Dhaka Univ. J. Sci. 58(1): 49-53 2010 (January)

A Modified Approach to Improve the Performance of Lazy Release Consistency (LRC) Model

Mosarrat Jahan, Shaily Kabir and Mohammad Asif Hossain Khan

Department of Computer Science & Engineering, Dhaka University, Dhaka-1000, Bangladesh

Received on 12.10.2008. Accepted for Publication on 01.06.2009

Abstract

This paper proposes a method for improving the performance of existing Lazy Release Consistency (LRC) model. In LRC model, before accessing a shared page a processor must acquire a lock associated with it as well as achieve all modifications made by other processors on the same shared page from the last lock releaser. All modifications are maintained by a data structure known as diff. For each access to a shared page a new diff is being created. As a result, the number of total diffs is increased in parallel with the access to shared page, causing a large amount of message transfer at the time of acquiring a lock. The proposed technique reduces the network traffic at a great extent by transferring only the critical section of shared page instead o f transferring a large number of diffs. It also ensures the transmission of message in reduced size for a particular page. Moreover, by eliminating the need of maintaining the redundant copies of a diff, the proposed method also reduces the vast memory requirement of LRC model. From the experimental results, it is found that the proposed modification significantly minimizes the message transfer as well as the memory requirement without affecting the functionalities of the LRC model.

Keywords: Distributed shared memory (DSM), Release Consistency (RC), LRC, Fast Lazy Release Consistency (FLRC), Consistency model, Critical Section (CS).

I. Introduction

Distributed shared memory (DSM) is a well-known inter-process communication technique for distributed system. It provides a vast amount of physically shared memory to the programmers and hides the explicit exchange of messages among various nodes [2]. In DSM, a shared page is replicated in the cache of different nodes to support simultaneous memory access operations. It provides the advantages of parallel read operations and also raises the need to maintain the consistency of the replica when a write operation takes place. The performance of DSM is limited by the additional communication traffic required in order to maintain memory consistency over physically separated nodes. To overcome this limitation, many memory consistency models have been proposed. A consistency model is a contract between the system and the application, which obligates the system to return an expected value to the application provided that the application accesses the memory following certain rules [3]. The first memory consistency model was Lamport's sequential consistency model [4] which had a very restricted memory access requirement and after that a number of consistency models were invented that relaxed the access constraint to a greater extent. It is observed that, the restrictive models guarantee better consistency but require large number of message transfer [3]. However, less restrictive models reduce the communication traffic and provide the necessary consistency required for the proper operation of the application programs [3].

This paper is organized into a number of sections: Section 2 represents the background of the paper, section 3 defines the basic terminologies, and section 4 and 5 provides the basic theme of LRC model. Section 6 contains the proposed modification and section 7 represents it's advantages. Section 8 and 9 presents the experimental result and discussion, respectively. Finally section 10 represents the conclusion.

II. Motivation of the Paper

A memory consistency model plays an important role in controlling the communication overhead of DSM. The first DSM implementation used sequential consistency (SC) model that was proposed in 1979 by Lamport [2]. In this model, all operations in a particular memory location are observed by all processes in the same order [2]. It requires that coherence operations had to be propagated immediately and processes had to wait for memory operations to complete before starting a new operation. After that, casual consistency model, pipelined random access model (PRAM) and processor consistency models were proposed to relax the strict memory access constraint of SC model [2]. All of them provided better consistency at the cost of large network traffic and reduced the concurrency. So, attempts were made by the researchers to relax the access constraints. In 1988, Dubios proposed weak consistency model that sends modification for a group of memory operations and uses a synchronization variable to determine the time to synchronize the memory [2]. Then RC model proposed that a process should access the "acquire" synchronization variable to collect all the changes made to the shared memory by other processors before entering a critical section (CS). The process must access "release" synchronization variable to send all the changes made by it in the CS to the other processors at the time of exiting the critical section. The execution on an RC memory produces the same results as that of SC memory for majority of the programs [5]. Based on RC model researchers developed the ERC model in Munin[5] and LRC in TreadMarks [6]. In ERC, a process sends all the modification to other processor when it exits from a critical section causing unnecessary modifications to propagate to a node that actually do not need them. LRC eliminates this problem by sending modification at the time of an acquire operation that involves achieving a lock ensuring mutual exclusion [6]. But it still suffers from the problem of transferring a large number of redundant modifications and also keeping a vast amount of storage for messages. For reducing huge network traffic and minimizing memory requirements, a new modification to LRC model is introduced in this paper.

III. Background of the Paper

In LRC model, a partial ordering known as happened-before-1 (hb1), $\xrightarrow{hb1}$ [1] is defined among the processors. This relationship is used to determine the access sequence of processors on a shared page. To define hb1 relationship the execution of each processor is divided into distinct time slots, known as interval. A new interval is started each time a processor performs an acquire or a release operation on a lock related to a shared page S. If all accesses in interval i_1 precede all accesses in interval i_2, then $i_1 \xrightarrow{hb1} i_2$. A processor P defines a vector timestamp $V^P(i)$ for each interval i where the number of entries in $V^p(i)$ is equal to the total number of processors in the system. . The entry for processor P in $V^P(i)$ is equal to i and the entries in $V^P(i)$ for other processors correspond to the most recent intervals where they performed modifications which are available on P. In FLRC, the vector timestamp is constructed using information stored in the page table of a shared page. In this model, if the P^{th} entry of $V^P(i)$ is i then it is interpreted as, “the node has received all write notices, for all intervals of processor P with the index less than or equal to i, during which intervals P has modified a shared page for which the lock has been requested” [7]. For Example, suppose, a system consists of three processors P_1, P_2 and P_3 and P_1 is going to access a shared page S. Now the vector timestamp constructed by P_1 is shown below:

P_1	P_2	P_3
3	4	5

Fig 1. Vector Timestamp Constructed by Processor P_1

This vector timestamp shows that S was accessed in interval 4 and 5 by P_2, P_3 respectively and P_1 is informed about these accesses. S was accessed by P_1 at interval 3 most recently. In the proposed modification, this definition of vector timestamp has been used.

In LRC model, to access a shared page S a processor P must make an acquire request to the lock manager associated with S. In the mean time, P determines the data required to update S. P calculates $V^P(i)$ using the information about S stored in its own page table and sends $V^p(i)$ to the most recent releaser, Q of the lock. On receiving $V^p(i)$, Q consults it’s page table to determine those intervals during which S has been modified but P is not notified to that. It sends the write notices for only those intervals together with the necessary diffs to P. Q also sends its currents vector timestamp $V^Q(j)$ to P. This is necessary to maintain the partial ordering between the intervals of different processors of the network.

Since a processor P must informed about all the diffs associated with a shared page S from the most recent releaser T before accessing S, T must maintain all previous diffs associated with S. It causes the redundant copies of same diff to be maintained on different nodes. Maintaining such a large number of diffs for a particular page consumes a large amount of memory space. Moreover, transmission of huge number of diffs as well as all the write notice records cause enormous network traffic. The total size of diffs transferred is usually much larger than the overall size of the page. These are the major limitations of LRC model that make it inefficient.

IV. Data Structures of LRC Model

The LRC model is implemented in the TreadMarks system, a *DSM system* used in the standard UNIX systems such as SunOS and Ultrix [6]. Figure-1 shows the data structures utilize in the TreadMarks system [6] used to calculate the timestamp V^P.

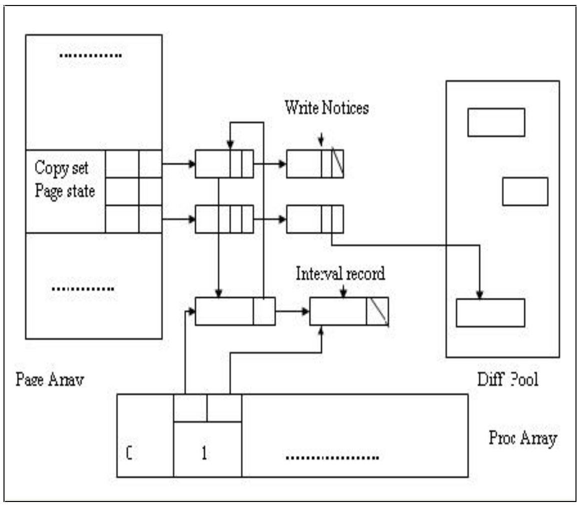

Fig. 2. Overview of the Data Structures Used in TreadMarks System

Page Array

Page array defines the page table for each node, having an entry for each shared page. For each page there is a list of write notice records for each processor, maintained by the head and tail pointers.

Proc Array

Proc array is indexed by all processors of the distributed system. Each entry of proc array contains pointers to the head and tail of a list of interval records.

Write Notice

A write notice is an indication that a page has been modified in a particular interval without having the actual modifications. If a diff corresponds to a particular write notice is currently in the diff pool of processor, then a pointer to this diff is present in that write notice record. Each write notice record contains a pointer to it’s interval record in the proc array.

Interval Record

Interval record contains a time stamp for an interval of a particular processor. Each of the interval records contains a pointer to a list of write notice records for that interval.

Diff Pool

Diff pool is a data structure used to store all the diff present at a node.

V. Proposed Modification to the Existing LRC Model

In LRC model, the most up-to-date page is constructed by combing the timely ordered diffs of different processors. It is a time-consuming and complicated process because it involves the transmission of large number of diffs. With time, a shared page is accessed so many times causing a huge number of diffs to exist for each shared page. It is common that the total size of the diffs can exceed the size of the page due to diff accumulation.

The proposed modification overcomes the restrictions of LRC model in a certain limit. In the modified approach, each processor only maintains the diffs for the intervals during which it modifies the shared page and when it releases a lock, it sends only the critical section (CS) of that modified shared page. The CS of a page contains the data sharing by a number of processors. Remaining section of a page contains processor's local data. A processor accesses CS between acquire and release pair and so diff contains only the modifications performed in CS on that period. Initially, the size of the diff may be small but with the progress of execution time, its size becomes as large as the size of CS. So, in LRC most of the diffs are of same size as CS and their cumulative size is much larger than the size of a CS of a shared page. The new modification reduces the total size of the data transferred over the network by sending the CS only instead of a huge number of diffs. In this method, a processor does not need to receive and/or maintain all the diffs of other processors that modify the same page. Each processor only maintains its own diffs. In case of a node failure, the diffs scattered on different nodes can be used to reconstruct a correct copy of a page. So, this scheme reduces the heavy network traffic as well as the large memory requirements of the existing LRC.

VI. Achievements of the Proposed Method

1. In proposed method, the process of getting most up-to-date page is simpler and shorter since it requests for the transmission of single message containing CS only.

2. In existing model, the same diff can be stored at multiple nodes requiring a huge memory storage which is very much costly. The proposed technique reduces the memory requirement by maintaining only the diffs that the processor generates.

3. Because of the existence of redundant copies of same diff on various nodes, a huge amount of traffic will be generated at the time of garbage collection which is executed periodically to reclaim the space used by the write notices, interval records and diffs. The proposed technique eliminates redundant copies of a diff and speed up the garbage collection procedure significantly by reducing the network congestion.

4. In DSM, thrashing occurs when the system spends a huge amount of time transferring shared page S from one node to another, without doing any fruitful work on S, degrading the system performance [2]. The proposed model is free from thrashing because every processor must obtain a lock before accessing S. A single message containing CS is transferred from a processor only when another processor acquires the lock on S.

VII. Experimental Results

Two simulation programs have written using C to evaluate the performance of existing LRC and the proposed LRC under the same load. The input and output parameters of the two programs are given below.

Input and Output Parameters

The input parameters:

- **MAX**: It is the number of nodes in the network. Increasing MAX increases the system load.
- **N**: It is the number of shared pages in the system.
- **L**: It is the number of critical sections in N shared pages.
- **TIMES**: It is the number of times the procedures are executed. In each iteration, lock manager selects a processor randomly.

The output parameters:

- Number of messages (Msg) transferred over the network to propagate the modifications.
- Number of diffs stored over the network

Case 1

N=20 and L=250. The output for different values of input parameters MAX and TIMES are given below.

Outpur:

Table 1. Comparisons between Existing LRC and the Proposed LRC when N=20 and L=250

Table. 1. Comparisons between Existing LRC and the Proposed LRC when N=20 and L=250

No. of Iterations (TIMES)	No. of Nodes (MAX)	No. of Stored Diffs (LRC)	No. of Stored Diffs (Proposal)	No. of Msg Transferred (LRC)	No. of Msg Transferred (Proposal)	Improvement in Msg transfer of proposed LRC (%)
5	5	49	16	33	14	57.58
10	5	139	36	103	39	62.14
10	10	590	80	510	87	82.94
15	10	1010	324	886	133	84.99
20	10	1503	568	1335	377	71.76
20	15	3526	962	3264	1076	67.03
20	20	7004	3157	6647	2371	64.33
20	30	10375	4838	8590	3130	63.56

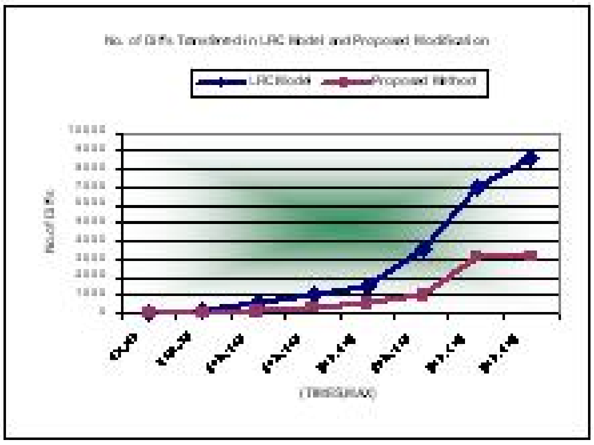

Chart 1. Number of diffs Transferred in Existing LRC and the Proposed LRC

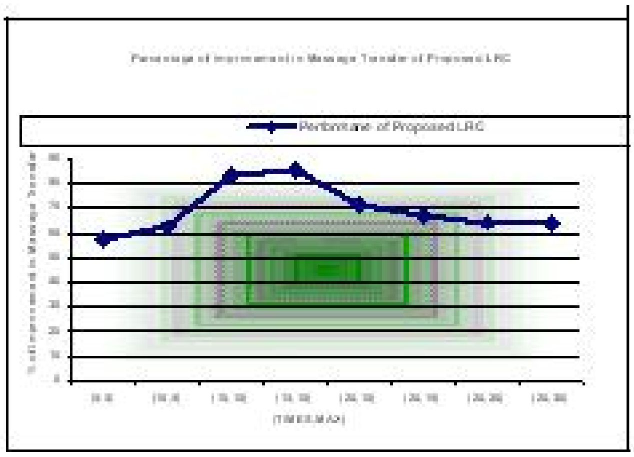

Chart 2. Percentage of Improvement in Message Transfer of Proposed LRC

Case 2

N=35 and L=500. The output for the input parameters MAX and TIMES are given below.

Output:

Table. 2. Comparisons between Existing LRC and the Proposed LRC when N=35 and L=500

No. of Iterations (TIMES)	No. of Nodes (MAX)	No. of Stored Diffs(LRC)	No. of Stored Diffs (Proposal)	Improvement in Storage of Diff s proposed LRC (%)	No. of Messages Transferred (LRC)	No. of Messages Transferred (Proposal)
5	5	49	16	67.34	38	12
10	5	139	36	74.10	122	40
10	10	590	81	86.27	452	86
15	10	1010	185	81.68	898	131
20	10	1603	370	76.91	1360	275
20	15	3526	1265	64.12	3129	725
20	20	7004	3159	54.89	5247	1172
20	30	10330	4852	53.03	9035	2659

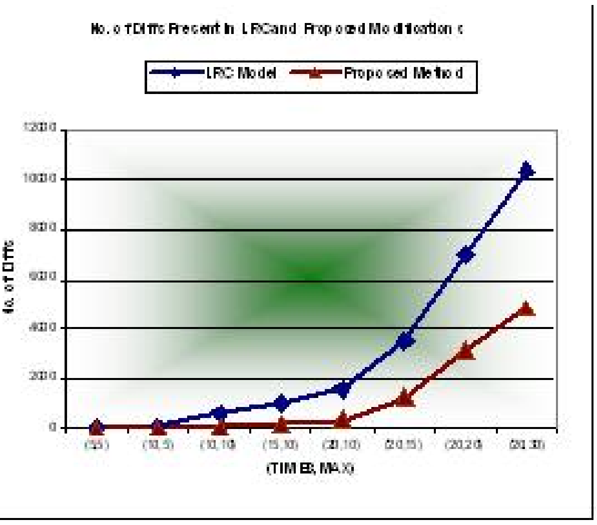

Chart 3. Number of diffs present in Existing LRC and the Proposed LRC

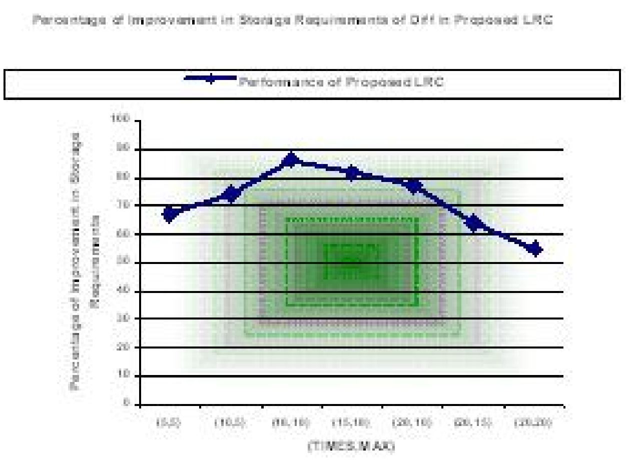

Chart 4. Percentage of Improvement in Storage Requirements of Diffs in Proposed LRC

VIII. Discussion

From the obtained result it can be stated that, for each combination of L, N, MAX and TIMES, the proposed method transfers far fewer messages as compared to the existing LRC. From table 1, it is observed that for N=20, L=250, MAX=10 and TIMES=15 the number of message transfer is reduced by (886-133) = 753, that is 84.99% improvement in reducing message traffic. From char 2, it can also be observed that the percentage of improvement in reducing the total no. of transferred messages in the proposed method is steadily rose to near about 85% at (15, 10) and after that, it is gradually declined. Since there is a relation between the no. of nodes and the no. of iterations with the total no. of messages transferred, so whenever the no. of nodes and the no. of iterations are increased, the no. of messages is also increased in parallel, declining the system performance by producing huge network traffic as well as raising vast memory requirements. From chart 4, it can also be perceived that the percentage of improvement in reducing storage requirements in the proposed method is increased to 86.27% at (10, 10) and after that, it is stepped down slowly. The memory requirement for the proposed modification is much less than that of LRC. So, the proposed LRC gives a significant reduction in network traffic as well as memory requirement than that of LRC.

IX. Conclusion

In this paper, it is examined the most widely used LRC model and proposed a modification in order to reduce its shortcomings for achieving the optimal system performance. The proposed method greatly condenses the number of message transferred over the network for keeping the distributed memory consistent. It also decreases the amount of storage requirement for diffs on different nodes in the system. The experimental results demonstrate that the proposed technique produces better result than the existing LRC model in terms of number of messages transferred over the network and the memory requirement of each node. So, the new modification proposed in this paper is useful in improving the performance of LRC model.

----

1. Keleher, P., Cox, A. L., and Zwaenepoel, W., 1992, "Lazy release consistency for software distributed shared memory", Proceedings of the 19th Annual International Symposium on Computer Architecture, 13-21.
2. Pradeep K. Sinha, 2001, "Distributed Operating Systems-Concept and Design", Prentice-Hall of India.
3. Yu, B., Huang, Z., Cranefield, S. and Purvis, M., 2004, "Homeless and home-based lazy release consistency protocols on distributed shared memory," Proceedings of the 27th Australian Computer Science Conference.
4. Lamport, L., 1979, "How to make a multiprocessor computer that correctly executes multiprocess programs", IEEE Transactions on Computers, 690-691.
5. Carter, J. B., Bennett, J. K., and Zwaenepoel, W. , 1991, "Implementation and performance of Munin," Proceedings of the 13th ACM Symposium on Operating Systems Principles, 152-164.
6. Keleher, P., Dwarkadas, S., Cox, A., and Zwaenepoel, W., 1994, "TreadMarks: Distributed shared memory on standard workstations and operating systems", Proceedings of the Winter Usenix Conference, 115-131.
7. Asif Hossain Khan and Mossadek H. Kamal, 2000, "FLRC: an improved cache-coherence protocol of software DSM", Dhake University of Journal of Science, 133-140.
8. Keleher, P., 1998, "On the Importance of Being Lazy", Technical Report UMIACS-TR-98-06, University of Maryland, College Park.
9. Sun, C., Huang, Z., Lei, W. J. and Sattar, A., 1997, "Heuristic Diff Acquiring in Lazy Release Consistency Model", Proceedings of Asian Computing Science Conference, Lecture Notes in Computer Science, 98-109.
10. John B. Carter, John K. Bennett and Willy Zweanepoel, 1995, "Techniques for Reducing Consistency-Related Communication in Distributed Shared Memory Systems", ACM Transaction of Computer Systems, 205-243.