データベースシステム 119−59
（１９９９．７．２３）

高並行性を考慮した投機的入れ子トランザクション処理

P. Krishna Reddy、喜連川 優

概要

本論文では投機的トランザクション処理における並行制御プロトコルの改善手法を提案する。提案する投機的入れ子ロック (SNL:speculative nested locking) プロトコルでは、副トランザクションが (実行後イメージを作り出す) データオブジェクトに対する仕事を終了した時には常に親がロックを継承するようになっている。待機中の副トランザクションは先行する副トランザクションの実行前後両方のイメージにアクセスすることによって投機的実行を行う。待機中のトランザクションは先行する副トランザクションの終了決定に基づき適切な実行を選択する。その結果、輻輳するトランザクション間の並列性が高められる。SNL アプローチでは投機的実行を支援するためにその分の処理と主記憶がさらに要求されることになる。この論文では SNL アプローチを提案し、このアプローチによって Moss の入れ子ロッキングプロトコルと比較してトランザクション内、トランザクション間の両方における並行性がいかに高められるかについて説明する。このアプローチでは限られた資源環境において主記憶と CPU 資源のバランス化を図ることにより並行性を高めている。

Increasing Concurrency of Nested Transactions Through Speculation

P. Krishna Reddy and Masaru Kitsuregawa
Institute of Industrial Science, The University of Tokyo
{reddy, kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract

We propose an improved concurrency control protocol for nested transactions based on speculation. In the proposed speculative nested locking (SNL) protocol, whenever a sub-transaction finishes work with a data object (produces after-image), it's parent inherits the lock. The waiting sub-transaction carries out speculative executions by accessing both before- and after-images of preceding sub-transaction. The waiting transaction selects appropriate execution after termination of preceding sub-transaction. As a result, parallelism among conflicting transactions increases. The SNL approach requires both extra processing power and main memory to support speculative executions. In this paper, we presented SNL approach and explained how it increases both intra- and inter-transaction concurrency as compared to Moss's nested locking protocol. This approach increases concurrency by trading main memory and CPU resources under limited resources environments.

Index terms Concurrency control, nested transactions, locking, serializability, transaction processing

1 Introduction

In nested locking (NL) protocol proposed by Moss [16], each leaf-transaction follows two-phase locking (2PL) protocol [8, 9] for concurrency control. If a sub-transaction obtains a write lock, its parent inherits the lock only after its termination, as per 2PL rules. The longer lock holding time degrades performance of the database system. In this paper we propose speculative nested locking (SNL) protocol to increase concurrency by employing extra computing resources. In SNL whenever a sub-transaction T_i finishes work with a data object (produces after-image), it's parent inherits the lock. The waiting (sub)transaction accesses both before- and after-images of T_i and then carries out speculative executions. However, the order is maintained; i.e., the waiting transaction selects appropriate execution only after termination of T_i. As such, there is no limitation on the number of levels of speculation but this number depends on the system's resources, such as the size of main memory and processing power. In SNL, the number of speculative executions carried out by a transaction increases exponentially as data contention increases. This approach can be extended under limited resources environments. Using SNL, both inter- and intra-transaction parallelism can be increased, without violating serializability criteria.

The work is motivated by the fact that with the continual improvement in hardware technology, we now have systems with significant amounts of processing speed and main memory, but more time is spent by transaction waiting for data (both I/O and remote data) than performing actual computations. Since the cost of both CPU and main memory is falling, extra processing power can be added to the system at reasonable cost. The strength of SNL is that it offers the potential to increase concurrency by trading extra main memory and processing resources without violating seralizability as a correctness criteria. The speculative processing is transparent to the user. Since SNL is lock-based, it could be integrated with existing applications based on Moss's nested locking protocol with little effort.

In section 2 we discuss related work. In section 3 we explain nested transaction model and nested locking protocol. In section 4, we present SNL approach. In section 5 we explain how SNL increases concurrency through an example. In section 6, we present concurrency analysis. In section 7, we extend SNL under limited resource environments. The last section consists of summary and conclusions.

2 Related work

Several protocols exist to synchronize the execution of nested transactions. Reed developed a time-stamp based

technique for nested transactions [20]. In [16], Moss presented a concurrency control algorithm using 2PL for a nested transaction environment. Nested transactions were implemented in system R [10], Argus [14], Clouds [5], Locus [17] and Eden [12]. In [19] theoretical framework has been presented to prove the serializability of synchronization protocols for nested transactions. In [18], overview of research in the area of nested transactions is given. In [11], a concept of downward inheritance is introduced to improve the parallelism within the nested transaction. In [18] the pre-write operation is introduced to increase concurrency in a nested transaction processing environment. This model allows some particular sub-transactions to release their locks before their ancestor transaction's commit. This allows other sub-transactions to acquire required locks earlier. However, it is assumed that once the sub-transaction pre-writes the value, it will not abort in future.

In the context of flat transactions, speculation has been employed in [2] to increase the transaction processing performance for real-time centralized environments that employ optimistic algorithms for concurrency control. In [13] we proposed a transaction processing approach for distributed database systems where a transaction releases locks after completing execution by employing static 2PL. The ordered sharing protocol [1], allows multiple transactions to hold conflicting locks on data objects as long as operations are executed in the same order as that in which locks are acquired. The altruistic locking protocol [21] allows transactions to donate previously locked objects, once they are done with them but before the object is unlocked. Another transaction may lock a donated object, but to ensure serializability it should remain in the wake of the original transaction. This protocol is proposed to synchronize long lived transactions.

The SNL differs from above approaches as it is a lock based approach and proposed for nested transactions. Also, in SNL approach a transaction releases locks before execution. The SNL approach trades extra resources to increase concurrency.

3 Nested transactions and locking protocol

3.1 Nested transaction model

In nested transaction model [16] a transaction may contain any number of sub-transactions, which again may be composed of any number of sub-transactions- conceivably resulting in an arbitrary deep hierarchy of nested transactions. The root transaction which is not enclosed in any transaction is called the top-level transaction (TLT). Transactions having sub-transactions are called parent transactions (PTs), and their sub-transactions are their children. Leaf-transactions (LTs) are those transactions with no children. The ancestor (descendant) relation is the reflexive transitive closure of the parent (child) relation. We will use the term superior (inferior) for the non-reflexive version of the ancestor (descendant). The children of one parent are called siblings. The set of descendants of a transaction together with their parent/child relationships is called the transaction's hierarchy. In the following, we will use the term 'transaction' to denote TLT, PT, and LT.

We employ $X, Y, \ldots$ to represent data objects. Transactions are represented by $T_i, T_j, \ldots$; where, $i, j, \ldots$ are integer values. The hierarchy of a top-level transaction (TLT) can be represented by a transaction tree. The nodes of the tree represent transactions, and the edges illustrate the parent/child relationships between the related transactions. In the transaction tree shown in Figure 1, T_1 represents TLT or root. A children of sub-transaction T_3 are T_4, T_6, and T_7, and the parent of T_3 is T_2. The properties defined for flat transactions are atomicity, consistency, isolated execution, and durability (ACID properties). In the nested transaction model, the ACID-properties are fulfilled for TLTs, while only a subset of them are defined for sub-transactions. A sub-transaction appears atomic to the other transactions and may commit and abort independently. Aborting a sub-transaction does not effect the outcome of the transactions not belonging to the sub-transaction's hierarchy, and hence sub-transactions act as fire-walls, shielding the outside world from internal failures. The durability of the effects of a committed sub-transaction depends on the outcome of its superiors. Even if a sub-transaction commits, aborting one of its superiors will undo its effects. A sub-transaction's effect becomes permanent only when its TLT commits.

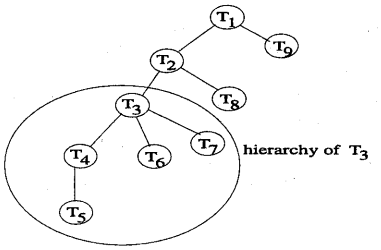


Figure 1. Example of a Transaction tree.

We assume that the PTs act as a place holders for the locks. Only LTs perform data manipulation operations and issue lock requests to obtain locks. In this paper, we consider LT as a flat transaction as defined in [3]. An LT is a representation of execution that identifies Read and Write operations and indicates the order in which these operations are executed. It is assumed that no transaction reads or writes data objects more than once. We assume that a transaction reads before it writes any data object.

Knowledge of after-image : Normally, an LT copies data objects through read operations into private working space and issues a series of update operations. We assume that for any data object X, write operation is issued whenever it completes work with the data object. This assumption is also adopted in [1, 21].

3.2 Nested locking protocol

In this section we will summarize the nested locking protocol proposed by Moss [16]. Conventional locking protocols offer two modes of synchronization - **Read**, which permits multiple transactions to share an object at a time, and **Write**, which gives the right to a single

transaction for exclusively accessing an object. Possible lock modes on an object are NL-, R-, and W-mode. The null mode (NL) represents the absence of a lock request for or a lock on the object. A transaction can acquire a lock on object X in some mode M; then it holds lock in mode M until its termination. Besides holding a lock a transaction can *retain* a lock. When a sub-transaction commits, its PT inherits its locks and then retains them. If a transaction holds a lock, it has the right to access the locked object (in the corresponding mode), which is not true for retained locks. A retained lock is only a place holder. A retained W-lock, indicates that transactions outside the hierarchy of the retainer can not acquire the lock, but that descendants of the retainer potentially can. That is, if a transaction T_i retains an W-lock, then all non descendants of T_i can not hold the lock in either W- or in R-mode. If T_i is a retainer of an R-lock, it is guaranteed that a non-descendant of T_i can not hold the lock in W-mode, but potentially can in R-mode. As soon as a transaction becomes a retainer of a lock, it remains a retainer for that lock until it terminates.

The NL rules for a transaction T_i are as follows.

- **NL1 :** T_i may acquire a lock in R-mode if
  - -- no other transaction holds the lock in W-mode, and
  - -- all transactions that retain the lock in W-mode are its ancestors.
- **NL2 :** T_i may acquire a lock in W-mode if
  - -- no other transaction holds the lock in W- or R-mode, and
  - -- all transactions that retain the lock in W- or R-mode are its ancestors.
- **NL3 :** When T_i commits, its parent inherits its (held or retained) locks. After that, T_i's parent retains the locks in the same mode (W or R) in which T_i held or retained the locks previously.
- **NL4 :** When T_i aborts, it releases all locks it holds or retains. If any of its superiors holds or retains any of these locks they continue to do so.

Note that the inheritance mechanism (Rule NL3) may cause a transaction to retain several locks on the same object. In such a case, a transaction retains a most restrictive lock.

4 Speculative nested locking

4.1 Lock modes and commit dependency

In the SNL approach, the duration of lock in W-mode is partitioned into three modes, EW- (**E**xecutive **W**rite)- , PSW(**P**assive **S**peculative **W**rite)- and ASW (**A**ctive **S**peculative **W**rite)-mode. The transactions only request either R- or EW-mode lock. Also, note that an LT holds a lock, and a PT (or TLT) retains a lock.

An LT requests lock in R-mode to read a data object and in EW-mode both to read and write a data object. An LT converts lock from EW-mode to PSW-mode whenever it produces after-image and holds the lock in the same mode until its termination. Whenever an LT holds lock in PSW-mode, its parent inherits and retains a lock in an ASW-mode. Let T_j be a PT and retained lock in ASW-mode on a data object. Whenever all sub-transactions of T_j finish work on a data object, T_j converts lock from ASW-mode to PSW-mode and retains in the same mode. Whenever T_j retains a lock in PSW-mode its parent inherits and retains lock in ASW-mode.

For X, a retained ASW-lock indicates that descendants of the retainer potentially can acquire lock in EW-mode, but all non descendants of the retainer can acquire a lock only after all its sub-transactions finish work with that object. As soon as a transaction becomes a retainer of a lock, it remains the retainer for that lock until it terminates. Similarly, a hold/ retained lock in PSW-mode indicates that any other transaction which accesses X should form commit dependency with T_j. If T_i forms commit dependency with T_j, T_i is committed only after termination of T_j. The **commit dependency rules** in SNL are as follows.

- If T_i obtains the lock in EW-mode, while T_j holds/retains a lock in R-mode, or PSW-mode on a data object, T_i forms a commit dependency with T_j.
- If T_i obtains the lock in R-mode, while T_j holds/retains a lock in PSW-mode on a data object, T_i forms a commit dependency with T_j.

4.2 Speculative nested locking protocol

We first explain the data structures used in SNL protocol.

- $tree_X$: We employ a tree data structure to organize the uncommitted versions of a data object produced by speculative executions. The notation $X_q (q \geq 1)$ is used to represent the q^{th} version of X. For a data object X, its tree is denoted by $tree_X$. It is a tree with committed version as the root and uncommitted versions as the rest of the nodes.
- $Depend\_set_i$: $Depend\_set_i$ is a set of transactions with which T_i has formed commit dependencies for all the data objects it has accessed.

We now present SNL synchronization rules. Each data object X is organized as a tree with X_1 as a root. We use the notation T_{im} to represent the m^{th} $(m \geq 1)$ speculative execution of T_i. Note that deadlock handling [16] algorithms needs to be initiated whenever a deadlock occurs.

- **SNL1 : Lock acquisition** Note that during lock acquisition whenever T_i forms a commit dependency with T_j, the identity of T_j is included in $depend\_set_i$.
  - -- Transaction T_i may acquire a lock in R-mode if
    - * no other transaction holds the lock in EW-mode, and
    - * all transactions that retain the lock in ASW-mode are ancestors of T_i and
    - * no other transaction retains the lock in ASW-mode and for each transaction that retains/holds a lock in PSW-mode,its TLT retains a lock in PSW-mode.
  - -- Transaction T_i may acquire a lock in EW-mode if
    - * no other transaction holds the lock in R- or EW-mode and

* all transactions that retain the lock in R- or ASW-mode are ancestors of T_i and
* no other transaction retains the lock in ASW-mode and for each transaction that retains/holds a lock in PSW-mode,its TLT retains a lock in PSW-mode.

- **SNL2 : Execution and inheritance**
  - **Execution** Suppose T_i be an LT, and is carrying out m speculative executions and obtains a lock in EW-mode on X. Let $tree_X$ contains n versions. Then, T_{iq} (q=1 ... m) splits into n speculative executions (one for each version of $tree_X$).
  - Inheritance The inheritance can be separated into two types: LT to PT and PT to PT.
    * **LT to PT** Whenever a transaction produces after-images during its execution, after including each after-image of X as a child to the corresponding before-image of X's tree, the lock in EW-mode is changed to PSW-mode and holds in the same mode. Next, its parent inherits and retains a lock in ASW-mode.
    * **PT to PT** Let T_j be a PT and retained lock in ASW-mode. Whenever all sub-transactions of T_j finish work on a data object, T_j converts the lock from ASW- to PSW-mode and retains in the same mode. Whenever a T_j changes the lock to PSW-mode its parent inherits and retains in ASW-mode.
- **SNL3 : Termination**
  - **Commit** A transaction T_i selects appropriate execution only after termination of all transactions in $depend\_set_i$. Each tree of locked data object is updated with after-image as the root. Its identity is removed from $depend\_set_i$ of all remaining transactions. The waiting transactions drop speculative executions carried out by reading before-images of committed transaction.
  - **Abort** A transaction T_i could abort at any time during processing. Let T_r be the TLT of both T_i and T_j. When T_i aborts, T_j is also aborted if (i) $T_j \in depend\_set_i$ or (ii) $T_i \in depend\_set_j$.

    When T_i aborts, each tree of a data object (accessed by T_i) is updated by removing after-images (with sub-trees) which were included by T_i. Its identity is removed from the $depend\_set$ of all waiting transactions. The waiting transactions drop speculative executions carried out by reading after-images of T_i.

4.3 SNLnp and SNLp approaccches

In SNL, after inheriting a lock from a sub-transaction (as per rule SNL2), a PT can not donate the locks in turn to its PT, unless all sub-transactions in its hierarchy finish the work with corresponding data object. Without having knowledge of data objects accessed by its sub-transactions, a lock is held by a PT till termination of all transactions in its hierarchy. Therefore, based on the prior knowledge of data objects accessed by a transaction, SNL can adaptively operate in two modes: SNLnp (SNL-no-predeclaration) and SNLp (SNL-predeclaration).

In SNLnp mode, a lock is held by a PT till termination of all transactions in its hierarchy. Therefore, SNLnp increases only intra-transaction parallelism (up to only one level in the nested hierarchy). Also, a transaction abort does not lead to cascading aborts. The waiting transaction selects appropriate execution.

On the other hand, in SNLp-mode, once inherited the speculative locks from a LT T_j, its PT T_i checks if any of its other sub-transactions requires access to corresponding data object. If none, then T_i's PT inherits the locks on the corresponding data object. In this way, speculative locks can be donated out side nested transaction before its termination. As a result, SNLp increases both intra- and inter-transaction concurrency. Also, in SNLp, since locks are eagerly propagated to outside TLT to increase concurrency, SNLp is robust with aborts only with respect to TLTs. That is, if a transaction aborts, other TLTs which accessed after-images of aborted transaction need not be aborted. But, within TLT, other LTs which have accessed data objects of aborted transaction have to be aborted to ensure serial consistency.

The two variations of SNL are appropriate for different kind of environments depending on the data contention. SNLnp is appropriate in environments where data contention is high within TLT. On the other hand, if data contention within TLT is low, predeclaration property of SNLp significantly enhances both itra- and inter-transaction concurrency.

5 Example

Consider follwing two transactions T_1(T_2 (T_4 : $\{V, X\}$ T_5 : $\{X, Y\}$), T_3 : $\{U, V\}$) and T_6(T_7 : $\{U\}$, T_8 : $\{Z\}$) which are simulataneously entered into the system (see Figure 2). Consider that all request locks in EW-mode. The processing employing NL and SNLp is as follows.

- **NL** Figure 3(a) depicts the processing employing NL. T_4 obtains lock on X only after termination of T_5. Similary T_3 obtains lock on V only after the abort of T_4 or the commit of both T_4 and T_2. Similarly, T_7 obtains lock on U only after the abort of T_3 or the commit of both T_3 and T_1.
- **SNL** Figure 3(b) depicts the processing with SNLp. At first, T_5, T_4, T_3, and T_8 obtain locks in EW-mode on X, V, U, and Z respectively. Whenever T_5 and T_4 produces after-images of X and V, respectively, T_2 inherits the lock in ASW-mode and whenever T_3 produces after-images of U, T_1 inherits the lock on ASW-mode. Next, T_4 obtains lock in EW-mode and carries out two speculative executions by accessing both before- and after-images of X. Due to pre-declared assumption (since T_5 would not access V), T_2 decides that it has finished work with V and therefore changes lock on V from ASW- to PSW-mode. Then, T_1 inherits lock in ASW-mode on V and retains in the same mode. Next, T_3 obtains lock on V in EW-mode and carries out two speculative executions. Due to predeclaration assumption (no other transaction would

access U), T_1 decides that it has finished work on U, and converts lock from ASW- to PSW-mode. T_7 carries out two executions by accessing before- and after-images of X.

In this way SNLp increases both intra- and inter-transaction parallelism of nested transactions.

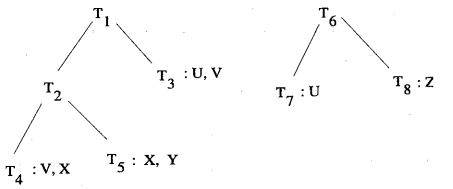
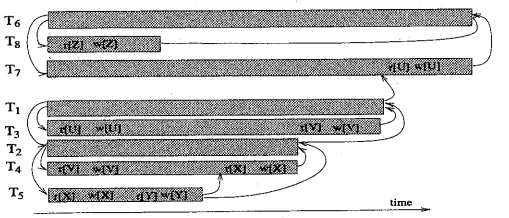
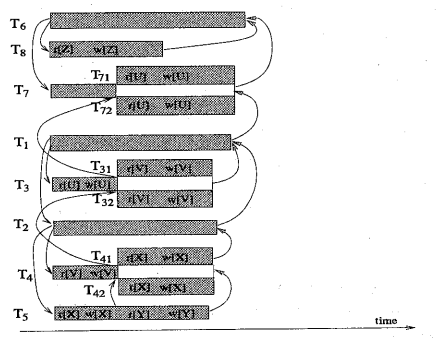


Figure 2. Depiction of T_1 and T_6.



(a) NL appraoch



(b) SNL approach

Figure 3. Depiction of processing (a) NL (b) SNLp

6 Concurrency analysis

In SNL approach, speculative executions of transaction depends on its speculation level and number of data objects it conflicts with other transactions. We first define the term *speculation level* which is used to quantify the parallelism that could be achieved using SL.

Definition. Speculation level: For T_j, the speculation level is denoted by ρ_j. If T_j executes without conflict, ρ_j = 0. Let T_j speculatively reads a set of data objects, say, *spec_set* updated by n transactions. Each $X \in spec\_set$ is updated by some T_k, at speculation level ρ_k. Let ρ_{max} be the maximum of all ρ_k, where T_k has updated a data object in *spec_set*. Then, $\rho_j = (\rho_{max} + 1)$.

Now we derive relationship between speculative executions of a transaction and its speculation level.

Let T_i conflicts on m ($m \geq 0$) data objects with other transactions. When T_i obtains lock on first data object with v_1 nodes in its tree, it carries out v_1 executions. When it accesses the second object having v_2 nodes in its tree, each one of the v_1 executions carries out v_2 executions. Following this, after accessing all m objects, the total number of speculative executions carried out by T_i = $\prod_{k=1}^{m} v_k$.

Note that, if a transaction has no conflict with other transaction on the k^{th} data object, v_k is one. Otherwise, if a transaction obtains the lock on k^{th} data object in speculative mode (some other transaction has updated the object tree), $v_k > 1$. For the sake of simplicity, let c be the mean of number of data objects that a transaction conflicts, ρ be the mean speculation level. Also, let v_ρ be the mean of number of versions in the tree of a data object and N_ρ be number of executions at level ρ. Then,

$$N_\rho = v_\rho^c \quad (1)$$

For the sake of simplicity we make two worst-case assumptions. First, we assume that a transaction requests only write locks and releases these locks after completing execution. And second when a transaction carries out N_ρ executions, N_ρ distinct versions are included to the tree of each data object it accessed after its execution. Then, the number of versions at the next level $v_{\rho+1}$ is given below.

$$v_{\rho+1} = v_\rho + N_\rho \quad where \quad v_0 = 1, N_0 = 1 \quad (2)$$

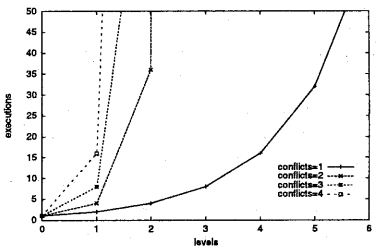


Figure 4. Number of levels versus speculative executions

From Equations 1 and 2, given N and c we can estimate ρ. Database systems vary with respect to available resources and data contention. We discuss how SNL increases concurrency in such environments.

A **Single conflict (Hot spots)** From Equations 1 and 2, with c=1, the relationship between ρ and N_ρ is, $N_\rho = 2^\rho$. Therefore, $\rho = \log N_\rho$. From Fig. 4, it can be observed that, in single conflict environments, even we support either speculative executions for a transaction (i.e., with N=8 and c=1), concurrency can be increased up to three speculation levels.

B **Multiple conflicts (long transactions)** From Equation 3, with ρ=1, the relationship between c and N_1 is, $N_1 = 2^c$. Thus in database environments in which majority of transactions conflict on multiple data objects, if we support 2^c speculative executions for a transaction, concurrency can

be increased up to one speculation level. So, SL achieves 1-level speculation with manageable extra resources. However, at multiple conflicts ($c > 2$) and higher speculation levels ($\rho > 2$), the value of N explodes.

7 Extension of SNL under limited resource environments

In SNL approach, the number of speculative executions of a transaction increases exponentially as data contention increases. Since each speculative execution needs separate work space, the size of main memory available in the system limits the number of speculative executions that can be carried out. With this limitation, processing cost may not be considered as a considerable overhead as current technology provides high speed parallel computers at low cost. Under limited resource environments the number of speculative executions of a transaction is limited as follows. Let amount of memory to carry out single execution is one unit. Based on the available memory units, we decide the feasible number of speculative executions that could be carried out by a transaction. During processing if the number of executions crosses the fixed value, the transaction is either put to wait or aborted.

8 Summary and conclusions

In this paper we have proposed concurrency control approach based on speculation for nested transactions. In a (sub)transaction releases lock on the data object when it produces after-image. The SNL approach increases concurrency without violating the serializability criteria. It requires extra processing and memory resources to carry out speculative executions. By trading extra resources SNL increases concurrency under limited resources environments. Through example we illustrated how SNL increases concurrency as compared to nested locking. Also, we analyzed how SNL increases concurrency under limited resource environments. As a part of future work, we evaluate the performance through simulation experiments.

Acknowledgments

This work is partially supported by Grant-in-Aid for Creative Basic Research # 09NP1401: "Research on Multimedia Mediation Mechanism for Realization of Human-oriented Information Environments" by the Ministry of Education, Science, Sports and Culture, Japan and Japan Society for the Promotion of Science, Japan.

References

[1] D.Agrawal, A.El Abbadi, and A.E.Lang, The performance of protocols based on locks with ordered sharing, *IEEE Transactions on Knowledge and Data Engineering*, vol.6, no.5, October 1994, pp. 805-818.

[2] Azer Bestavros and Spyridon Braoudakis, Value-cognizant speculative concurrency control, *proc. of the 21th VLDB Conference, 1995*, pp. 122-133.

[3] P.A.Bernstein, V.Hadzilacos and N.Goodman, *Concurrency control and recovery in database systems*(Addison-Wesley, 1987).

[4] P.K.Chrysanthis and K. Ramamritam. A formalism for extended transaction models. In proc. of 17th VLDB conference, 1991.

[5] P.Dasgupta, R.Liblanc Jr, and W.Appelbe. The clouds distributed operating system. In proceedings of 8th International Conference on Distributed Computing Systems, San Jose, CA, 1988.

[6] C.T. Davies, Data processing spheres of control, IBM Systems Journal. 17(2), pp. 179-198, 1978.

[7] A.K.Elmagarmid (ed.), Database transaction models for advanced applications, Morgan Kaufmann, 1992.

[8] K.R.Eswaran, J.N.Gray, R.A.Lorie, and I.L. Traiger, The notions of consistency and predicate locks in a database system, Communications of ACM, November 1976.

[9] J.N.Gray, Notes on database operating systems: in operating systems an advanced course, *Volume 60 of Lecture Notes in Computer Science*, 1978, pp. 393-481.

[10] J.Gray. et all. The recovery manager of the system R database manager, ACM Computing Surveys, 13, pp.223-244, 1981.

[11] T.Harder and K.Rothermel, Concurrency control issues in nested transactions, The VLDB Journal, vol.2, no.1, pp.39-74, 1993.

[12] W.H.Jessop, D.M.Jackobson, J.Baer, and C.Pu. An introduction to the Eden transactional file system. In proceedings of 2nd IEEE Symposium on Reliability in Distributed Software and Database Systems, Pittsburgh, PA, 1982.

[13] P.Krishna Reddy and Masaru Kitsuregawa, Improving performance in distributed database systems using speculative transaction processing,*in proceedings of The Second European Parallel and Distributed Systems conference (Euro-PDS'98)*, 1998, Vienna, Austria.

[14] B.Liskov, Distributed computing in Argus, Communications of ACM, 31, pp.300-312, 1988.

[15] S.K.Madria, A study of the concurrency control and recovery algorithms in nested transaction environment, The Computer Journal, vol. 40, no.10, pp.630-639, 1997.

[16] J.E.B.Moss, Nested transactions: An approach to reliable distributed computing. Cambridge, mass, MIT Press, 1985.

[17] E.T. Mueller, J.D.Moore, and G.Popek. A nested transaction mechanism for Locus. In proceedings of 9th ACM Symposium on Operating Systems Principles, Bretton Woods, USA, 1983.

[18] S.K.Madira, S.N.Maheswari, B.Chandra and Bharat Bhargawa, Crash Recovery algorithm in open and safe nested transaction model, *Lecture Notes in Computer Science*, vol. 1308, Springer-Verlag, 1997, pp. 440-451.

[19] R.F.Resende, Synchronization in nested transactions, Ph.D thesis, University of California, Santa Barbara, 1994.

[20] D.P.Reed, Naming and synchronization in a decentralized computer system. Ph.D thesis. Technocal report MIT/LCS/TR-205, MIT Laboratory for Computer Science, MA.

[21] K.Salem, H.Garciamolina and J.Shands, *Altruistic locking*, ACM Transactions on Database Systems, vol. 19, no.1, March 1994, pp. 117-165.