2010 12th IEEE International Conference on High Performance Computing and Communications

Extending Speculation-Based Protocols for Processing Read-only Transactions in Distributed Database Systems

Mohit Goyal, T. Ragunathan and P. Krishna Reddy
International Institute of Information Technology, Hyderabad, India.
Email: goyal@research.iiit.ac.in, ragunathan@research.iiit.ac.in and pkreddy@mail.iiit.ac.in

***Abstract*—The main issues in processing read-only transactions (ROTs) are correctness, data currency and performance. The popular two-phase locking (2PL) protocol processes the transactions correctly according to serializability criteria, but its performance deteriorates with data contention. To improve the performance, snapshot isolation (SI)-based approaches have been proposed. Even though SI-based approaches improve performance, however they compromise both correctness and data currency aspects. In the literature, an effort has been made to propose improved approaches to process ROTs based on the notion of speculation. The speculation-based approaches improve performance without compromising both correctness and data currency aspects. In this paper, we have extended the speculation-based protocols for processing ROTs in a distributed database systems. It has been identified that an ROT under speculation-based protocols in distributed database systems require a commit phase. In addition, additional messages are required for making speculative versions available to ROTs during the execution phase of an update transaction. In spite of these overheads, the proposed protocols reduce the waiting time of ROTs significantly by increasing the parallelism without violating both correctness and data currency aspects. The simulation experiments show that the proposed protocols significantly improve the performance over 2PL and SI-based protocols.**

I. Introduction

Distributed database management systems (DDBMSs) are major components of modern information systems for storing data and for serving large number of information access requests issued by online users. These information systems frequently receive read-only transactions (ROTs) or queries. An ROT only reads the data objects from the database and does not change the state of database. On the other hand, update transactions (UTs) performs both read and write operations on the database. The main issues in developing protocols for processing ROTs are: correctness [14], data currency and performance. A protocol to process ROTs should execute transactions in a correct manner. It means if a set of transactions are processed in parallel, the processing should be equivalent to a serial execution of the same set of transactions[2]. The notion of "data currency" is explained as follows. 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 [13]. If 't'=0, data currency for a transaction is the highest. The protocol should process transactions without any data currency problems, i.e., 't'=0. Regarding performance, the protocol should provide high throughput.

Two-phase locking protocol (2PL) [5] provides high data currency to ROTs by satisfying serializability criteria [2]. Snapshot-isolation (SI) based approaches [1] are proposed for improving the performance of ROTs by compromising both correctness and data currency aspects. In the literature, the speculation-based (SL) protocols [15] have been proposed for improving the performance of ROTs without compromising both correctness and data currency aspects.

In this paper, we have extended the SL-based protocols for processing ROTs in distributed database systems. In distributed systems, transaction processing can be divided into two parts. In the first part, transaction obtains locks and complete the execution and in the second part, transaction commits the values by following two-phase commit (2PC) protocol [5]. For processing ROTs in distributed database systems, it has been identified that an ROT under SL-based protocols has to undergo additional commit phase. Further, additional messages are required for sending speculative versions during execution of an UT. We have developed the SL-based protocols for ROTs by incorporating message transmission and commit processing aspects. The proposed protocols improve the performance significantly by reducing the waiting time for ROTs. The simulation experiments show that the proposed protocols significantly improve the performance over 2PL and snapshot-isolation based protocols.

The rest of the paper is organized as follows. In the next section, we explain the related work. In Section 3, concurrency control protocols for centralized systems are explained. In Section 4, we explain the proposed concurrency control protocols for ROTs in distributed databases. In Section 5, we present the experimental results. Last section contains summary and conclusions.

II. Related work

In the literature, the transaction processing problem has been well studied in the contexts of both centralized database systems and distributed databases [7] [4] [6]. An approach has been proposed in [8] for distributed environment in which ROTs are processed with a special algorithm that is different from the one used for UTs. Efforts are also made in real-time

978-0-7695-4214-0/10 $26.00 © 2010 IEEE
DOI 10.1109/HPCC.2010.69

DDBMS by following separate algorithm for ROTs [17] but serializability is not ensured.

A protocol is proposed in [9] for managing data in replicated multi-version environment. In [10], an approach has been discussed for maintaining multiple versions of data objects. In this technique, based on the arrival time, ROTs read particular versions of the data.

A dual copy method is proposed in [11] for processing ROTs. To improve the performance of ROTs, a new isolation level called "Snapshot Isolation (SI)" was proposed in [1]. Note that ROTs processed at SI receive low data currency [3].

In the literature, an effort has been made to improve the transaction processing performance by using the notion of speculation. Two speculation-based approaches are proposed for centralized systems [15]. We discuss the details of these protocols in Section (III).

III. The 2PL, SI-based and SL-based ROT protocols

In this section, after explaining the system model and notations, we discuss the processing of ROTs under 2PL, SI-based and SL-based protocols.

A. System Model and notations

A database system consists of a set of data objects which are denoted by 'x','y',..., transactions are represented by T_i, T_j, T_k, ..., and sites are represented by S_i , S_j,..., where i, j, k, ..., are non-negative integer values. 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 $WS(T_i) \cap WS(T_j) \neq \emptyset$.

Data objects are denoted with 'x','y',... 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' by the transaction T_i and $w_i[x_j]$ denotes that the transaction T_i performs a write operation on a particular version of 'x' and produces 'x_j'. The notations 's', 'e', 'c', and 'a' denote the start, completion of execution, completion of commit processing and abort of transactions, respectively.

B. Two-phase locking protocol

In 2PL, ROTs are forced to wait for the completion of UTs incase of conflicts. The processing of ROTs under 2PL is depicted in Figure 1. In this, both T_1 and T_3 are UTs and T_2 is an ROT. It can be observed that 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.

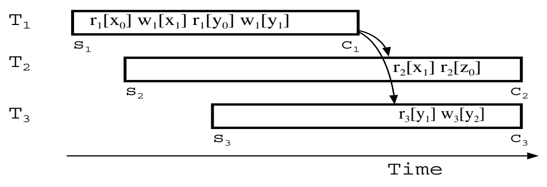

Fig. 1. Depiction of transaction processing with 2PL

C. SI-based protocol

In SI-based techniques [1], an ROT (T_i) reads data from the snapshot of the (committed) data available when T_i has started or generated the first read operation. The modifications performed by other concurrent UTs, which have started their execution after T_i are unavailable to T_i. In SI-based FCWR (first committer wins rule), a transaction (T_i) commits if and only if no concurrent transaction (T_j) has already committed writes of data objects that T_i intends to write. In Figure 2, 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. Simultaneously, T_3 also reads 'x_0' and produces 'x_2'. Note that, FCWR allows only one of the conflicting UTs to commit. So, T_3 has to be aborted as T_1 commits. However, as per FCWR, 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 and therefore receives low data currency.

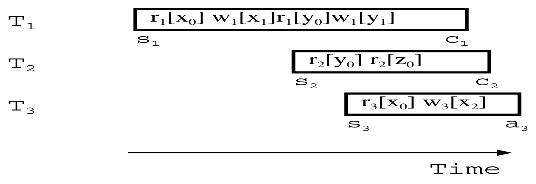

Fig. 2. Depiction of transaction processing with FCWR

D. SL-based protocols for ROTs in centralized database systems

In SL-based protocols, UTs follow 2PL and ROTs are processed using speculation. Two SL-based protocols namely synchronous speculation-based locking protocol for ROTs (SSLR) and asynchronous speculation-based locking protocol for ROTs (ASLR) are proposed in the literature which is explained below.

(i) Synchronous speculation-based protocol for ROTs

In SSLR protocol [12], whenever an UT produces after-image, it allows ROTs to access the after-image. Whenever an ROT conflicts with UT, waits for the production of after-images and carries out speculative executions by accessing both before- and after-images of conflicting data objects. After completion, based on the commit status of the conflicting UTs, it retains the appropriate speculative execution.

The lock compatibility matrix of SSLR is shown in Figure 3. The W-lock is divided into executive write(EW)-lock and

Lock requested	**Lock held by T_i**			
by T_j	RR	RU	EW	SPW
RR	yes	yes	no	ssp_yes
RU	yes	yes	no	no
EW	no	no	no	no

Fig. 3. Lock compatibility matrix for SSLR

speculative write(SPW)-lock. The EW-lock is requested by UTs for writing the data object. The RU-lock (Read lock for UT) is requested by UTs for reading a data object. The RR-lock (read lock for an ROT) is requested by ROT for reading a data object. When a transaction produces after-image for a data object, the EW-lock is converted into SPW-lock. The entry "ssp_yes" (synchronous speculation yes) indicates that the requesting ROT carries out speculative executions and forms a commit dependency. In SSLR, if T$_i$ carries out speculative executions and forms a commit dependency with T$_j$, T$_i$ commits whenever it is completed (say at time 't') by retaining appropriate speculative execution based on the termination status of T$_j$ at time 't'. The processing of ROTs

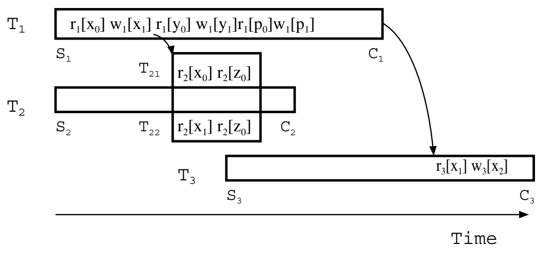

Fig. 4. Depiction of Transaction processing with SSLR

under SSLR is illustrated in Figure 4. Here, T$_1$ and T$_3$ are UTs which are processed with 2PL and T$_2$ is an ROT which is processed with SSLR. T$_1$ obtains EW-lock on data object 'x'. It reads 'x_0' and produces 'x_1' and converts the EW-lock on 'x' to SPW-lock. T$_3$, being a UT, waits till T$_1$ releases the lock on 'x'. The ROT T$_2$ is processed as follows. Note that even though both T$_1$ and T$_2$ have arrived at the same instant, T$_2$ waits till T$_1$ produces after-image 'x_1', T$_2$ carries out two executions T$_{21}$ and T$_{22}$ by accessing 'x_0' and 'x_1' respectively. Note that, T$_{21}$ and T$_{22}$ are carried out synchronously. After T$_2$'s completion, T$_{21}$ is retained even though T$_1$ is not yet committed. We can observe that, T$_2$ is committed without waiting for the termination of T$_1$. Also, the transactions are serialized as per the order T$_2$ $\ll$ T$_1$ $\ll$ T$_3$.

(ii) Aynchronous speculation-based protocol for ROTs

In ASLR protocol [13], whenever an UT produces after-image, it allows ROTs to access the after-image. Whenever an ROT conflicts with a UT, it carries out speculative executions by accessing available before-images. Whenever after-image is produced, further speculative executions are started. In this way, speculative executions are carried out in an asynchronous manner. Whenever one of the speculative executions completes, it confirms with the conflicting UTs. If it is successful, it commits.

The lock compatibility matrix of ASLR is shown in Figure 5. The locks used here are similar to SSLR protocol. However, there is a difference with SSLR, on the aspect of compatibility of RR-lock with other locks. In SSLR, RR-lock is speculatively compatible only with SPW-lock whereas in ASLR, RR-lock is speculatively compatible with both EW-lock and SPW-lock. But, the nature of compatibility is different from SSLR. So, a different name called "asp_yes"(asynchronous

Lock requested	**Lock held by** T_i			
by T_j	**RR**	**RU**	**EW**	**SPW**
RR	yes	yes	asp_yes	asp_yes
RU	yes	yes	no	no
EW	no	no	no	no

Fig. 5. Lock compatibility matrix for ASLR

speculative yes) is used here which means that the ROT carries out the possible speculative executions by accessing available versions of data object and forms a commit dependency with the preceding UTs. The method of commitment is also different (refer to discussion regarding "commit processing"). Figure 6 depicts the processing under ASLR. Here, T$_1$ and

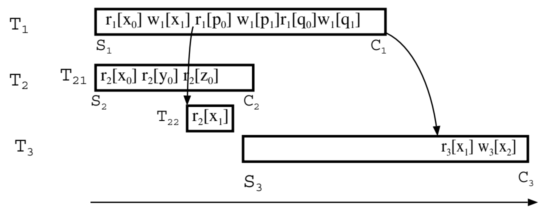

Fig. 6. Depiction of Transaction processing with ASLR

T$_3$ are UTs which are processed with 2PL and T$_2$ is an ROT which is processed with ASLR. 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. We can observe that, T$_{21}$ does not depend on T$_1$ and it is committed once it completes the execution without waiting for T$_1$. So, T$_{22}$ is aborted. Note that being UT, T$_3$ waits for T$_1$ for the release of the lock on 'x' as per 2PL rule.

IV. **Distributed speculation-based protocols for ROTs**

A. Transaction processing in DDBMS

The processing of transactions in DDBMS is divided into two phases. We discuss the processing by considering distributed 2PL [14].

1) *Execution*: Suppose a transaction T$_i$ arrives at a site which we call as "home site". When a transaction T$_i$ requires to read/write a remote data object 'x', it sends lock request to the object site where 'x' resides. If lock is granted, data object is sent to the home site along with reply message. Once it has obtained all the locks, it completes execution and enters into the commit phase.
2) *Commit phase*: T$_i$ initiates 2PC [5] by taking home site as coordinator site and the sites from where it has read data objects as participant sites. In first phase, it sends *PREPARE* message to all the participant sites. If all sites respond with "yes" then it issues *GLOBAL_COMMIT*

message. If any one of the sites respond negatively then T_i is aborted and *GLOBAL_ABORT* message is sent to all participants sites.

B. Basic idea

While extending SL-based approach to DDBMS, following issues arise.

- Firstly, in SL-based protocols, UTs follow 2PL and they produce after-images whenever their last write operation on data object is finished. This after-image needs to be communicated to corresponding object site and target sites where ROTs are waiting to access the same.
- Secondly, the commitment of ROTs is different from the procedure discussed in SL-based protocols of centralized database systems. In SL-based protocols for ROTs in DDBMS, the ROT has to exchange one more round of message communication to select appropriate speculative execution for completing. So, this is an additional requirement for ROTs in the proposed protocols.

By incorporating these issues, we have proposed two distributed SL-based protocols for ROTs namely distributed synchronous speculative locking protocol for ROTs (DSSLR) and distributed asynchronous speculative locking protocol for ROTs (DASLR).

C. The DSSLR protocol

We explain the DSSLR, by listing the steps to be followed for UTs and ROTs.

Protocol for UTs:

1.1 *Execution*: UTs follow 2PL. Whenever after-image is produced, it is communicated to the object site and the target sites where ROTs are waiting.

1.2 *Commit phase*: After completion, 2PC is followed.

Protocol for ROTs:

2.1 *Execution*: The ROT sends lock requests to the object sites. If the lock request of the ROT conflicts with UTs, it waits till the production of after-image by UTs and both before- and after-images are transferred to the home site of the ROT. An ROT carries out speculative executions by accessing both before- and after- images. A “dependent_set” is also maintained for each ROT. It is a set which contains identifiers of conflicting UTs. In addition a “dependent_list” is also maintained for each speculative execution which contains the identifiers of UTs from which it has read the after-images. When T_i obtains all the locks, it completes the execution and enters into commit phase.

2.2 *Commit phase*: After completion, an ROT may have several speculative executions. For selecting an appropriate speculative execution, the ROT exchanges one round of communication with home sites of UTs specified in the “dependend_set” to know their commit status. Two kinds of replies are received: committed or active/aborted. Based on these replies, the speculative execution which contains the effect of committed UTs will be chosen for commitment by considering “dependent_list”.

D. The DASLR protocol

We explain the DASLR, by listing the steps to be followed for UTs and ROTs.

Protocol for UTs:

1.1 *Execution*: UTs follow 2PL. Whenever after-image is produced, it is communicated to the object site and the target sites where ROTs are waiting.

1.2 *Commit phase*: After completion, 2PC is followed.

Protocol for ROTs:

2.1 *Execution*: The ROT sends lock requests to the object sites. Whenever a lock request of an ROT conflicts with UTs based on a data object and after-image of that data object is not available, then ROT continues the speculative executions by reading the before-image of that data object. Whenever the after-image of the data object is sent by the conflicting UT, further speculative executions are generated for that ROT. A “dependent_set” is also maintained for each ROT which contains identifiers of conflicting UTs. In addition a “dependent_list” is also maintained for each speculative execution which contains the identifiers of UTs from which it has read the after-images. When T_i obtains all the locks, it completes the execution and enters into commit phase.

2.2 *Commit phase*: Whenever one of the speculative executions of the ROT is completed, one round of communication is performed with home sites of UTs specified in the “dependent_list” of this ROT, to know their commit status. Two kinds of replies are received: committed or active/aborted. Based on these replies, the speculative execution is committed if it contains the effect of committed UTs.

V. **Discussion**

Correctness proof for synchronous approach is given in [13]. Asynchronous protocol can also be proved on the lines of SSLR. Both ASLR and SSLR do not suffer from data currency issues. It can be observed that the notion of speculation allows ROTs to access the after-images produced by UTs which results in increased parallelism. So, in DDBMS, the proposed protocols perform better than both 2PL and SI-based FCWR protocols in spite of the overhead involved in the communication of after-images and commitment of ROTs due to increased parallelism because of speculative processing. So, by carrying out multiple speculative executions for an ROT, the proposed protocols provide an opportunity to improve the performance in DDBMS without compromising correctness and data currency aspects.

VI. **Simulation results**

We first explain the simulation model and then discus the simulation results of 2PL, SI, SL, DSSLR and DASLR locking protocols.

A. Simulation model

We developed a discrete event simulator on a closed queuing model. For the sake of simplicity, the communication network is modeled as a fully-connected network. Any site can send messages to all the sites at the same time. Each site is modeled with multiple CPU servers and I/O servers [16]. Requests in the CPU and I/O queue are served in the first-come first-serve (FCFS) manner. The I/O model is a probabilistic model of a database that is spread out across all of the disks. A separate queue is maintained for each I/O server. Whenever a transaction needs service, it randomly (uniform) chooses a disk and waits in the I/O queue of the selected I/O server[16]. The meaning of each model parameter for simulation is given in Table 1. A new transaction is assigned an arrival site which is chosen randomly over num_sites. The parameter trans_delay is the time required to transmit a message between sites. The WAN behavior is realized by varying trans_delay. The parameter resource units(RUs) is the number of resource units at each site. The settings for db_size, trans_size, rotMaxTranSize, rotMaxTranSize, utMinTranSize, utMaxTranSize parameters are given in Table 1 [16]. The parameter MPL denotes the number of active transactions existing in the system.

B. Performance metrics

The primary performance metric of our experiments is throughput, that is, the number of transactions completed per second. We have also considered number of messages as a performance metric which takes into account all types of messages exchanged among sites like lock-request, lock-reply etc.

C. Performance Results

Figure 7 shows the throughput performance of 2PL, FCWR, SL, DSSLR and DASLR protocols at $trans\_delay$ =10ms. The performance of 2PL decreases with the data contention due to increased waiting. Even though FCWR protocol performs better than 2PL, its performance is saturated above 2PL due to more number of aborts. It can be observed that SL performs better than both 2PL and FCWR. Both DSSLR and DASLR improve the performance significantly over other protocols due to reduced waiting time of ROTs. DSSLR performs better than DASLR till MPL=60. A performance cross-over is observed among DSSLR and DASLR at MPL=60. This is due to the fact that as data contention increases, the waiting time is increased in DSSLR whereas waiting time is reduced in DASLR, as the ROTs are executed in an asynchronous manner.

Figure 8 shows the throughput performance for 2PL, FCWR, SL, DSSLR and DASLR at $trans\_delay$=100ms. The performance of 2PL, FCWR and SL protocols follow similar trends of LAN environment. It can be observed that the performance cross-over is moved to MPL=80 due to increase in transmission time.

Figure 9 shows the number of messages exchanged for 2PL, FCWR, SL, DSSLR and DASLR. As expected, 2PL requires less number of messages. The DASLR protocol requires more messages than DSSLR. The FCWR protocol requires highest number of messages due to more aborts.

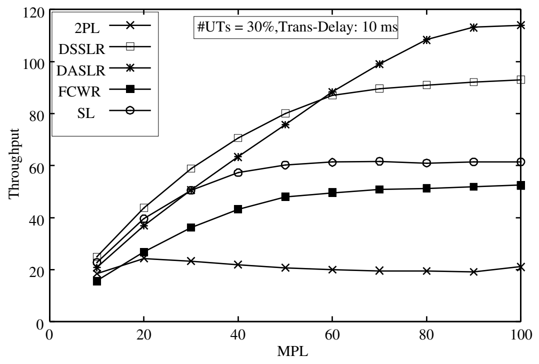

Fig. 7. MPL vs Throughput

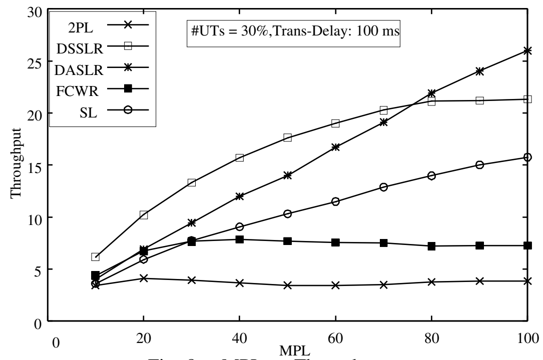

Fig. 8. MPL vs Throughput

Figure 10 shows the throughput performance for 2PL, FCWR, SL, DSSLR and DASLR protocols at MPL=60. As $trans\_delay$ increases, the performance of all protocols fall because of increased waiting time.

VII. **Conclusions and future work**

In this paper, we have proposed SL-based protocols for DDBMS to process ROTs. The ROT processing with speculation in DDBMS requires a commit phase. By carrying out multiple speculative executions for an ROT, the proposed

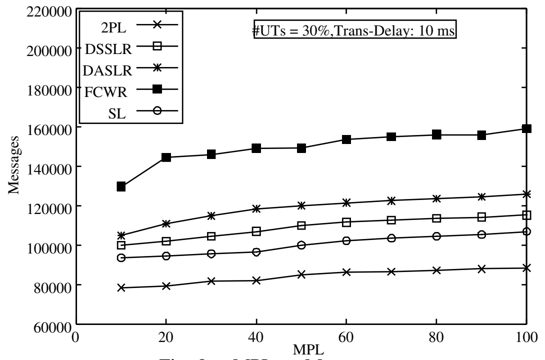

Fig. 9. MPL vs Messages

TABLE I

SIMULATION PARAMETERS, MEANING AND VALUES

Parameter	Meaning	Value
dbSize	Number of objects in the database	1000
num_sites	Number of sites in DDBMS	5
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
% of ROTs	Percentage of ROTs	70%
% of UTs	Percentage of UTs	30%
MPL	Multiprogramming Level (10-100)	Simulation Variable
trans_delay	Transmission time between two sites(5-100)	Simulation Variable

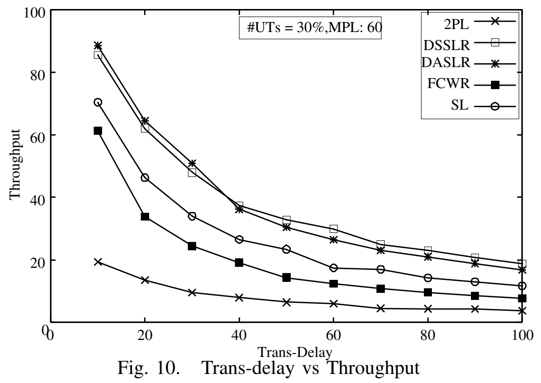

Fig. 10. Trans-delay vs Throughput

protocols provide an opportunity to improve the performance in DDBMS without compromising correctness and data currency aspects. Through simulation experiments, it has been shown that the proposed approaches improve the performance significantly over 2PL and SI-based protocols. As a part of future work, we are planning to investigate speculation-based approaches for replicated database systems.

REFERENCES

[1] 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.

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

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

[4] C.Mohan, B.Lindsay and R.Obermark. "Transaction Management in the R* Distributed Database Management System", ACM Trans. Database Systems, vol.11, no.4, pp.378-396, 1986.

[5] J.N.Gray,"Notes on Database Operating Systems:In Operating Systems an Advanced-Course", Lecture Notes in Computer Science, vol.60, pp.393-481, 1978.

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

[7] V.Kumar. "Concurrency Control Mechanisms in Centralized Database Systems" Prentice-Hall, 1996.

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

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

[10] 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.

[11] 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.

[12] T. Ragunathan and P. Krishna Reddy, "Improving the Performance of Read-only Transactions through Asynchronous Speculation," SpringSim Society for Computer Simulation International, pp. 467-474, 2008

[13] T. Ragunathan and P. Krishna Reddy. "Improving the Performance of Read-Only Transactions Through Speculation," DNIS 2007, LNCS 4777, pp. 203.221, 2007.

[14] K. Eswaran, J. Gray, R. Lorie and Traiger, " The notions of consistency and predicate locks in database systems", Communications of the ACM 19(11), 624633 , 1976.

[15] T. Ragunathan and P. Krishna Reddy. "Speculation-Based Protocols for Improving the Performance of Read-only Transactions", IJCSE, Inderscience Publications, 2010 (to appear).

[16] R.Agrawal, M.J.Carey and M.Livny. "Concurrency Control Performance Modeling: Alternatives and Implications," ACM Trans. Database Systems, vol.12, no.4, pp.609-654, Dec.1987.

[17] Kwok-Wa Lam, Sang H. Son, Victor C.S. Lee and Sheung-Lun Hung. 1998. "Using Separate Algorithms to Process Read-Only Transactions in Real-Time Systems." In Proceedings of the IEEE Real-Time Systems Symposium, 50-59.