Providing Persistence for Sensor Data Streams by Remote WAL

Hideyuki Kawashima, Michita Imai, and Yuichiro Anzai

Information and Computer Science, Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, JAPAN

Abstract. Rapidly changing environments such as robots, sensor networks, or medical services are emerging. To deal with them, DBMS should persist sensor data streams instantaneously. To achieve the purpose, data persisting process must be accelerated. Though write ahead logging (WAL) acceleration is essential for the purpose, only a few researches are conducted.

To accelerate data persisting process, this paper proposes **remote WAL with asynchronous checkpointing** technique. Furthermore this paper designs and implements it. To evaluate the technique, this paper conducts experiments on an object relational DBMS called KRAFT.

The result of experiments shows that remote WAL overwhelms performance disk based WAL. As for throughput evaluation, best policy shows about 12 times better performance compared with disk based WAL. As for logging time, the policy shows lower than 1000 micro seconds which is the period of motor data acquisition on conventional robots.

1 Introduction

In the fields of sensor networks (SN) [1] or data stream management systems (DSMS) [2], immediate data persisting process is not considered at all. However, if it is realized, the application domains of data warehouse can be expanded to real-time or ubiquitous computing fields such as robots [3].

To realize the vision, a new type of database management system (DBMS) should be designed for managing sensor data streams. The DBMS should be able to (C1) manage massive data, (C2) providing persistence for data certainly, (C3) manage variable length tuple, and **(C4) providing persistence for data instantaneously**. Most of conventional DBMS satisfy (C1), (C2), and (C3). However, only (C4) is not satisfied yet. Consequently, we set the purpose of this paper as realizing a technique which accelerates reliable data persisting processing on DBMS. To efficiently avoid data loss at system crash, DBMS makes log records, writes them onto persistent storage, and then updates durable storage [4]. In this paper, the process to write log records onto persistent storage is denoted as "persisting process".

To accelerate persisting process, differential logging [5] and remote logging [6,7] are proposed. [5] is for main memory database system (MMDB). Since it prepares a log file for each page, log files are divided and persisting process is accelerated. [6] is also for MMDB. It uses two remote memories as persistent storage with 2 phase commit protocol. Since remote memory is faster than than local disk, it accelerates persisting

A Min Tjoa and J. Trujillo (Eds.): DaWaK 2006, LNCS 4081, pp. 524–533, 2006.
© Springer-Verlag Berlin Heidelberg 2006

process. [7] also uses remote memories, and the application domain is focused on sensor data. Inherently sensor data has continuous nature and thus it is considered that a few data loss can be recovered by interpolation. Based on the concept, the paper proposes imprecise remote logging.

Unfortunately, these techniques do not satisfy the above four conditions. Since differential logging [5] is for MMDB and thus the number of log files are limited. Therefore the technique is difficult to apply for our purpose. Since neighbor-WAL [6] does not describe how to deal with remote memory overflow, this technique is difficult to apply for our purpose. Since imprecise-WAL [7] is unreliable, this technique is difficult to apply for our purpose.

To achieve the purpose, this paper proposes **remote WAL with two level asynchronous checkpointing** technique, and realizes it on an object relational database system KRAFT [8].

This paper is organized as follows. Section 2 summaries related work. Section 3 formulates problems and describes basic architecture of object relational DBMS called KRAFT for the preparation of the proposition. Section 4 presents the two level asynchronous checkpointing technique. Section 5 describes experiments to evaluate the proposition. Section 6 discusses with this work. Finally Section 7 describes conclusions and future work.

2 Related Work

Differential Logging. P*TIME [5] adopts "differential logging" technique to accelerate WAL for MMDB. Differential logging adopts transient logging method and it reduces the amount of log records compared with ARIES [9] method since transient logging requires bit-level XOR difference of before image and after image while ARIES method requires both of them. Although the differential logging technique shows great performance, the maximum size of data which P*TIME can deal with is restricted to the size of physical memory. Since the size of sensor data increases monotonically as time goes, it is obvious that sensor data devours physical memories soon. Therefore differential logging technique is not proper for frequently sensor data insertion environment unfortunately.

Neighbor-WAL. Neighbor-WAL [6] uses two remote computer's memories (remote memories) as a persistent storage instead of a disk. The transfer of log records is executed on two phase commit protocol. Since the response time of remote memories is faster than a local disk, neighbor-WAL achieves faster performance compared with traditional disk based WAL. The weak point of neighbor-WAL is memory overflow. Since log records are stored on remote memories, the memories often overflow. Although this is the most important problem, no approach is described in [6].

Imprecise WAL. To accelerate sensor data insertion, imprecise WAL method is proposed in [7] as a modification of neighbor-WAL. This technique accelerates neighbor-WAL by reducing network traffic cost. On the imprecise WAL, DBMS does not receive any ACK of log record transfer from remote log servers. Although some log records might be lost, the authors of [7] consider that they can be interpolated since they are

Table 1. Comparison of Related Work

Study	Massive Data Management (C1)	Precise Persistence (C2)	Variable Length Tuple (C3)	Fast Data Insertion (C4)
Differential Logging [5]	Never	Good	Good	Good
Neighbor-WAL [6]	Never	Good	Good	Good
Imprecise WAL [7]	Good	Never	Never	Good
Conventional DBMS	Good	Good	Good	Bad
DSMS [2]	Never	Never	Good	Never
Ideal	Good	Good	Good	Good

gradually changing sensor data. To achieve performance, the imprecise WAL method loses certain persistence for each data. Since the target of this paper is providing persistence for all of sensor data to realize predictor applications, the technique is not proper for our purpose.

The summary of related work is shown in Table 1. We present the technique from the rest of this paper, and conducts the performance on our prototype DBMS called KRAFT.

3 Preparation

This Section prepares for our proposition in Section 4. The first subsection formulates problems to achieve our purpose, and the second subsection describes basic architecture the database system KRAFT in which we will built our proposition.

3.1 Problem Formulation

The purpose of this paper is realizing a technique which accelerates reliable data persisting processing on DBMS. In other words, a technique to solve (C4) should be proposed, and furthermore it should be realized on DBMS.

To evaluate the contribution of our proposition to (C4), we measure throughput of transactions, logging time of for a transaction, and blocking time to deal with remote memory overflow on KRAFT. Thus, we formulate the problems of this paper as follows.

$$\text{(P1)} : \text{Maximizing Throughput on DBMS}$$
$$\text{(P2)} : \text{Minimizing Logging Time on DBMS}$$

3.2 Basic Architecture of KRAFT

The proposition of this paper is based on an object relational database system KRAFT which we have developed. To clarify the novelty of the proposition, we describe basic architecture of KRAFT here. The implementation was done by programming language C on FreeBSD 5.3 Release. The number of lines is over 15000.

Overview. KRAFT is a database system that supports a variety of sensor data, and provides the following features: (1) freshness for sensor data without losing persistence, (2) abstract data type for sensor data, and (3) efficient periodic monitoring.

Data Model. KRAFT provides object relational data model as shown in Figure 1. Each tuple is constituted of RELATION part and SENSOR part and the format is dynamically set by CREATE TABLE command. The types which a tuple can have are INT (fixed length 4 byte), TEXT (variable length, however, the size should be lower than DBMS buffer pool size), and SENSOR which is constructed by the set of sensor data objects. A sensor object has four attributes: arrival time(16 byte), generate time (16 byte), sensor data (8 byte), and meta data (64 byte).

Software Architecture. Fig.2 shows basic software architecture of KRAFT. Since KRAFT conducts remote logging for persisting process, it is constituted of **DBMS Server** and **Log Server**. Although KRAFT has many modules, only **RecoveryManager**, **LogReceiver**, **CheckPointer**, **BufferManager** relate to the proposition of this paper.

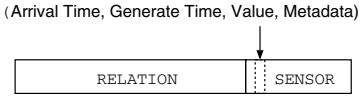

Fig. 1. Data Model of KRAFT

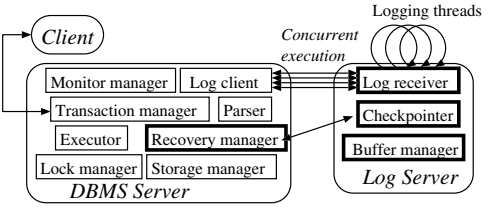

Fig. 2. KRAFT Architecture

Transaction Model and Operations. KRAFT supports INSERT, DELETE, and APPEND operations. INSERT and DELETE are used to manipulate tuples, while APPEND operation is used for inserting new sensor data objects to SENSOR area.

KRAFT recognizes **one operation as one transaction**. In other words, each operation is executed transactionally. Therefore, all of data on buffer pools are assured to be finished the persisting process.

Buffer Pool. Buffer pool is managed by storage manager on DBMS Server. If all of page are used, one page is selected as a victim. And the victim page is written to disk (durable storage), and it is initialized for next requirement. Buffer pool is constituted of N-th buffer pages. To optimize disk I/O, the size of each buffer page is set to the multiplies of PAGESIZE variable which is dependent on hardware.

Basic Remote WAL Protocol. KRAFT conducts remote WAL for the persisting process. The basic protocol of KRAFT's remote logging protocol is shown in the Fig.3. This figure omits error handling because of space limitation.

4 Approach to Problems

4.1 Two Level Asynchronous Checkpointing

To solve the problems which Section 3.1 described, blocking time should be decreased. To decrease the blocking time, this paper proposes the **two level asynchronous checkpointing** technique. The overview of the technique is shown in Fig.4.

When a Logger detects the end of k-th LogPage, it immediately switches to $k+1$-th LogPage. The switching time is the blocking time which have to be decreased. As Fig.4 shows, the technique is constituted of four threads, Redoer, Logger, LogWriter, and LogTransfer. And, Fig.4 shows asynchronous **buffer transfer** (log server's memory → log server's disk) and asynchronous **file transfer** (remote disk → DBMS storage area).

```
1: if (Send Log to LogSrv1 == timeout) {
2:     Recover using LogSrv2; }
3: if (Send Log to LogSrv2 == timeout) {
4:     Recover using LogSrv1; }
5: if (ACK from LogSrv1 == timeout) {
6:     Recover using LogSrv2; }
7: if (ACK from LogSrv2 == timeout) {
8:     Recover using LogSrv1; }
```

Fig. 3. Remote WAL Protocol

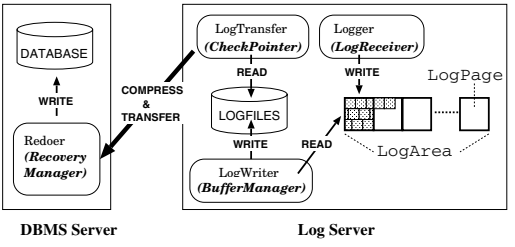

Fig. 4. Two Level Asynchronous Checkpointing

```
1: Set Current LogPage C = 0; (Initialization);
2: while (TRUE) {
3:     Recv log record L;
4:     Lock entire LogArea & Lock C;
5:     if (C is full) {
6:         Unlock C & Switch C &Lock C; }
7:     Unlock entire LogArea;
8:     Write L's size to C's header;
9:     Write L onto C;
10:    Unlock C; }
```

Fig. 5. Logger Algorithm

```
1: Dirty LogPage D = 0; (Initialization)
2: while (TRUE) {
3:     if (C > D) {
4:         Get T_i (Current Time);
5:         Make a log file of which name is T_i;
6:         Transfer D to T_i.
7:         Switch D; }
8:     sleep(TIME_LW); }
```

Fig. 6. LogWriter Algorithm

```
1: while (TRUE) {
2:     while (log file T_i exists) {
3:         Compress T_i & send it to DBMS Srv;
4:         Recv ACK from DBMS Srv;
5:         Delete T_i;}
6:     sleep(TIME_LT);}
```

Fig. 7. LogTransfer Algorithm

```
1: while (TRUE) {
2:     Receive the size of T_i;
3:     Allocate space to extract T_i;
4:     Recv T_i & Extract T_i;
5:     ptr = header of the first log record;
6:     while (ptr != NULL) {
7:         REDO using a log indicated by ptr;
8:         ptr = ptr->next; }
9:     Release allocated space;}
```

Fig. 8. Redoer Algorithm

4.2 Algorithm Descriptions

This subsection presents the description of algorithms to realize Redoer, Logger, LogWriter, and LogTransfer.

The algorithm of Logger is shown in Fig.5 Each log record has header area in which the size of log record is stored. Without the headers, it is impossible to reorganize log record from LogPage.

The algorithm of LogWriter is shown in Fig.6. LogWriter chases the current page C, but it never passes C. If LogWriter works slow, Logger soon finishes up all of log pages and waits for LogWriter, and it incurs blocking phenomenon. To avoid it, TIME_LW (Fig.6) should be slow enough.

Fig.7 shows LogTransfer algorithm. As long as log files T_i exists, LogTransfer compresses T_i and transfers it to Redoer. The compression reduces the amount of necessary network resource. In our experiment, the compression enhanced the performance of our system.

Fig.8 shows Redoer algorithm. Redoer receives compressed T_i, extracts it, conducts REDO processing by calculating the address of each log record. If the address is not written by storage manager, Redoer discards the log records because the access must make old the state of the page, which is never permitted.

5 Evaluation

5.1 Experimental Environment

Hardware and System Parameters

Hardware. The specification of a machine for DBMS server and clients is Pentium 4 (3 GHz) CPU, 4GB RAM, and FreeBSD 5.3-Release OS.And, the specification of machines for log servers are Pentium4 (2.4 GHz) CPU, 1GB RAM, and FreeBSD 5.3-Release OS. For network, 100 Mbps Ethernet interfaces and Gigabit Switching Hub FXG-08TE were used for the experiment.

System Parameters. Both TIME_LW (Fig.6) and TIME_LT (Fig.7) were adjusted as 1 micro second. The number of DBMS buffer pools was 32. FIFO was applied for page replacement algorithm. The number of log buffer pages on each log server was 128, and each size was 16 KB. To improve network response time, TCP_NODELAY option was set not to use Nagle algorithm on TCP/IP. For each experiment, all of clients are generated on a DBMS Server machine. The number of log servers is 2 for each experiment.

Comparison Methods. "DWAL (GC)" shows disk-based "Willing To Wait" policy group commit implementation on KRAFT.

"RWAL (Simple)" shows RWAL without group commit. In other words, all of log transfers are executed isolately. In this case, the number of Logger threads on each log server is the same as the number of DBMS clients which means the number of sensor data streams in this experiment. In this case, the size of log record is smaller than 1 KB.

"RWAL (GC)" shows RWAL-based "Willing To Wait" policy group commit implementation on KRAFT. The number of Logger threads on each log server is one since log transfers are integrated for group commit. The size of WAL buffer on DBMS Server is 16 KB.

"PostgreSQL" shows PostgreSQL-7.3.6 which implements "Willing To Wait" policy group commit.

Experiment Descriptions. We conduct three experiments. They are "throughput", "logging time" and "log insertion time on log server". As for "throughput", each client generages 1000 operations. Clients are concurrently executed. Total execution time is measured and then throughput is calculated. "Logging time" is the time for one WAL execution. This is measured at the internal of DBMS Server. "Log insertion time on a log server" is the time for LogPage modification on a log server. If memory overflow incurs blocking, this values would show high.

5.2 Results

Throughput. Fig.9 shows average of throughput. It shows that RWAL overwhelms DWAL. "RWAL (Simple)" shows 12 times better performance compared with "DWAL (GC)" in the maximum case (4 concurrency). However, "DWAL (GC)" shows worse performance than "PostgreSQL". Though we do not clarity the precise reason, the difference of buffer management algorithm (FIFO vs. clock) or "Willing To Wait" optimization might be related.

Fig.10 shows standard deviation of throughput. Though "RWAL (Simple)" shows unstable behavior, the reason is not clarified.

Logging Time. Fig.11 shows the average of logging time. "RWAL (Simple)" shows lower than 1000 micro seconds while concurrency is low, but the performance degrades in accordance with concurrency obtaining 4000 micro seconds at 500 concurrency. However, in all of concurrency, "RWAL (Simple)" shows better performance than "RWAL (GC)".

Fig.12 shows standard deviation of logging time. Though "RWAL (Simple)" shows unstable behavior, the reason is not clarified.

Log Insertion Time on a Log Server. Fig.13 shows average of one log record insertion time on a log server. In the worst case, "RWAL (Simple)" shows 7 micro seconds while "RWAL (GC)" shows 25 micro seconds. From the results, it is considered that blocking did not occur on log servers. the difference of 7 micro seconds and 25 micro seconds would be related to the size of insertion size. For "RWAL (Simple)", only one log record is written while sets of log records are written for "RWAL (GC)".

Fig.14 shows standard deviation time of one log record insertion time on each log server. All of values are smaller than 18 micro seconds, which are enoughly small.

Summary. As for performance "RWAL (Simple") policy showed the best performance in all of experiments. Since the policy overwhelms disk based WAL as for throughput, it satisfies **(P1) Maximing Throughput on DBMS**. In addition, since the policy also showed better performance than "RWAL (GC)", it most appropriately satisfies **(P2) Minimizing Logging Time on DBMS**. The reason why logging time was minimized, was because of non-blocking on log servers, which was clarified in the experiments with log insertion times.

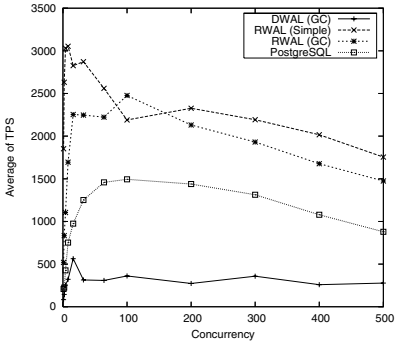

Fig. 9. Average of Throughput

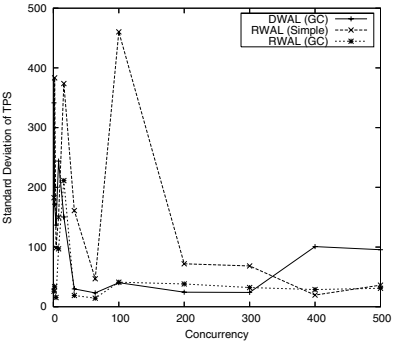

Fig. 10. Standard Deviation of Throughput

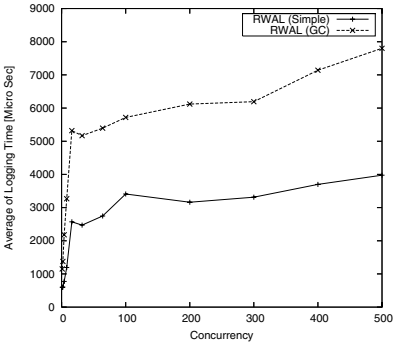

Fig. 11. Average of Logging Time

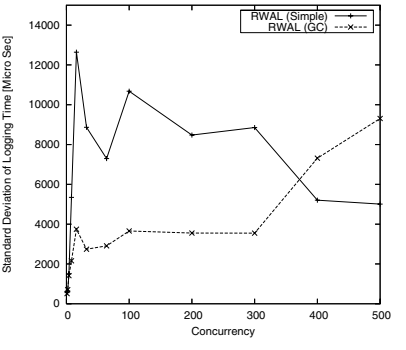

Fig. 12. Standard Deviation of Logging Time

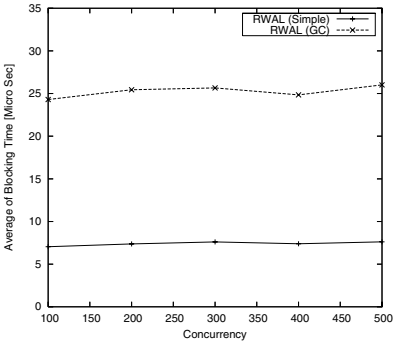

Fig. 13. Average of Log Insert Time

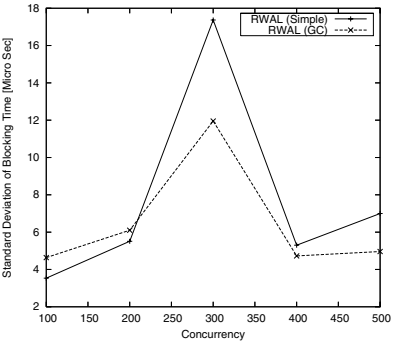

Fig. 14. Standard Deviation of Log Insert Time

6 Discussion

Persistence Strongness of Log Records by Remote WAL. If failure occurs on a host which log server runs, all of log records stored in the log server are lost. To cope with such a situation, our protocol shown in Fig.3 requires database server sending a log

record to two log servers. Therefore, our remote WAL protocol does not lose log records unless both of hosts failures at the same time, and we think the probability of the phenomenon would not be high. We think this philosophy is the same as the ClustRa [6] which is accepted for industry community. Therefore we consider remote WAL provide enough strongness of persistence for log records.

Preciousness of Sensor Data Stream. Currently sensor data streams are not considered to be enough to precious to persist in the field of sensor network community or data stream community. However we consider it will be precious in this decade, because (1) sensory/image communication is easier to understand compared with text based communication, (2) the price of disk is rapidly decreasing, and (3) real-time applications which use sensor data are emerging. Therefore we predict that data warehouses would store fine-grained sensor data streams in the near future.

Having N disks. Having N disks on the DBMS server, the performance of DWAL may be increased. However, the ratio would be low since (1) conventional group commit technology [10] is highly established and (2) the management of multiple group commit buffers requires cost.

SAN. By using SAN, the performance of DWAL would improve dramatically since batteries are equipped on disk cache device and thus no need to write some data on harddrive to persist the data. However, this paper focuses on low-end devices and thus expensive SAN is out of range.

7 Conclusions and Future Work

The purpose of this paper was to propose a technique which accelerates reliable data persisting processing on DBMS. To achieve the purpose, this paper proposed the **two level asynchronous checkpointing** technique, and implemented the proposition on an actual DBMS. And, this paper tackled the following three problems. (1) Maximizing throughput. (2) Minimizing logging time.

The result of experiments showed that remote WAL provided better performance than disk based WAL. As for throughput evaluation, the "RWAL (Simple)" policy showed about 12 times better performance compared with disk based WAL in the maximum case. As for logging time, the policy showed lower than 1000 micro seconds which is the period of motor data acquisition for conventionally used robots. Furthermore it also showed stable performance on log insertion time on log server. Therefore we consider the "RWAL (Simple)" policy most appropriately solves problems formulated in Section 3.1.

Therefore the proposition in this paper satisfies (C4) in Table 1. Furthermore, KRAFT already satisfies (C1), (C2), and (C3). Therefore new KRAFT reinforced by this paper satisfies all of four conditions in Table 1. Hence we conclude that our work achieved the purpose of this paper.

For further improvement, non volatile memories such as ram-disks should be used. Even if non volatile memories are used, the two level check pointing technique we proposed in this paper should be used because the available size of non volatile memories is still limited.

References

1. Madden, S. R., Franklin, M. J., Hellerstein, J. M. and Hong, W.: The Design of an Acquisitional Query Processor for Sensor Networks, *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 491–502 (2003).
2. Babcock, B., Babu, S., Datar, M., Motwani, R. and Widom, J.: Models and Issues in Data Stream Systems, *ACM Symposium on Principles of Database Systems* (2002).
3. Imai, M. and Narumi, M.: Generating common quality of sense by directed interaction, *Proceedings of the 12th IEEE International Workshop on Robot and Human Interactive Communication(RO-MAN 2003)*, pp. 199–204 (2003).
4. Gray, J. and Reuter, A.: *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers (1993).
5. Cha, S. K. and Song, C.: P*TIME: Highly Scalable OLTP DBMS for Managing Update-Intensive Stream Workload, *Proceedings of 30th International Conference on Very Large Data Bases*, pp. 1033–1044 (2004).
6. Hvasshovd, S.-O., Torbjørnsen, Ø., Bratsberg, S. E. and Holager, P.: The ClustRa Telecom Database: High Availability, High Throughput, and Real-Time Response, *Proceedings of the 21th International Conference on Very Large Data Bases*, pp. 469–477 (1995).
7. Kawashima, H., Toyama, M., Imai, M. and Anzai, Y.: Providing Persistence for Sensor Streams with Light Neighbor WAL, *Proceedings of Pacific Rim International Symposium on Dependable Computing(PRDC2002)*, pp. 257–264 (2002).
8. Kawashima, H., Imai, M. and Anzai, Y.: Improving Freshness of Sensor Data on KRAFT Sensor Database System, *International Workshop on Multimedia Information Systems*, pp. 1–8 (2004).
9. Mohan, C.: Repeating History Beyond ARIES, *Proceedings of 25th International Conference on Very Large Data Bases*, pp. 1–17 (1999).
10. Spiro, P. M., Joshi, A. M. and Rengarajan, T. K.: Designing an Optimized Transaction Commit Protocol, *Digital Technical Journal*, Vol. 3, No. 1, pp. 1–16 (1991).