Extending Speculation for Improving the Performance of Read-only Transactions**

T. Ragunathan
Center for Data Engineering
International Institute of Information
Technology, Hyderabad, India
+91-40-23002314
ragunathan@students.iiit.ac.in

ABSTRACT

This paper presents an ongoing Ph.D. thesis work which aims at improving the performance of read-only transactions (ROTs) in database systems using the notion of speculation. In the literature, speculative locking approach has been proposed to improve the transaction processing performance in online transaction processing environments. In this thesis, we have proposed two protocols to improve the performance of ROTs, by making appropriate modifications to the existing speculative locking protocol. The proposed protocols process ROTs without any data currency and correctness issues. The simulation results show that the proposed protocols improve the throughput performance significantly over two-phase locking (2PL) and snapshot isolation (SI)-based approaches with manageable extra processing resources.

Advisor: P. Krishna Reddy

1. INTRODUCTION

In the emerging web databases and e-commerce scenario, information systems should meet intensive information requirements from a large number of users. The information systems frequently process read-only transactions (ROTs) or queries. In such systems, the ROTs should be processed with acceptable response time without any correctness and data currency issues. Research efforts are being made in the literature to investigate the approaches to improve the performance of ROTs. As a part of Ph.D. thesis work, we are addressing this problem and proposing speculation-based protocols to improve the performance of ROTs.

A read-only transaction (ROT) does not modify any data. The main issues in processing ROTs are correctness (serializability), data currency and performance. The widely used two-phase locking (2PL) protocol [1][2] processes ROTs with serializability as correctness criteria. However, it performs poorly as data contention increases due to increased waiting. In the literature, there are efforts to improve the performance by processing ROTs with a multi-version based approach [11], at lower isolation levels [3] and by proposing separate protocols for ROTs and update transactions [10][11]. Snapshot Isolation (SI)–based methods [3] are widely used to process ROTs. Even though, SI-based approaches improve performance, they compromise on the aspects of both data currency and correctness (serializability).

** Selected for EDBT Ph.D. Workshop 2008, March 25, Nantes, France

We briefly explain about data currency. The aspect of data currency is discussed for a data warehousing environment in [6] and for a replicated environment in [19]. The term "data currency" refers to how current or up-to-date system can guarantee a data object to be, for a transaction. Based on this, we define data currency for DBMS environment as follows. Let T_i and "t" denote a transaction and time duration, respectively. *The data currency of the data object provided to* T_i *is the value of "t" which is the time difference between the commit time of the transaction which created the latest version of the data object and the commit time of the transaction which created the version of that data object that was read by* T_i. If "t" is less/more, it means that transactions are provided with high/low data currency.

In the literature, a speculative locking [SL] protocol [7] is proposed to improve the transaction processing performance in distributed database systems. In SL, a transaction carries out multiple executions by accessing the uncommitted values produced by the preceding transactions. The SL protocol is proposed to improve the transaction processing performance of OLTP environment by considering transactions which contain both the read and write operations. Through SL, the performance can be improved by trading extra processing resources without violating serializability criteria.

As a part of Ph.D. thesis, we are making efforts to develop speculation-based protocols to improve the performance of ROTs in database environment which processes both the ROTs and update transactions (UTs). We have proposed speculative protocols to improve the performance of ROTs by making appropriate modifications and extensions to SL through identifying features specific to ROT processing environments. As a result, there is an opportunity to improve the performance by processing ROTs with few speculative executions as compared to SL [7]. The proposed modifications result in two protocols for ROTs. One is synchronous speculative locking protocol [8] and another is asynchronous speculative locking protocol [20].

Using the proposed protocols, ROTs can be processed with high performance and without any data currency and correctness issues. The simulation results under limited resource environments show that these protocols improve the performance significantly over the other approaches including 2PL and SI-based approaches by adding a fraction (0.2 times) of additional resources.

1.1 System Model

A Transaction is a particular execution of program that manipulates the database by means of read and writes operations [17]. A transaction can read a set of data objects from the database which forms the read-set (RS) of the transaction and modify the values of another set of data objects which forms the write-set (WS) of the transaction. The transactions T_i and T_j are said to have a conflict, if $RS(T_i) \cap WS(T_j) \neq \emptyset$, or $WS(T_i) \cap RS(T_j) \neq \emptyset$ or $WS(T_i) \cap WS(T_j) \neq \emptyset$. An ROT does not contain write operations and a UT includes both read and writes operations. The database management systems support components like transaction manager and data manager [17]. The transaction manager supervises the processing of transactions, while the data manager manages the individual databases.

We explain here some notations. Data objects are denoted with 'x','y', … Transactions are represented with T_i, T_j,… . For the data object 'x', 'x_i' (i = 0 to n) represents i^{th} version of 'x'. The notation $r_i[x_j]$ indicates that read operation is executed on 'x_j' and $w_i[x_j]$ denotes that write operation is executed on a particular version of 'x' and as a result 'x_j' is produced. The notations, 's', 'c' and 'a' depict the start, commit and abort of transactions. T_{ij} indicates j^{th} speculative execution of T_i.

1.2 Paper Organization

The rest of the paper is organized as follows. In the next section, we discuss the state of the art and open problems in processing ROTs. In section 3, we explain the speculative locking protocol. In section 4, we explain the basic idea of SSLR and ASLR protocols, the proof of correctness and the performance evaluation results. In section 5, we discuss how the proposed protocols differ with the SL approach proposed in [7]. In section 6, we discuss about the implementation issues. The last section contains a summary and conclusions.

2. PROCESSING OF ROTS: STATE OF ART AND OPEN PROBLEMS

As a part of state of art, we first present the related work. Next we discuss the two main protocols, 2PL and SI-based approaches related to processing of ROTs. Subsequently, we discuss the open research problems.

2.1 Related Work

In this section, we review the approaches proposed in the literature for improving the performance of ROTs. We also discuss the approaches based on speculation.

Four isolation levels are specified in ANSI/ISO SQL-92 standard [9] for processing transactions. These isolation levels are read uncommitted, read committed, repeatable read, and serializable. The processing of transactions is considered as correct if they are processed at serializable isolation level. The popular 2PL protocol [1][2] processes ROTs at serilaizability isolation level. Even though 2PL processes ROTs correctly with no data currency related issues, the performance deteriorates as data contention increases. We consider strict 2PL [17] for discussion and comparison.

To improve performance of ROTs, a new isolation level called "Snapshot Isolation (SI)" was proposed in [3]. (Pease refer to section 2.2 for details). Note that ROTs processed at SI violate the serializability criteria and receives low data currency.

In [4], a theory is discussed to convert non serializable executions under SI into serializable executions by modifying the program logic of the applications. However, this approach requires programmers to detect the static dependencies between the application programs and to modify the program which will lead to a semantically equivalent application program that can be executed correctly without violating serializability criteria. In [5], automating the task of modifying the program logic to satisfy the serializability criteria is discussed.

An approach has been proposed in [10] for distributed environment, in which ROTs are processed with a special algorithm that is different from the one used for UTs. A protocol is proposed in [11] for managing data in a replicated multi-version environment. In this protocol, the execution of ROTs is completely independent of the underlying concurrency control and replica control mechanisms. In [12], an approach has been discussed by maintaining multiple versions of data objects. In the dual copy method proposed in [13], ROTs are separated from UTs.

Speculation has been extended in [14] to optimistic protocol for improving the deadline performance in centralized real-time environments. In [7], speculation has been extended to improve the performance of distributed database systems (please refer to section 3 for details) by considering transactions which contain both the read and writes operations.

The approaches proposed so far (other than speculation approaches), improve the performance of ROTs by compromising data currency. We have proposed approaches to improve both the performance and data currency of ROTs by extending the notion of speculation.

2.2 2PL and SI-based protocols

The 2PL protocol is widely deployed in DBMS for transaction management and SI-based protocols are widely deployed to process ROTs. In this section, we explain how these protocols process ROTs.

2.2.1 Processing of ROTs in 2PL

Under 2PL [17], a transaction obtains "read (R) lock" to read an object and a "write (W) lock" to write/update the data object. In 2PL, a transaction should obtain all the required locks before performing any unlock operation. We have considered a variation of 2PL called "strict two-phase locking protocol" [17]. The strict 2PL scheduler releases all of a transaction's locks together, when the transaction terminates.

The lock compatibility matrix for 2PL [17] is shown in Figure 1. A transaction can request for Read (R)-locks or Write (W)-locks. Once a transaction releases locks, it cannot request for any more locks. The entry "yes" indicates that corresponding locks are compatible. The entry "no" indicates that the corresponding locks are incompatible. The processing of ROTs under 2PL is depicted in Figure 2. T_2, which is an ROT, has to wait for lock on the data object 'x' until T_1 commits due to read-write conflict. Similarly, T_3 which is a UT has to wait for lock on 'y' till T_1 commits due to write-write conflict.

Lock Request by T_i	*Lock Held by T_j*	
	R	W
R	yes	No
W	no	No

Figure 1. Lock Comaptibility Matrix for 2PL

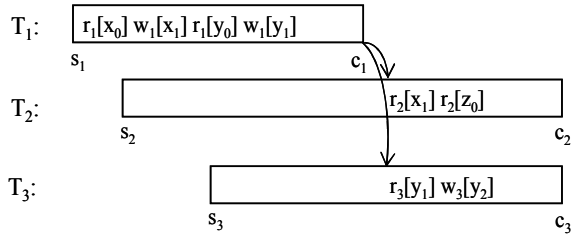

Figure 2. Depiction of Transaction Processing with 2PL

2.2.2 Processing of ROTs with SI-based protocols

The performance of ROTs can be improved by processing them at lower isolation levels by compromising both the correctness and data currency [3]. To improve the performance of ROTs, a new isolation level called "Snapshot Isolation (SI)" was proposed in [3]. In SI-based techniques, an ROT reads data from the snapshot of the (committed) data available when the transaction started or generated the first read operation. The modifications performed by other concurrent UTs which have started their execution after the ROT (T_i), are unavailable to T_i.

A variation of SI-based protocol called "First Committer Wins Rule (FCWR)" works as follows. Let T_i and T_j be UTs. T_i can successfully commit if and only if no concurrent T_j has committed writes of data objects that T_i intends to write. The processing of ROTs using FCWR is depicted in Figure 3. Both T_1 and T_3 are UTs, and T_2 is an ROT. It can be observed that T_2 reads the currently available values 'y_0' and 'z_0' and proceeds with the execution. As T_1 commits, T_3 has to be aborted as per the FCWR. It can be noted that T_2 commits with the old values and it has not accessed the updates produced by T_1 even though T_1 commits before its completion. It can be observed that T_2 has missed the updates produced by T_1 and thus violates the serializability criteria.

Note that ROTs processed at SI violate serializability criteria and receive low data currency [4]. A theory is discussed in [4], which characterizes when non-serializable executions of applications can occur under SI. It is shown in [4] that by modifying the logic of the application program, it is possible to make SI serializable. In [5], automating the task of modifying the program logic to satisfy the serializability criteria is discussed.

2.3 Open Problems

The main issues regarding processing of ROTs are correctness, data currency and performance. Even though 2PL processes ROTs correctly with no data currency related issues, the performance deteriorates as data contention increases. On the other hand, SI-based techniques improve the performance by compromising both correctness and data currency. So, the development of high performance protocol to process ROTs without any correctness and data currency issues is a open research problem.

In this thesis work, we have made an effort to develop high performance protocols without any correctness and data currency issues.

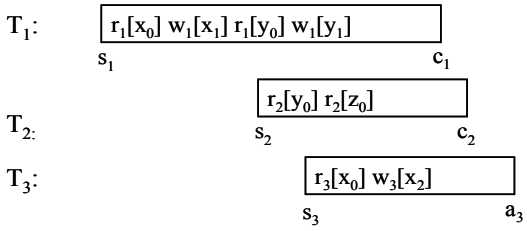

Figure 3. Depiction of Transaction Processing FCWR

3. SPECULATIVE LOCKING PROTOCOL

In [7], speculation has been extended to 2PL for improving the transaction processing performance. In a database system, whenever a transaction T_i reads data objects, these objects are copied into the private working space of that transaction. It was assumed that T_i issues $w_i[x]$ (write request on the data object 'x') after completing all its work on data object 'x'. This assumption is also adopted in [15] [16].

In the speculative locking (SL) protocol [7], a transaction produces after-images whenever it completes the work with that object. A waiting transaction is allowed to access the after-images produced by the conflicting active transactions. By accessing both before- and after-images of conflicting active transactions, the waiting transaction carries out multiple speculative executions and retains one of the executions based on the termination status of preceding transactions. The requesting transaction forms commit dependencies with the preceding transactions; it commits only after the termination of preceding transactions with which it has formed commit dependencies.

Figure 5 depicts the processing of transactions with SL. T_{ij} indicates j^{th} ($j > 0$) speculative execution of T_i. It can be observed that T_2 starts speculative executions T_{21} and T_{22}, once T_1 produces the after-image 'x_1'. T_{21} is carried out by reading 'x_0' and T_{22} is carried out by reading 'x_1'. Here, T_2 forms commit dependency with T_1; It means T_2 can only commit after T_1's termination. If T_1 commits, T_2 also commits by retaining the execution T_{22}. Otherwise, if T_1 aborts, T_{21} is retained. Note that speculation improves parallelism among T_1 and T_2. We can observe that the speculative executions of T_2 are started in a synchronous manner.

Lock compatibility matrix of SL is shown in Figure 4. Here the W-lock is partitioned into two locks: exclusive write (EW)-lock and speculative write (SPW)-lock. Transactions request R-lock for read and EW-lock for write. When a transaction produces after-image for a data object, the EW-lock is converted into SPW-lock. Under SL, only one transaction holds an EW-lock on a data object at any time. However, note that, multiple transactions can hold the R and SPW-locks on a data object at the same time. The entry "sp_yes" indicates that the requesting transaction carries out speculative executions and forms commit dependencies with the preceding transactions that hold SPW-locks. SL ensures

serializability by making transactions to wait by forming a commit dependency.

Lock Request by T_i	*Lock Held by T_j*		
	R	EW	SPW
R	yes	No	sp_yes
EW	sp_yes	No	sp_yes

Figure 4. Lock Comaptibility Matrix for SL

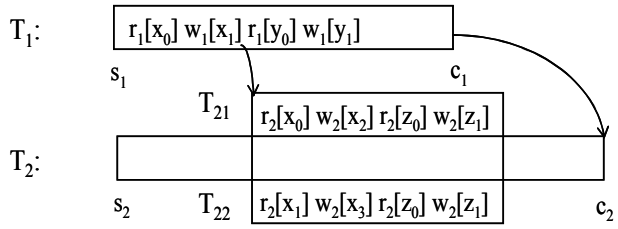

Figure 5. Depiction of Speculative Transaction Processing

In SL, at a time, a data object may have multiple versions which are organized using tree data structure with one committed value (at the root) and uncommitted values at other nodes. Whenever a transaction executes a write operation by reading a particular version ('x_i'), new object versions are created and are added as children to the corresponding node ('x_i').

A family of SL protocols, SL(n), SL(1) and SL(2) are proposed in [7]. Through simulation experiments, it has been shown that SL improves the performance significantly over 2PL by trading extra resources. Also, SL protocol produces serializable executions.

4. CONTRIBUTION OF THE THESIS

The contribution of this Ph.D. thesis is the development of two speculation-based protocols. One is synchronous speculative locking protocol for ROTs (SSLR) and the other is asynchronous speculative locking protocol for ROTs (ASLR). In this section, we first present the basic idea of both protocols. Next, the proof of correctness is discussed. Subsequently, we present the performance evaluation results.

4.1 Synchronous Speculative Locking Protocol

The SL protocol [7] was proposed to process UTs; i.e., the transactions that contain both read and write operations. In that protocol, at a time, a data object may have multiple versions which are organized using a tree data structure. Whenever a transaction executes a write operation, new uncommitted object versions are created and added to the corresponding object trees. It can be observed that write operations result in the generation of new uncommitted versions. Also, SL allows several UTs to have a SPW-lock. As a result, the number of versions for contentious data objects, and the number of speculative executions of the transactions explode with the increase in data contention. It can be noted that more extra processing power is required to support the increased number of speculative executions and data object versions.

However, regarding processing of ROTs, it can be observed that an ROT only reads the existing data and does not generate any new versions. So, if we process only ROTs through speculation, it is possible to improve the performance with less extra processing resources as compared to the resources used for processing UTs and ROTs with speculation. The explanation is as follows.

Suppose we apply SL to ROT environments which contains both ROTs and UTs. Each UT obtains EW-lock and reads the after-images produced by preceding transactions and produces new uncommitted images and converts EW-lock into SPW-lock. This allows other waiting transactions to get EW-locks. So under SL, several UTs can have SPW-lock on the same data object which causes the explosion of data object versions. As a result, the waiting transactions including ROTs have to carry out increased number of speculative executions as they conflict with all the transactions which have obtained SPW-locks on the common conflicting data objects.

It can be noted that a UT reads before-images and produces after-images and an ROT only reads before-images and does not produce any after-images. If we process only ROTs with speculation and UTs with 2PL, the number of speculative executions can be reduced due to the following reasons. When we process UTs with 2PL, only one transaction holds EW-lock at any time. If one UT is accessing a data object, other UTs have to wait. As a result, the number of versions for any data object never exceeds two. So an ROT can have conflict with only UTs which have accessed the common data objects. As a result, it is possible to reduce the number of speculative executions if we process only ROTs with speculation and UTs with 2PL.

The synchronous speculative locking for ROTs (SSLR) is proposed by adding two aspects to the basic SL protocol.

a) In SSLR only ROTs are processed with speculation. The UTs are processed with 2PL. We assume that a UT releases the locks (converts EW-lock into SPW-lock) whenever it produces after-images. Whenever an ROT conflicts with a UT, it carries out speculative executions by accessing both before- and after-images of the preceding UTs.

b) The other aspect is regarding the commitment of ROTs. In the SL [7], a waiting transaction carries out speculative executions and waits for the commitment of preceding transactions. Whereas, in SSLR whenever ROT completes execution, it commits by retaining appropriate execution. In SSLR an ROT does not wait for the termination of conflicting active transactions. However, it can be noted that, a UT waits for the termination of preceding UTs and ROTs.

The lock compatibility matrix of SSLR is shown in Figure 6. Similar to the case of speculative locking, W-lock is divided into EW-lock and SPW-lock. UTs request EW-lock for writing the data object. The EW-lock is converted into SPW-lock after the work on the data object is completed. Separate read-locks are employed for UTs and ROTs. A UT requests for RU-lock (read lock for UT) to read a data object and an ROT requests for RR-lock (read lock for ROT) to read a data object. The entry "sp_yes"

indicates that the requesting transaction carries out speculative executions and forms commit dependency with the lock holding transaction. In [8], SSLR is discussed in detail.

Lock Request by T_i	***Lock Held by T_j***			
	RR	RU	EW	SPW
RR	yes	yes	no	sp_yes
RU	yes	yes	no	no
EW	no	no	no	no

Figure 6. Lock Comaptibility Matrix for SSLR

It can be noted that the formation of commit dependency among transactions in SSLR is different from that of SL. Let T_i be an ROT and T_j be a UT. Suppose T_i forms a commit dependency with T_j. In SL, T_i commits only after the termination of T_j. Whereas in SSLR, whenever T_i completes, it can commit by retaining one of the speculative executions without waiting for T_j to terminate.

Figure 7 depicts the processing under SSLR. Here, T_2 is an ROT and T_1 and T_3 are UTs. Whenever T_1 produces after-image 'x_1', T_2 accesses both 'x_0' and 'x_1' and carries out two executions T_{21} and T_{22}, respectively. After T_2's completion, T_{21} is retained even though T_1 is not yet committed. Note that, being a UT, T_3 waits for T_1 for the release of the lock on 'x' as per 2PL rule.

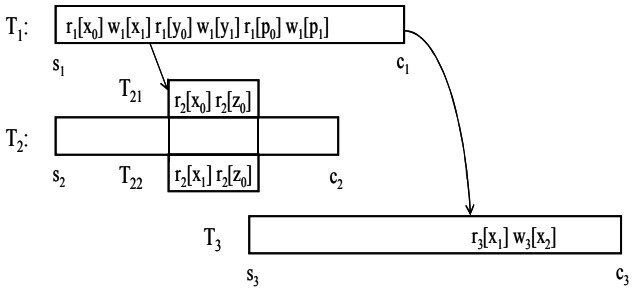

Figure 7. Depiction of Transaction Processing with SSLR

4.2 Asynchronous Speculative Locking Protocol

Note that we can process ROTs in two ways by employing speculation. One is synchronous speculation in which an ROT waits till the preceding transaction produces after-image. It means that all the speculative executions of an ROT progress at the same pace and complete at the same time. For example, in Figure 5, T_2 starts speculative executions when T_1 produces x_1. With this option, the SSLR protocol is proposed.

Alternatively, the speculative executions of ROTs can be processed in an asynchronous fashion. The basic idea is as follows. The speculative executions of an ROT can be carried out in an independent manner. The ROT is allowed to access the available data object versions and carry out the speculative executions. Whenever preceding transaction produces after-image, further speculative executions can be started in a dynamic manner. The asynchronous method of processing ROTs reduces waiting and improves the performance. We call the proposed protocol as asynchronous speculative locking protocol for ROTs (ASLR). In [20], ASLR is discussed in detail.

Figure 8 depicts the processing under ASLR. Here T_2 accesses the before-image 'x_0' and other available values of data objects 'y_0' and 'z_0' and starts speculative execution T_{21}. Once the after-image 'x_1' becomes available, another speculative execution T_{22} is started. Note that T_{21} and T_{22} are executed in a parallel manner. Whenever the processing is completed for any one of the speculative execution the ROT can be committed, provided it contains the effect of committed transactions at that instant. Note that being UT, T_3 waits for T_1 for the release of the lock on 'x' as per 2PL rule.

The lock compatibility matrix of ASLR is shown in Figure 9. Similar to the case of speculative locking, W-lock is divided into EW-lock and SPW-lock. The UTs request EW-lock for writing the data object. The EW-lock is converted into the SPW-lock after the work on the data object is completed. We propose separate read-locks for UTs and ROTs. A UT requests RU-lock (read lock for UT) for reading a data object and an ROT requests RR-lock (read lock for ROT) for reading a data object.

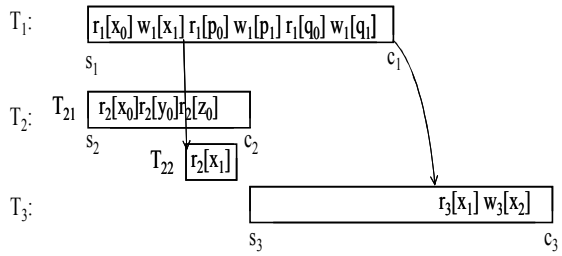

Figure 8. Depiction of Transaction Processing with ASLR

Lock Request by T_i	***Lock Held by T_j***			
	RR	RU	EW	SPW
RR	yes	yes	sp_yes	sp_yes
RU	yes	yes	no	no
EW	no	no	no	no

Figure 9. Lock Compatibility Matrix for ASLR.

4.3 Correctness

We briefly argue that the schedules produced by ASLR are serializable [17]. Under SSLR and ASLR, the UTs are handled using 2PL rules which capture all Read-Write and Write-Write conflicts. The SSLR and ASLR rules capture all the Write-Read conflicts. For each Write-Read conflict, SSLR and ASLR rules ensure that Read operation reads from the preceding Write operation. Suppose, let T_i be an ROT and conflicts with "n" transactions and commits at time "t". We can divide "n" transactions into two sets. One set is "committed set (CS)" which includes the transactions which have committed before "t" and another set is "uncommitted set (US)" which includes the transactions which are not committed at time "t". As per SSLR and ASLR rules, T_i is committed by including the effects of all the transactions in CS. So, T_i's execution is equivalent to the serial execution produced after CS. The execution of each transaction in US is equivalent to the serial execution after T_i. It means that

the execution is equivalent to the serial order CS << T_i << US ("<<" denotes the partial order). So, it can be easily proved that SSLR and ASLR produce serializable schedules.

4.4 Performance Evaluation

In this section, first we discuss the simulation model and protocols considered for comparison briefly. Next we present the evaluation results by considering both unlimited and limited resource environments.

Table 1. Simulation Parameters, Meaning and Values

Parameter	Meaning	Values
dbSize	Number of objects in the database	1000
cpuTime	Time to carry out CPU request	5ms
ioTime	Time to carry out I/O request	10ms
rotMaxTranSize	Size of largest ROT transaction	20 objects
rotMinTranSize	Size of smallest ROT transaction	15 objects
utMaxTranSize	Size of largest UT transaction	15 objects
utMinTranSize	Size of smallest UT transaction	5 objects
noResUnits	Number of RUs(1 CPU, 2 I/O)	8
mpl	Multiprogramming Level (10 – 100)	Simulation Variable

A discrete event simulator based on a closed-queuing model has been developed based on [18]. The description of parameters used in the simulation with values is shown in Table 1. We have employed throughput as the performance metric which can be defined as the number of transactions completed per second.

Protocols: We have compared SSLR and ASLR with 2PL, FCWR, SI-2PL, and SL. In 2PL, SL, SSLR and ASLR transactions request for locks in a dynamic manner, one by one. For SL, SSLR and ASLR, we have assumed that all the speculative executions of a transaction are carried out in parallel. In FCWR, the conflicts between UTs are managed by aborting the transactions. Aborted transactions are resubmitted after the time duration which equals to the average response time. We also consider SI-2PL approach. SI-2PL is a variation to the approach proposed in [11][12]. In SI-2PL, ROTs are processed with snapshot isolation and UTs are processed with 2PL.

(i) Results under unlimited resources

Figure 10 shows how throughput performance for 2PL, FCWR, SL, SSLR, ALSR and SI-2PL vary with MPL. It can be noted that the performance of ASLR is significantly higher than that of 2PL and FCWR. 2PL performs poorly as the waiting time of the transactions is more. In FCWR, the performance deteriorates due to its "first committer wins rule" as more number of UTs gets aborted as data contention increases. Note that ASLR's performance is better than both SL and ASLR due to the reduced waiting as a result of asynchronous speculation. Note that in both SL and SSLR, ROTs wait for the lock conversion from EW-lock to SPW-lock. We observe that the performance of SL and SSLR is close. Also, it can be observed that the performance of ASLR is more than that of SI-2PL when contention increases. (Note that both SI-2PL and FCWR suffer from correctness and data currency problems.)

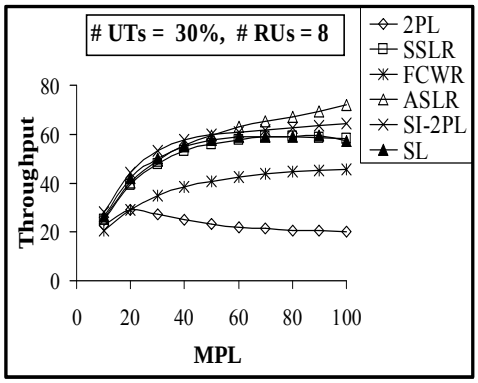

Figure 10. MPL versus Throughput

(ii) Results under limited resources

Figure 11 shows the performance of 2PL, SL, SSLR and ASLR protocols by simulating limited resources environments. The resources are allocated in terms of memory units (MUS). We assumed that each memory unit carries out one speculative execution. If sufficient number of MUS is not available to carry out speculative executions, the transaction is put to wait. It can be observed that the performance of both ASLR (also SSLR) reaches the maximum value and saturates at MUS values equal to 1.2*MPL. Note that the performance of SL does not reach the performance of ASLR even after doubling the MUS values equal to 2*MPL. Also, note that the performance of 2PL is immune to the additional resources.

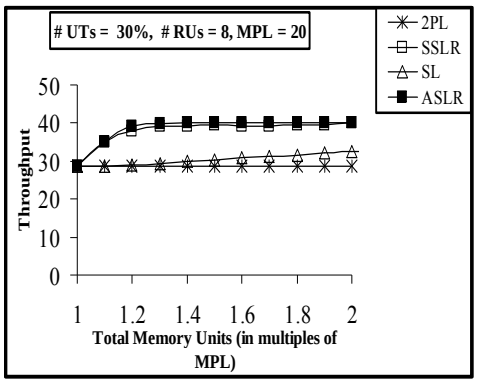

Figure 11. Throughput Performance with limited resources

Overall, the simulation experiments show that the performance of ASLR is better than 2PL, FCWR, SL and SSLR protocols. ASLR requires a fraction of additional resources equal to 0.2*MPL to achieve high performance.

Table 2 shows the comparison of SSLR, ASLR with SL, 2PL and FCWR protocols on several aspects. Overall the proposed

protocols are better over 2PL in case of performance, over SL protocol regarding consumption of extra processing resources, and over SI-based protocols regarding correctness and data currency,

Among SSLR and ASLR, comparison shows that ASLR is better in terms of resource utilization and performance. However, we feel that in case of wide area network environments, the difference in performance improvement may not be the same. We investigate this, as a part of future work.

Table 2. Comparison of ASLR, SSLR, SL, 2PL and FCWR protocols

Parameter	ASLR	SSLR	SL	2PL	FCWR
Throughput	More than SSLR	High	High	Low	Medium
Device utilization	More than SSLR	High	High	Low	High
Performance with extra resources	Increases	Increases	Slow increase	No Change	No Change
Extra resources requirement	Manageable (0.2 times)	Manageable (0.2 times)	Very High	Not required	Not required
Data Currency	High	High	High	High	Low
Correctness of processing	Serializable	Serializable	Srializable	Serializable	Not Serializable

5. DIFFRENCES BETWEEN SL AND PROPOSED PROTOCOLS

In this section, we discuss the differences between the SL protocol and the SSLR and ASLR approaches.

(i) Aim: The SL protocol is proposed to improve the performance of UTs in distributed database systems. The SSLR and ASLR protocols are developed to improve the performance of ROTs by identifying the specific characteristics of ROT processing environments.

(ii) Basic Strategy: In SL, all transactions are allowed to speculate. In the proposed protocols, only ROTs are allowed to carry out speculative executions and UTs follow 2PL.

(iii) Number of versions of data object: In SL, whenever a UT accesses a data object, it adds after-images to the data object tree, allows other waiting transactions to access the same data object tree. So the number of versions in the data object tree explodes.

However, in the proposed protocols, no two UTs can access a particular data object simultaneously, as UTs follow 2PL. So, at a particular instant of time, for a data object, the available versions never exceed two.

(iv) Speculative executions: In SL, whenever a transaction accesses a data object, each execution of that transaction starts new speculative executions equal to the number of versions in the data object tree. As number of versions explodes, the number of speculative executions of a transaction also explodes.

In the proposed protocols, the number of versions for a data object never exceeds two, as per our earlier discussion. So, in the proposed SSLR and ASLR protocols, less number of speculative executions is carried out by ROTs.

(v) Commitment of transactions: In SL, a transaction can commit only after the termination of preceding transactions with which it has formed commit dependencies.

However, both SSLR and ASLR protocols allow an ROT to complete its execution, without waiting for the termination of preceding transactions with which it has formed commit dependencies. As a result the performance improves.

(vi) Type of speculation: In SL, it was proposed that speculative executions of a transaction are carried out in a synchronous manner.

We have developed SSLR by considering synchronous method of speculative executions as in SL. In addition, we have also proposed ASLR protocol by considering asynchronous method of carrying out speculative executions.

(vii) Extra resources requirement: The SL requires more extra processing resources.

The proposed protocols require less number of extra processing resources due to the optimizations proposed.

(viii) Lock requirement: Three types of locks used in SL: R-lock, EW-lock and SPW-lock.

In the proposed protocols four types of locks are used: RR-lock, RU-lock, EW-lock and SPW-lock. An ROT requests RR-lock to read a data object, whereas a UT requests RU-lock to read. These two read-locks (RU-lock and RR-lock) are necessary to distinguish between read operations of ROTs from read operations of UTs. Note that, only read operations of ROTs are processed with speculation and the read operations of UTs are executed without speculation.

6. DISCUSSION

In this section, we discuss the implementation issues regarding processing of ROTs in SSLR and ASLR. However, the detailed investigation on these issues will be carried out as a part of future work.

(i) Pre-compilation. In this paper, we assume that a UT releases the lock whenever it produces the after-image. We assume that it is possible to put markers for each data object to indicate when the transaction finishes work on that object. Since the transactions are stored procedures, we believe that it is possible to put the lock conversion markers by analyzing the stored procedures.

(ii) Speculative executions. We have assumed that speculative executions of transaction are carried out in parallel by considering multi-processor environment. It can be noted that additional memory can be added to the system at lesser cost. Since CPU speed is high in the orders of magnitude than the disk I/O, even in a single processor environment, the CPU time can be used productively to improve the performance of ROTs.

(iii) Scan and index. We have to investigate the handling of indexes and scans when modifications are performed to the database under SSLR and ASLR protocols.

(iv) Performance of SSLR and ASLR protocols in wide area network environments. We conducted the experiments for the performance evaluation of SSLR and ASLR protocols by considering centralized environment. However, the effectiveness of SSLR and ASLR protocols in distributed environment has to be investigated.

7. SUMMARY AND CONCLUSIONS

The development of high performance protocol to process ROTs without any correctness and data currency issues is a open research problem. As a part of the Ph.D. thesis work, we have investigated high performance synchronization protocols for ROT intensive environments. The proposed protocols have been developed using speculation and they do not suffer from any correctness and data currency issues. We have developed two speculation-based approaches. One is synchronous speculative locking protocol for ROTs and the other is asynchronous speculative locking protocol for ROTs. Through simulation results, it has been shown that the proposed protocols improve the performance significantly over 2PL and SI-based protocols with a fraction (0.2 times) of additional resources.

As a part of future work, in addition to the issues listed in section 6, we are also planning to investigate the performance of the proposed protocols through benchmarks after implementing the protocols in a prototype DBMS. We are also planning to develop protocols based on speculation, to improve the performance of real-time read-only transactions.

Improving the performance of ROTs without correctness and data currency issues is a crucial factor in several e-commerce environments like stock marketing, airline operating systems and other web services. Also, currently multi-core CPUs are available with high processing power. Main memory cost is also coming down. The proposed protocols provide the scope for improving the performance of ROTs in such environments by trading extra processing resources.

8. REFERENCES

[1] Eswaran, K., Gray, J., Lorie, R., Traiger, I. 1976. "The notions of consistency and predicate locks in database systems". Communications of the ACM, 19, no.11: 624-633.

[2] Gray, J.; Reuter, A. 1993. Transaction Processing: Concepts and techniques. Morgan Kaufmann.

[3] Hal Berenson, Phil Bernstein, Gray, J., Jim Melton, Elizabeth O'Neil and Patrick O'Neil. 1995. "A Crtique of ANSI SQL Isolation Levels." In ACM SIGMOD.

[4] Fekete, A., Liarokapis, D., O'neil, E.,O'neil, P., Shasha,D. 2005."Making Snapshot Isolation Serializable.", ACM Transactions on Database Systems,30, no.2, (June): 492-528.

[5] Sudhir Jorwekar, Alan Fekete, Krithi Ramamritham, and S. Sudarshan. 2007. "Automating the Detection of Snapshot Isolation Anomalies." In VLDB '07, Austria, (September).

[6] Dimitri Theodoratos, Mokrane Bouzeghoub. 1999. "Data Currency Quality Factors in Data Warehouse Design." In Proceedings of the International Workshop on Design and Management of Data Warehouses, Germany, (June). 15-1 – 15-15.

[7] Krishna Reddy, P., Masaru Kitusuregawa. 2004. "Speculative Locking Protocols to Improve Performance for Distributed Database Systems",IEEE Transactions on Knowledge and Data Engineering, 16 , no.2, (February): 154-169.

[8] Ragunathan, T., Krishna Reddy, P. 2007. "Improving the performance of Read-only Transactions through Speculation." In 5th International Workshop on Databases in Networked Information Systems, DNIS 2007, (October). LNCS, vol. 4777, 203-221.

[9] ANSI X3.135-1992, 1992. American National Standard for Information Systems- Database Language-SQL, November.

[10] Hector Garcia-Molina, Gio Wiederhold. 1982."Read-Only Transactions in a Distributed Database", ACM Transactions on Database Systems, (June):209-234.

[11] Satyanarayanan, O. T., Agrawal, D. 1993. "Efficient Execution of Read-only Transactions in Replicated Multiversion Databases", IEEE transactions on Knowledge and Data Engineering, 5, no.5, (October): 859-871.

[12] C. Mohan, Hamid Pirahesh, and Raymond Lorie. 1992. "Efficient and Flexible Methods for Transient Versioning of Records to Avoid Locking by Read-Only Transactions." In ACM SIGMOD.

[13] Bajojing Lu, Qinghua Zou and William Perrizo. 2001. A "Dual Copy method for Transaction Separation with Multiversion Control for Read-only Transactions." In Proceedings of the ACM Symposium on Applied Computing. 290-294.

[14] Bestavros, A., Braoudakis, S. 1995. "Value-Cognizant Speculative Concurrency Control Protocol." In Proceedings of 21st Very Large Databases Conference. 122-133.

[15] Agrawal, D., El Abbadi, A., Lang, A. E. 1994. "The Performance of Protocols Based on Locks with Ordered Sharing". IEEE Transactions on Knowledge and Data Engineering, 6. no.5, (October):805-818.

[16] Salem, K., Garcimolina, H., Shands, J. 1994. "Altruist Locking". ACM Transactions Data base Systems, 19. no.1, (March):117-165.

[17] Bernstein, P. A., Hadzilacos, V., Goodman, N. 1987. Concurrency Control and Recovery in Database Systems. Addison-Wesley.

[18]. Rakesh Agrawal, Michael J. Carey, Miron Livny. 1987. "Concurrency Control Performance Modeling: Alternatives and Implications". ACM Transactions on Database Systems, 12, no. 4, (December): 609-654.

[19] Hongfei Guo, Per-Ake Larson, Raghu Ramakrishnan, Jonathan Goldstein. 2004. "Relaxed Currency and Consistency: How to say "Good Enough" in SQL." In ACM SIGMOD, (Paris, France, June 13-18). 815-826.

[20] Ragunathan, T., Krishna Reddy, P. 2008. "Improving the Performance of Read-only Transactions through Asynchronous Speculation." (To appear in High Performance Computing and Simulation Symposium, HPCS 2008, Ottawa, Canada).