performance in DDBMS with Fragmentation design. The third one focus on solving the synchronization problem among different storage devices.

The classification schema of our surveyed paper is shown in figure 1.