경영정보학연구
제6권 1호
1996년 6월

느슨히 결합된 데이타베이스 시스템에서 최적의 저장점 유도

최 재 화[1] 김 성 언[2]

Optimal Savepoint in a Loosely-Coupled Resilient Database System

This paper investigates the performance improvement opportunities through a resiliency mechanism in the distributed primary/backup database system. Recognizing that a distributed transaction executes at several servers during its lifetime, we propose a resiliency mechanism which allows continuous transaction processing in distributed database server systems in the presence of a server failure. In order to perform continuous transaction processing despite failures, every state change of a transaction processing can be saved in the backup server.

Obviously, this pessimistic synchronization may give more burdens than benefits to the system. Thus, the tracking needs not be done synchronously with the transaction progress. Instead, the state of all transaction processing in a system is saved periodically. This activity is referred to as a savepoint. Then, the question is how often the savepoint has to be done. We derive the optimal savepoint to identify the optimization parameters for the resilient transaction processing system.

1) 단국대학교 경영학과
2) 대구효성가톨릭대학교 경영학과

I. INTRODUCTION

Database applications are evolving to the point where failure to provide service is unacceptable [Gray86]. This increased demand for continuous operation of database systems in recent years makes it necessary to maintain backup systems, which allow the system to continue its operation even in the presence of primary failures [Borr84, Burk90, IBM87, Lyon90]. The use of backups allows both hardware and software failures to be masked at the application level. The goal is to make application services available to users despite hardware or software failures by backup systems which run on distinct hardware processors and maintain information about the global service state.

In a parallel/distributed database server system, most transactions are distributed transactions which are processed by multiple database servers. For nondistributed transactions, the effect of failures is limited to that database server and it is handled by the transaction recovery mechanism locally. In the conventional distributed system, when failures occur during transaction processing, but before the commit operation, then the entire transaction is simply restarted. However, aborting the whole effort of processing distributed transactions has a significant performance penalty. This is particularly important for long-lived transactions as the loss of computing time is likely to be significant.

However, the performance improvement can be partly accomplished by minimizing the processing efforts lost due to failures [Shet87, LeeY87]. Recognizing that a distributed transaction executes at several servers during its lifetime, we propose a resiliency mechanism which allows continuous transaction processing in a distributed primary/backup database server system in the presence of a server failure. We investigate the performance improvement opportunities by reducing the loss of computing time due to the failure. The approach considers a transaction component within a server as a unit of recovery and tracks the progress of the execution of a transaction on distributed servers.

In order to perform continuous transaction processing despite failures, every state change of a transaction processing can be saved in the backup server. Obviously, this pessimistic synchronization may give more burdens than benefits to the system. The tracking needs not be done continuously with the transaction progress. Instead, the

state of all transaction processing in a system is saved periodically. This activity is referred to as a savepoint. Then, the question is how often the savepoint has to be done. We derive the optimal savepoint for a resilient transaction processing system.

The purpose of this paper is to analyze the performance improvement opportunities with the resiliency mechanism and to identify the optimization parameters for the resilient transaction processing system. In the next section we describe the system architecture and a model for the distributed transaction processing. In Section 3, the implementation of the continuous transaction processing system is briefly described. Section 4 analyzes the optimal savepoint of the resiliency mechanism. The conclusion is given in the final section.

II. The X* System

We are developing a distributed primary /backup database server system, called X*. The system is basically a number of clustered database servers as shown in Figure 1.

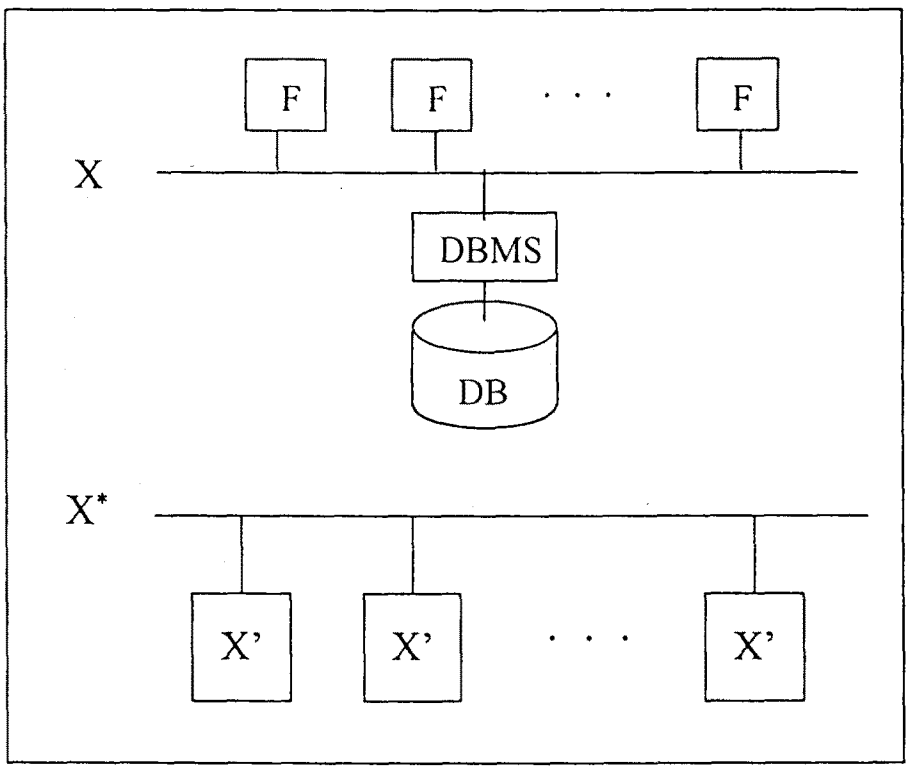


Figure 1. The X* System

In the figure, a server is represented by X′ which is an extension of a conventional database system, X, serving front end users, F, and the entire system is represented by X* which is a multiple of X′. The basic component X′ is a database server based on the client/server architecture and is an extension of X, or XDB, a relational database system commercially available and is fully compatible with IBM's DB2. Our project is to integrate existing XDB database servers running under OS/2 and interconnected through a local area network [XDB91]. One of the goals of X* is not only to provide high level availability but also to achieve high performance. The system is intended to provide the capability of continuous transaction processing in the presence of failures.

X* differs from other primary/dedicated backup systems in the sense that none of the database server is a dedicated backup for anther database server. A database server can become a backup database server for another server simply by keeping partially replicated backup data. The primary/dedicated backup system can not survive the simultaneous failure of both primary and backup. In order to sustain multiple failures, X* may keep two or more identical copies of each relation in the system, one being the primary copy and the others being backup copies. Thus, one or more database servers become backup database servers for the server with the primary copy. In this way we can have a distributed cross backup database server system on a local area network. The survival of at least one backup copy allows the distributed transaction processing to continue.

Each database server has a copy of the global directory. Thus, every database server in the system knows which database server has the primary and backup copies of a table. All read and write requests go to the primary copy only. Propagating updates to the backup copy is done synchronously during the transaction commit phase by the primary server which contains the primary copy.

We define a failure without distinction as the total loss of processing power and access to data at one server for a period of time, and possibly permanent loss of data at that server. When a server fails, all servers must be informed of the failure. The global directories are updated. One of the backup copies is now the primary copy. The new primary copy is used for all transaction processing. A certain number of backup copies may be always main-

tained. However, in order to simplify the discussion in the paper, we consider only one backup server for a primary server.

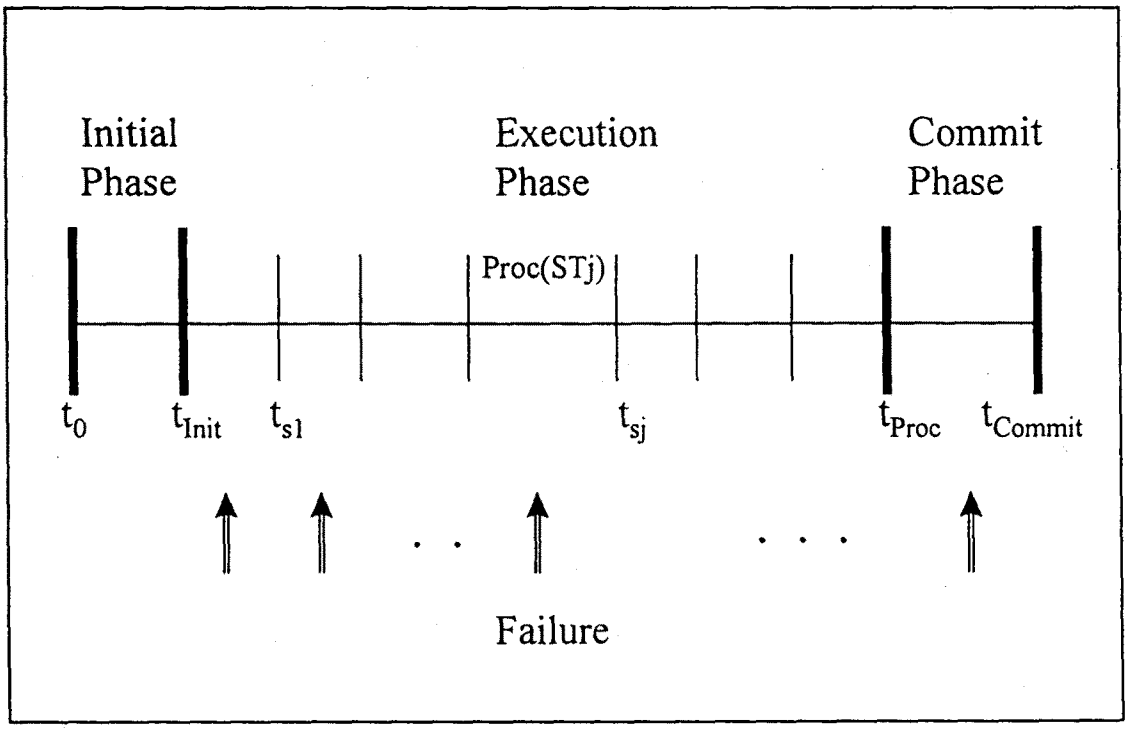


Figure 2. Transaction Processing Phases and Failure Window

As shown in Figure 2, distributed transaction processing can be divided into three phases: initial phase, execution phase, and commit phase. During the initial phase a transaction schedule is developed. In the transaction schedule, subtransaction dependency is represented by a data flow graph. The execution phase consists of one or more sequential steps that are divided according to the subtransaction dependency. In each step of the execution phase one or more subtransactions are processed, possibly in parallel. The commit phase involves the distributed two-phase commit.

In distributed transaction processing, subtransactions are executed in multiple servers. Figure 2 shows that failures, from various reasons and at different nodes, may occur at any time during the distributed transaction processing. The effect of a failure during the commit phase is on the consistency of the distributed database. However, the effect of a failure before the commit phase is on the performance.

Restart of the transaction entirely will lose valuable computing time. From the figure 2, if the effect of a node failure can be contained to a processing step, the performance loss will be reduced. This confinement of the effect of failure is the key idea for the resiliency mechanism with the distributed primary/backup database server system.

III. Resilient Transaction Processing System

The transaction processing in each node X′ involves three components of a database server. The global transaction manager(GTM) controls the execution of transactions. GTM decomposes a transaction (T) into subtransactions(ST) and schedules them for execution. The local transaction manager(LTM) interacts with the database manager(DM) for executing subtransactions and distributed commits. DM performs retrieval of data from the database and update to the database, and controls the concurrent accesses to the database. The inter−server communications are done by the communication manager (CM) through the message passing scheme.

For a distributed transaction processing a server plays the role of either GTM, LTM, or both. We refer to a server with the role of GTM and a server with the role of LTM as SGTM and SLTM respectively. For each transaction there is only one SGTM at which the transaction is initiated. A distributed transaction processing involves more than one SLTM. Incidentally, for the nondistributed transaction, SGTM and SLTM are the same server. Also, we refer to a backup server for SGTM and a backup server for SLTM as SBGTM and SBLTM respectively.

We classify the activities in a distributed transaction processing into two processing modes : transaction processing mode and restart processing mode.

During the transaction processing mode the system not only executes transactions as in the conventional distributed database systems but also maintains status information about transactions being processed in the system. This information is collected and maintained in the structure called the transaction table. In the event of failure, restart processing mode is invoked to perform continuous transaction processing.

In the following we first describe the structure and content of transaction tables. Then, we explain the general algorithm of two processing modes. Here, we

assume the tracking is done synchronously with the transaction progress. The optimal way of keeping information about transaction processing is described in Section 4.

3.1 Transaction Table

The description of the resiliency mechanism starts with the concept of transaction state. As transaction processing progresses, the state of a transaction is changed. After the initial phase is completed by SGTM without failure, the state of the transaction becomes "initiated". During the execution phase the state of a subtransaction is changed to "processed" as the completion of each subtransaction processing is known to SGTM. Each SLTM also changes the state of a subtransaction (s) to "processed" after it completes execution. When all subtransactions are "processed", SGTM changes the state of the transaction to "prepared", meaning the transaction is ready to be committed. This state information is crucial for continuous transaction processing.

To keep track of the state of transaction processing by the system, each server maintains transaction tables : T_Table and ST_Table and LST_Table. Although the global state information in T_Table and ST_Table can be integrated into one table, we decided to store them in separate tables for clarity. T_Table containsa unique identifier of a transaction, the primary server and backup server, and the processing state for each transaction. For the purpose of restart processing, the actual transaction itself is stored in the T_Table.

ST_Table contains information about the global state of subtransactions. It contains a unique identifier for a subtransaction, the primary server and backup server, and the processing state for each subtransaction. For the purpose of restart processing, the actual subtransaction is stored in the ST_Table. LST_Table contains information about the local state of subtransactions. It contains the type of subtransaction, the servers and backup servers for the subtransaction and its transaction, and the processing state for each subtransaction. The actual subtransaction is stored in the LST_Table for the purpose of restarting by a backup server.

3.2 Transaction Processing

During the initial phase, SGTM saves the transaction, prepares a schedule for the transaction and starts to coordinate participating SLTMs in the processing of

the transaction. First, when SGTM receives a transaction Ti, it selects a backup server, SBGTM, to backup itself for Ti, and enters a transaction record into the T_Table. After the transaction is decomposed into subtransactions, SGTM also adds a record for each subtransaction of the transaction into the ST_Table. This phase corresponds to the period of between t_0 and t_{Init} (Figure 2 and 3). Then, subtransactions are sent to all participating S_{LTM}s.

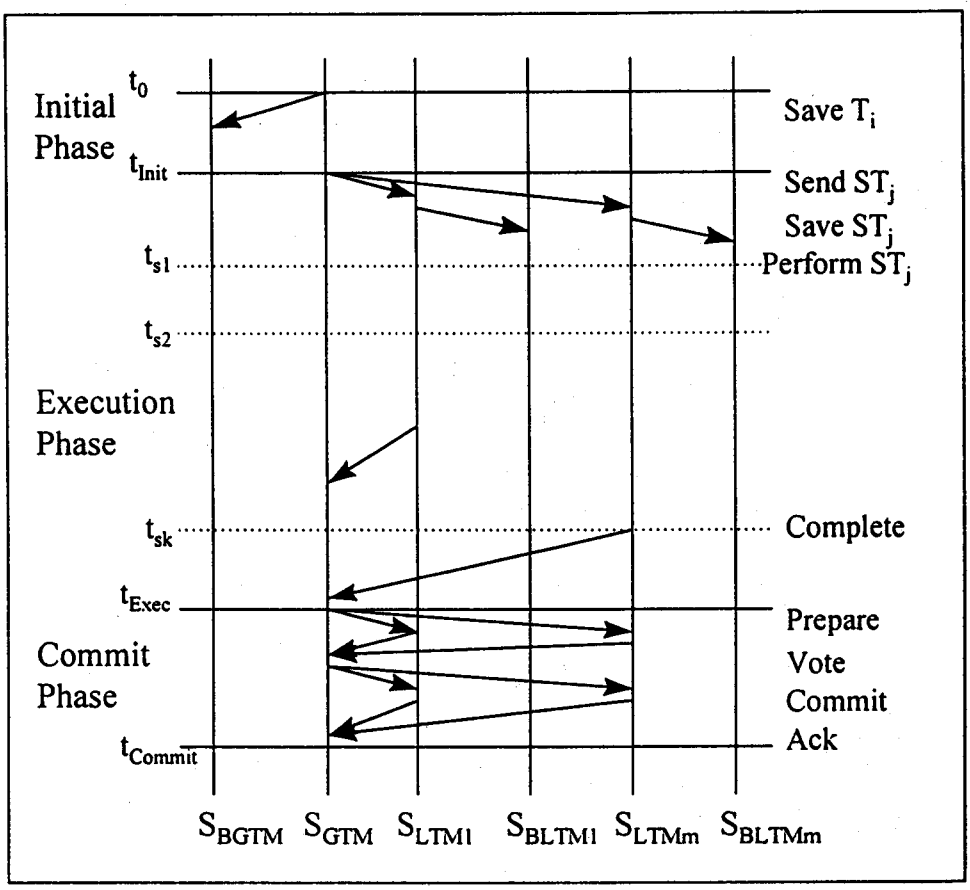


Figure 3. Resilient Transaction Processing with Cross Backup

The execution phase begins when a server receives a subtransaction ST_j of a transaction T_i. Using the dependency relationship among the subtransactions, the execution phase is further divided into execution steps. Figure 3 illustrates k execution steps in the execution phase. The processing is executed independently by all S_{LTM}s at different point of time or in parallel. Each participating server, S_{LTM}, designates

a server as its backup server, S_{BLTM} for the subtransaction, ST_i. Thus, if m S_{LTM}s participate in a transaction processing, m backup servers, $S_{BLTM}1, \ldots, S_{BLTMm}$, are elected as the backup server for subtransactions, as shown in Figure 3. Then, S_{LTM} adds a subtransaction record in the LST_Table. Once the backup recording is done, S_{LTM} asks its DM to process database requests and waits for the completion. When the execution is complete, S_{LTM} changes the state of subtransaction in LST_Table and returns results or appropriate messages to S_{GTM}. As S_{GTM} receives the reply from S_{LTM}, it also records a change in the state field of ST_Table. The execution phase completes at t_{Exec} at which k execution step finishes (Figure 3).

S_{GTM} begins the two-phase commit by sending Vote_Action message to all updating S_{LTM}s. S_{GTM} activates timeout and wait for reply from S_{LTM}s. If the decision is "commit", SGTM writes a commit log record and sends Commit_Action to updating S_{LTM}s. If the decision is "abort", S_{GTM} writes an abort log record and sends Abort_Action to updating S_{LTM}s. Each S_{LTM} acts according to S_{GTM}'s decision and removes the relevant LST_Table record. When S_{GTM} receives the Abort_Transaction from the user, it acts similarly. When S_{GTM} decides to commit or abort, the relevant records are removed from the T_Table and ST_Table.

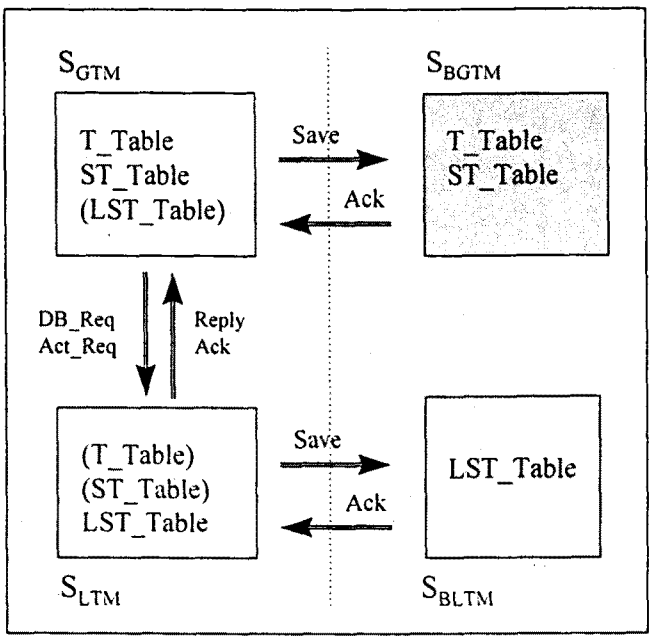


Figure 4. Information Flow between Primary and Back System

Figure 4 shows the flow of data and control messages during the transaction processing. The global state of transaction processing is reflected into T_Table and ST_Table in S_{GTM} and the local state of a particular subtransaction is reflected into LST_Table in a S_{LTM}. The resiliency is prepared at backup servers, S_{BGTM} and S_{BLTM}s. Backup for the transaction information is sent from S_{GTM} to S_{BGTM}. That is, records for relevant transactions and subtransactions in T_Table and ST_Table are same in the two servers. Backup for the subtransaction information is sent from S_{LTM} to S_{BLTM}. Also, records for relevant subtransactions in LST_Table are the same in the two servers.

3.3 Restart Processing

Once the transaction processing phase is in the commit phase, the failure is handled according to the two phase commit protocol. Thus, the discussion about restart processing in resilient transaction processing deals mainly with the initial and execution phase. In this paper we briefly describe the algorithm for restart processing. The detailed algorithm is described in a separate paper. We refer to the server which ceases to operate as the failed server and the rest of servers in the system as survived servers in the following.

A server failure is immediately informed to all database servers in the system. If a survived server finds any transaction and/or subtransaction affected by the failure, it has to perform restart processing. In order to determine what each survived server needs to do, it examines its T_Table, ST_Table, and LST_Table and identify transactions and subtransactions and its relationship with the failed server for each of them. Restart processing for the failed server depends on the state of transactions and subtransactions which are being processed at the time of failure. Four cases are possible.

Case 1 :

The survived server is a backup server for transactions that are initiated by the failed server. The survived server has to take over processing of the transactions. As the primary server has failed, the backup server becomes the primary server for those transactions, selects a new backup server for each transaction, and performs appropriate actions for the continuous transaction processing. The appropriate action is determined by examining the state information in the

T_Table record.

Case 2 :

The survived server is a backup server for subtransactions that are being executed by the failed server. The survived server has to take over processing of the subtransactions. The backup server becomes the primary server for those subtransactions, selects a new backup server, and performs appropriate actions for the subtransaction processing. The appropriate action is determined by examining the value of state and the type of subtransaction in the LST_Table record.

Case 3 and 4 :

The failed server is the backup server for the transactions and subtransactions processed by the survived server. Thus, new backup servers for the affected transaction and subtransactions have to be selected. A new backup server will be created by copying transaction records from the primary server.

These processes for restart processing should have higher priority than new transactions that entered the system after the failure of a server. This will be needed to provide fast response time. Since the effect of a server failure is distributed to several backup servers, the task by individual backup servers is not expected to be large.

Ⅳ. Performance Study of Resilient Transaction Processing System

4.1 Optimal Savepoint

The primary server sends transaction table records to the backup server to keep the backup server informed of its current state. Ideally, these records in the backup server have to be identical to the ones in the primary. However, it may be costly to send each transaction table record to the backup server whenever a state change is made to any transaction table. Thus, to reduce the overhead, an alternative is to collect transaction table records and send them as a batch. Transaction table records are saved into the backup server at synchronization points. We refer to the time to save transaction records savepoint cost.

If every state change of each transaction processing is saved in the backup system, the resiliency mechanism can perform continuous transaction processing without any restart processing in the event

of failure. However, in order to minimize the overhead, we save redundant information periodically. Restart processing may be necessary for those portions of processing which are completed, but not reflected in the backup server. We call this loss due to periodic savepointing reprocessing cost.

A proper interval between savepoints must be found. We refer to the time between two savepoints as the intersavepoint. If the intersavepoint is too small, savepointing overhead is high, and if the intersavepoint is too large, reprocessing overhead is high. The problem is to choose the optimal savepoint time which minimizes the sum of savepoint cost and reprocessing cost.

In order to derive optimal intersavepoint, we borrow the model for the determination of optimal checkpoint [Youn74, Chan75]. Based on the model, an optimal intersavepoint formula is derived. The model consists of the following assumptions and definitions.

Assumptions and Notations

The following assumptions are made:

1. Failures occur at any time. A Poisson distribution of failure is assumed.
2. The time required to reprocess affected transactions is proportional to the number of transactions in the transaction table since the savepoint.
3. The transaction arrival rate is constant at all times.
4. No failure occurs during reprocessing. This assumption is reasonable since reprocessing time and savepoint time are small compared to the mean time between failure(MTBF).
5. Savepoints and reprocessing are assumed to take a short time period compared to the intersavepoint time.

In addition, we define the following definitions and notations:

SP — expected time to perform a savepoint

Z — time between savepoints

$C_{RP}(Z)$ — expected time spent for reprocessing given Z

$C_{OH}(Z)$ — total overhead time associated with savepoint and reprocessing given Z

$C_{OH}(Z) = SP + C_{RP}(Z)$

$\Phi(Z)$ — expected unit time overhead given Z

$\Phi(Z) = C_{OH}(Z)/Z$

β — the failure rate of a server, i.e., $\beta = 1/MTBF$, where MTBF is the mean time between failure.

μ — arrival rate ; μ transactions per unit time

τ average transaction processing rate ; τtransactions per unit time

Γ reprocessing(processing) rate ; Γ transactions per unit time
$\Gamma = \mu/\tau$

t time since last savepoint, $0<t<Z$

$C_{RP}(t)$ expected reprocessing time given that a failure occurs t time units after last savepoint
$C_{RP}(t) = \Gamma t$

$C^{RP}(x)$ expected cost of reprocessing of interval [x, Z], i.e., from x to next savepoint
$C^{RP}(Z) = 0$,
$C^{RP}(0) = C_{RP}(Z)$; expected cost of reprocessing between savepoints

Derivation of Optimal Intersavepoint

First, we derive an expression for the expected reprocessing cost of an arbitrary interval, $C^{RP}(x)$, in terms of failure rate β and reprocessing rate Γ given an occurrence of failure at random time. Let us consider the interval $(t, t+\Delta t)$.

Assume that a failure happens during the interval with the probability of $\beta\Delta t$, where β is the failure rate as defined earlier. The expected reprocessing cost, $C^{RP}(t+\Delta t)$, is expressed as

$$C^{RP}(t+\Delta t) = \beta\Delta t[C_{RP}(t)+C^{RP}(t)]+(1-\beta\Delta t)C^{RP}(t).$$

The first term represents the expected cost of reprocessing when the failure occurs during the interval with the probability of $\beta\Delta t$. It consists of the expected reprocessing time at the time of failure and the expected cost of reprocessing for the period between the failure point and the next savepoint. The second term represents the expected cost of reprocessing when there is no failure during the interval. Taking the limit as Δt goes to 0, we get $(\sigma/\sigma t)\ C^{RP}(t)=\beta C_{RP}(t)$, for $0\le t\le Z$. As $C_{RP}(t)=\Gamma t$ for the period between the last savepoint and t, by replacing $C_{RP}(t)$ with Γt, we get $(\sigma/\sigma t)\ C^{RP}(t)=\beta\Gamma t$. Thus, we have $C^{RP}(t)=\beta\Gamma t^2/2$.

Next, we derive the expected unit time overhead, $\Phi(Z)$, in terms of failure rate β and reprocessing rate Γ. From the expected reprocessing cost expression above, we have $C^{RP}(t=0)=C_{RP}(Z)=\beta\Gamma Z^2/2$. And $C^{RP}(t=Z)=0$. By definition $\Phi(Z)=C_{OH}(Z)/Z = (SP+C_{RP}(Z))/Z$. By replacing $C_{RP}(Z)$ with $\beta\Gamma Z^2/2$, we have $\Phi(Z) = SP/Z + \beta\Gamma Z/2$.

Now, the following theorem shows the optimal savepoint value which minimizes the unit time overhead.

Theorem

The optimal value of intersavepoint time Z^* which minimizes the unit time overhead is

$$Z^* = (2SP/\beta\Gamma)^{1/2}.$$

The corresponding value of unit time overhead Φ^* is

$$\Phi^* = (2\beta\Gamma SP)^{1/2}.$$

Proof

We have $\Phi(Z) = SP/Z + \beta\Gamma Z/2$. By taking the second derivative of with respect to Z, we get

$$\sigma^2\Phi/\sigma Z^2 = 2SP/Z^3 > 0.$$

Thus, Φ has a unique local minimum. By taking the first derivative of with respect to Z and equating to 0, we have

$$Z^* = (2SP/\beta\Gamma)^{1/2} \text{ and}$$

$$\Phi^* = (2\beta\Gamma SP)^{1/2}.$$

Substituting $Z^* = (2SP/\beta\Gamma)^{1/2}$ in $C_{RP}(t)$, we also find that the maximum reprocessing time at optimality is $C_{RP}(Z^*) = (2SP\Gamma/\beta)^{1/2}$.

4.2 Performance Implication

The optimal intersavepoint is determined with the objective of minimizing the unit time overhead. When the intersavepoint is small, the system will incur large savepoint overhead, but small reprocessing overhead. When the intersavepoint is large, the system will incur small savepoint overhead, but large reprocessing overhead. In order to provide some ideas about determining the value of parameters to the database system implementer, we discuss the performance implication by giving example values to the parameters in the formulae we derived above. The relationships among important input parameters are shown in Figures 5−7. The graphs in the figure are generated using the following parameters:

$SP = 1 \text{ min} = 1/60 \text{ hour}$

$\beta = 1/24 \text{ hour}$

$\Gamma = 1/5.$

Figure 5 shows a graph of unit time overhead as a function of intersavepoint time. As the intersavepoint time Z changes from 1 to 24 hours, the expected unit time overhead, $\Phi(Z)$, along with its two components, SP/Z, $\beta\Gamma Z/2$ are computed and plotted. While the reprocessing overhead, $\beta\Gamma Z/2$, increases monotonically, the savepoint overhead, SP/Z, decreases rapidly and flattens. As the intersavepoint increases, the reprocessing overhead portion of the unit time overhead dominates that of the savepoint overhead. Thus, the unit time overhead, $\Phi(Z)$, decreases rapidly, stays relatively flat at optimality, and in-

crease gradually. The optimal intersavepoint Z* can be found at which the unit time overhead, $\Phi(Z)$, is at the minimum.

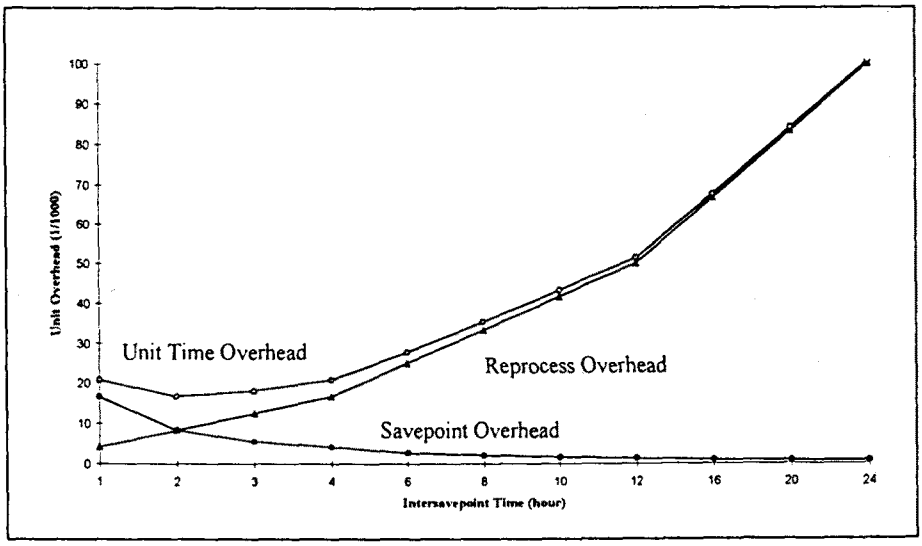


Figure 5. Unit Time Overhead

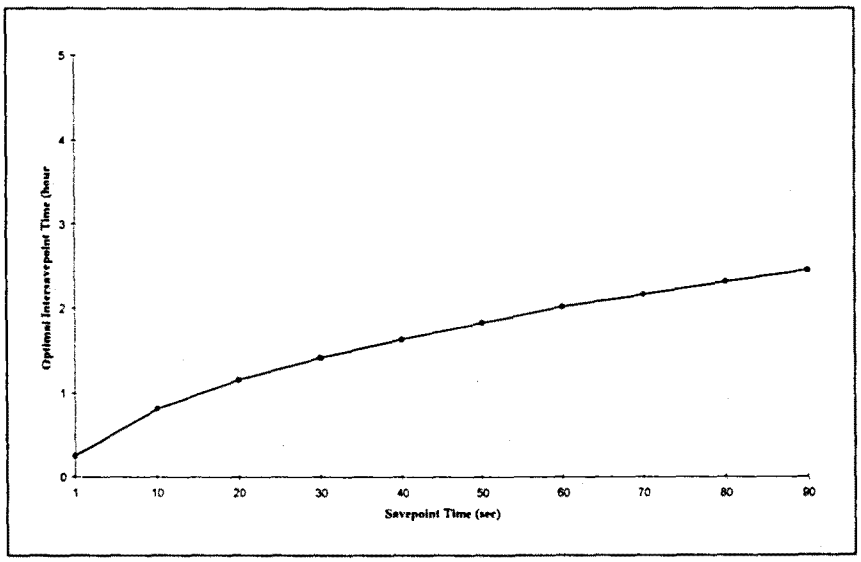


Figure 6. Optimal Intersavepoint Time

Next, we look at two relationships at the optimal intersavepoint. Figure 6 shows the optimal intersavepoint time as a function of savepoint time. As the savepoint time

(SP) increase, the optimal intersavepoint time, $Z^* = (2SP/\beta\Gamma)^{1/2}$, increases slowly. Figure 7 show the optimal unit time overhead as a function of MTBF. As the MTBF decreases, the unit time overhead, $\Phi^* = (2\beta\Gamma SP)^{1/2}$, decreases. This is because $\beta = 1/\text{MTBF}$.

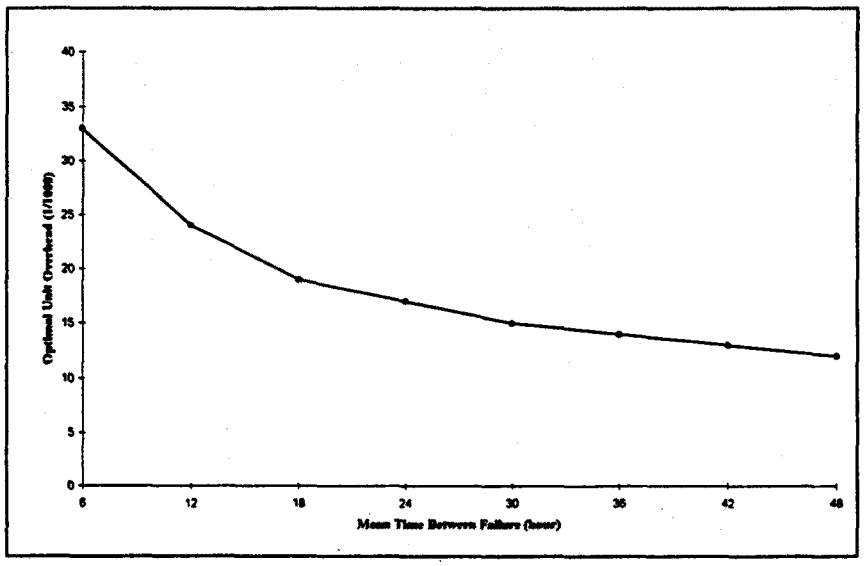


Figure 7. Optimal Unit Overhead vs. Mean Time Between Failure

V. Conclusions

In this paper we described an implementation of the resiliency mechanism which allows the continuous transaction processing on a loosely-coupled distributed database server system. To keep track of a distributed transaction processing on multiple servers, the state of a transaction processing is maintained. In the case of a server failure, survived servers restart affected transactions and subtransactions based on the state information maintained.

The mechanism does provide performance improvement over conventional distributed systems by reducing lost computing efforts in distributed transaction processing. We believe that, even with the overhead, the primary/backup approach will provide performance advantages for certain applications with long-lived transactions. The design of a beneficial resiliency mechanism requires an efficient implementation with proper system parameters. Thus, we derived the optimal inter-

savepoint in order to minimizing the sum of savepointing cost and reprocessing cost and examined the relationships among important input parameters.

References

[Borr84] Borr, A., "Robustness to Crash in a Distributed Database : A Non Shared-Memory Multiprocessor Approach," Tandem Technical Report 84.2, Sept. 1984

[Burk90] Burkes, D. L. and Treiber, R. K., "Design Approaches for Real-Time Transaction Processing Remote Site Recovery," Proc. of IEEE Compcon, 1990

[Chan75] Chandy, K. M., Browne, J. C., Dissly, C. W., and Uhrig, W. R., "Analytic Models for Rollback and Recovery Strategies in Data Base Systems," IEEE Trans. SE, Vol. SE-1, March 1975

[Gray86] Gray, J., "Why do Computers stop and What can be done about it?," Fifth ACM/IEEE Symposium on Reliability in Distributed Software and Database Systems, January 1986

[IBM87] IMS/VS Extended Recovery Facility (XRF), Technical Reference, 1987

[LeeY87] Lee, Y., Yu, P. S., and Iyer, B. R., "Progressive Transaction Recovery in Distributed DB/DC System," IEEE Transactions on Computers, Vol.C-36, No. 8, August 1987

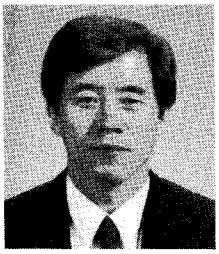
[Lyon90] Lyon, J., "Tandem's Remote Data Facility," Proc. of IEEE Compcon, 1990

[Shet87] Sheth, A. P., Singhal, A., and Liu, M. T., "Performance Analysis of Resiliency Mechanisms in Distributed Database Systems," Proc. of IEEE 1987

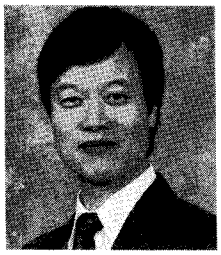
[XDB91] XDB Reference Manual, XDB Systems, Inc., Laurel, MD, 1991

[Youn74] Young, J. W., "A First Order Approximation to the Optimum Checkpoint Interval," Communication of ACM, Vol 17. Sept. 1974

◇ 저자소개 ◇



공동저자 최재화는 단국대학교 경상대학 경영회계학부에 재직중이다. 연세대학교 경제학과를 졸업하고, Drexel University에서 MBA, University of Maryland에서 정보시스템학(Information Systems) 박사학위를 취득한 후 University of Michigan-Dearborn에서 조교수로 4년간 재직하였다. 주요관심분야로는 Parallel/Distributed Database Systems, Business Application of Artificial Intelligence, Multimedia Application, Quality Issues for Information Systems 등이다.



공동저자 김성은은 서울대학교를 졸업하고 Louisiana 주립대학교에서 전산학 석사, 경영학 박사(경영정보학 전공)를 취득하였으며, 현재 대구효성가톨릭대학교 경영학과에 재직중이다. 주요관심분야는 데이타베이스 시스템과 정보시스템 설계 및 개발이다.