LUC Model : A Timestamp Ordering based View Consistency Model for Distributed Shared Memory

Rubaiyat Islam Rafat, Kazi Sakib
Institute of Information Technology, University of Dhaka, Dhaka, Bangladesh
Email: rafat.mit.806@gmail.com, sakib@univdhaka.edu

***Abstract*— Excessive locking and cumulative updates in Distributed Shared Memory (DSM) not only reduces the parallelism for block access but also causes a serious degradation in response time for a dense network. This paper proposes a new consistency model in DSM named Last Update Consistency (LUC) model, where the model uses logical clock counter to keep the DSM consistent. The logical clock always increases never decreases. So the increasing order of the logical clock value is used to provide the request to the DSM. In this model, multiple nodes can perform READ operations over the same block at a time. For WRITE operation over the same block, only the last modification will exist and the earlier WRITE operations will be treated as obsolete WRITE and should be discarded. The experimental and analytical analysis showed that the proposed model effectively reduces the unnecessary network traffic and cumulative block updates that exist in the Sequential Consistency Model and Release Consistency Model.**

***Index Terms*— View Consistency Model, Timestamp Ordering Protocol, Distributed Shared Memory.**

I. INTRODUCTION

Increasing network density invokes the scalability issue for DSM. When large number of nodes exists in the distributed system, regular broadcast of block modification not only degrades the performance of DSM but also abuse the network traffic of the system. Cumulative block modification is another factor that degrades the performance of DSM and becomes a threat for the existing limited storage structure [6].

When a DSM model provides more concentration on sequential ordering, parallelism is reduced as a trade off. For maintaining consistency, one phase locking and two phase locking (SHARED and EXCLUSIVE) protocol [9] have been used by Release Consistency (RC) and Entry Consistency (EC) models. But both of these models are highly infected with cumulative modification of block updates and degrades throughput of the system. Although the ancient Sequential Consistency (SC) model is free from cumulative updates, it is not free from broadcasting block updates to the irrelevant nodes in the DSM [14].

This paper is based on "Logical Clock based Last Update Consistency model for Distributed Shared Memory" by Rubaiyat Islam Rafat and Kazi Sakib, which appeared in the 12th International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, December 2009.

If a DSM model uses different techniques for maintaining SC rather than locking mechanism, the higher response time issue in DSM might be minimized. Such a technique should be Timestamp Ordering (TO) Protocol. The proposed Last Update Consistency (LUC) model uses TO or logical clock counter to maintain SC, because TO is easy to implement and always ensures SC [3].

If the view level of data is considered, we can see that the user always get the last updated view of the data, no matter how and who has modified the data. Under this phenomenon, the cumulative updates of data are useless for the view layer of the user. If we want to ensure view consistency [12] and optimize our storage structure, we need to clean up the cumulative updates and keep the last modified data in the DSM. The proposed LUC model ensures view consistency [14] by storing only the last updated data [1].

The experimental result shows that the bytes transferred and response time for both WRITE and initial WRITE (UPLOAD) operation for different models considerably differs from each other. RC model shows better performance in bytes transferred for UPLOAD operation, while scores worst on response time for WRITE operation. SC model scores well in response time for WRITE operation but stuck on bytes transferred for UPLOAD. On the other hand, the proposed LUC model performs better for both the cases compared to existing consistency models. In response time comparison, LUC model minimizes 24% (average) over SC model and 32.5% (average) over RC model. In network traffic comparison, LUC model minimizes 22% (average) over SC model and 44% (average) over RC model. Moreover, the simulation result has supported LUC model which has minimized throughput to 24% and jitter to 40% than existing RC model.

The rest of the paper is organized as follows; section 2 elaborates the motivation of this research work. Section 3 describes the proposed model, its architecture and in general how it maintain consistency. Section 4 elaborates important and widely used consistency models for DSM and necessary background study for developing LUC model. The next section focuses on how this model operates in DSM and underlying algorithms of LUC model. Section 6 introduces some analytical information and comparative study between LUC model and other existing consistency models. The next section illustrates performance analysis

doi:10.4304/jcp.5.12.1828-1838

of LUC model and other existing consistency models in real world scenerio and simulation environment. The next section provides improvement discussion of LUC model for network traffic, response time, throughput and jitter factors and finally a summary of this paper and future scope of this model is discussed in the Conclusion section.

II. Motivation

The consistency model of a DSM system specifies the ordering constraints on concurrent memory accesses by multiple processors. So, it occupies a fundamental impact on DSM systems programming convenience and implementation efficiency [7]. The Sequential Consistency (SC) model has been recognized as the most natural and user friendly DSM consistency model. The SC model guarantees that the result of any execution is the same as if the operations of all processors were executed in some sequential order and the operations of each individual processor appear in this sequence in the order specified by its own program. This implementation is correct but it suffers from serious performance degradation [6].

To improve the performance of the strict SC model, a number of weaker SC models have been proposed [9], [13], [14]. However, none of them can achieve data selection without programmer annotation [2]. Usually consistency models should not impose any extra burden on programmers, such as annotation of lock-data association and scope-data association in Scope Consistency (ScC) [16].

View Consistency (VC) model is relatively new in the area of consistency models of DSM. Compared to other DSM consistency models, this model can achieve data selection without user annotation and reduce more false sharing effect by only updating data objects in the view of a processor [12]. Like some other weak Sequential Consistency models, the VC model can guarantee the same execution result as the SC model [4]. VC model allows an entity to only access later versions of data than what the entity had previously accessed. A data version is considered as older one if it is the same as or newer than the latest version accessed by any of the components of the entity. The optimistically replicated behavior of VC model provides highly available data even when communication between data replicas is unreliable or unavailable [5]. The advantages of the view-consistency model are two folds closely cooperating clients observe mutually consistent data, and distant clients do not pay the instantaneous cost of maintaining consistency (during accesses). Therefore view consistency enables useful collaboration in many large scale environments [2].

The high availability of replicas might create inconsistent accesses of data in VC model of DSM [4]. For ensuring synchronization, accesses of data can be regulated using locks before and after accessing a shared memory segment. Under lock-based concurrency control, an operation can obtain a lock only if another operation does not hold a conflicting lock on the same data item. When a lock is set, other operations that need to be set a conflicting lock are blocked until the lock is released, usually when the operation is completed. The more operations that are running concurrently, the greater the probability that operations will be blocked. These Excessive locking lead to reduce throughput and increase response times [6]. The resource costs involved in lock maintenance are considerable and these costs dramatically escalate as the number and complexity of concurrent operations increases. Uses of Timestamp Ordering (TO) protocol might be a good solution to avoid unnecessary locking [3]. TO does not attempt to maintain SC by mutual exclusion so that it should not cause deadlock. Moreover, TO ensures SC without degrading throughput [3].

In this paper, we propose a View-based Consistency model where Timestamp based ordering is used, namely Last Update Consitency (LUC) model which, utilizes VC model ensuring consistency by TO.

III. Background and Related Work

In this section, we briefly describe the necessary Thomas Write Rule for maintaining consistency in TO protocol. We also discuss other related work in the arena of DSM consistency model.

A. Thomas Write Rule for Timestamp Ordering

While implementing the Timestamp based protocol one major challenge was obsolete write operation. When a later node executes a write operation over the same block that was writing in previous by another node, the earlier nodes' write operation has been treated as an obsolete write because only the latest modification will exist in the system. To avoid such problem LUC model uses Thomas Write Rule provided by R.H. Thomas [11] for obsolete write operation. Thomas Write Rule states that if Timestamp on a transaction T is $TS(T)$ and Write Timestamp on an object O is $WTS(O)$ and such a situation occurs that, $TS(T) < WTS(O)$, the current write action has been made obsolete by the most recent write of O, which follows the current write according to Timestamp ordering. If we consider a non-conflict serializable transaction schedule S where T_1 and T_2 transactions can perform concurrently, Thomas Write Rule relies on the fact that T_1's write on object A is never seen by any transaction and postulates that the schedule in Figure 1 is equivalent to the schedule in Figure 2, where T_2 occurs strictly after T_1, and that hence the write of T_1 can be ignored. This happens because the view level of data only gets the last modification of the data and all previous modifications are ignored.

B. Related Work

In the following sub section important research solutions for improving efficiency of the existing DSM system has been discussed.

Cristiana Amza in her paper describes Trademarks [8] DSM system that implements Lazy Release Consistency (LRC) model [13]. This model does not require the update

$$\begin{bmatrix} T_1 & T_2 \\ R(A) & \\ & W(A) \\ & COMMIT \\ W(A) & \\ COMMIT & \end{bmatrix}$$

Figure 1. A non conflicting serializable transaction schedule

$$\begin{bmatrix} T_1 & T_2 \\ R(A) & \\ W(A) & \\ COMMIT & \\ & W(A) \\ & COMMIT \end{bmatrix}$$

Figure 2. Equivalent non conflicting serializable transaction schedule

propagation at release time. Although, the Lazy Release Consistency (LRC) model improves the Release Consistency (RC) model [15] by performing time selection and processor selection, the unnecessary update propagation is further postponed and stored until another processor has successfully executed an acquire phase.

Bershad, describes Midway [10], a DSM that introduces a new consistency Model - The Entry Consistency (EC) model [9] that tried to remove the propagation of useless updates in LRC by requiring the programmer to annotate association between ordinary data objects and synchronization data objects (e.g. locks). Although the useless updates are prohibited in the model, the cumulative updates still remain in this system.

L. Iftode et al. describe another new consistency model namely, The Scope Consistency (ScC) model [16]. This model is very similar to EC, except it can partially automate the association between ordinary data objects and synchronization data objects by introducing the concept of consistency scope. The major disadvantage of this technique is, scopes have to be explicitly annotated by the programmer for non-critical regions. If the annotation is not correct, ScC cannot guarantee Sequential Consistency for the program.

Z. Huang et al. introduce view based consistency model and how it should be implemented [12]. They defined view as a set of ordinary data objects that a processor has the right to access in shared memory if it has gained exclusive access to the data object or the data object is read-only. Views have been classified as Critical Region Views (CRV) and Non-critical Region Views (NRV) [4]. A processor's CRV is its view while it executes inside a critical region. A processor's NRV is its view while it executes inside a non-critical region. VC model is free from user annotation for data selection. But VC needs to maintain strict SC because of high availability of data [5].

Object based DSM [7] is another type of DSM where a document is considered as an object. The Parent Class inherits the critical sections of the Child class, so false sharing and network traffic is reduced in such a system. LUC model is implemented on page based DSM not based on object based DSM.

Different consistency models emphasizes on different factors. SC model emphasizes more on sequential consistency but highly infected with increasing network traffic due to broadcasting updates [13]. RC model emphasizes more on locking mechanism but highly infected with increasing network traffic and response time due to cumulative updates and excessive locking [9]. LRC model emphasizes more on propagation of network traffic but highly infected with increasing volume of storage, uses for cumulative updates [16]. EC model emphasizes more on removing unnecessary locking mechanism but highly infected with increasing volume of storage, uses for cumulative updates [3]. ScC model emphasizes more on scope annotation but highly infected to guarantee sequential consistency, if the scope annotation is not correct [12]. So, the users of DSM demand such a model that not only overcomes the above limitations of the existing models but also ensures high degree of parallelism. The proposed LUC model tries to overcome the limitations of the existing models. It ensures sequential consistency, reduces the network traffic and response time, free from cumulative updates and also ensures view consistency that increases the parallelism in DSM.

IV. Overview

Figure 3 gives an overview of our framework, which consists of a set of distributed nodes having own memory and processing units, communication medium and a software DSM (SDSM) Layer. SDSM systems provide programmers with a shared memory programming environment across distributed memory architectures. In contrast to the message passing programming environment, the SDSM can resolve data dependencies within the application without the programmer having explicitly specify communication [12].

For developing LUC Model, a page based DSM system has been chosen like IVY [17]. The sequential consistency is maintained by the incremental order of timestamp or logical clock counter that only increases and never decreases. LUC model uses strict Timestamp Ordering (TO) [3] to synchronize concurrent accesses and guarantee SC. To ensure sequential access, timestamps generated at all nodes must be unique and totally ordered since a timestamp assigned to a transaction initiated at one site is also used to access a replica at another site. A global timestamp at the LUC matrix is increased whenever there is a new access, and a DSM manager is used to update timestamp it receives in a message from the LUC matrix to generate its local timestamps. In this model, all DSM managers maintain a LUC matrix for tracking the request and response of different nodes in DSM (Table I). When a node wants to perform a READ or WRITE operation, the DSM manager will provide that node a logical counter value C_i. If the operation is a READ operation, the

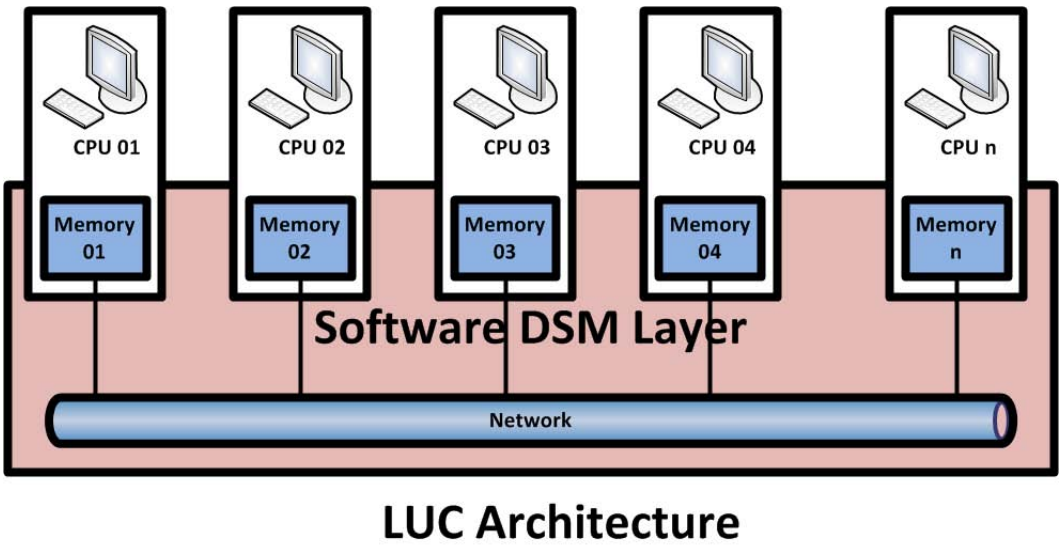

Figure 3. Overview of LUC Architecture

TABLE I.
LAST UPDATE CONSISTENCY MATRIX

Request	Counter	Node ID	Block ID	Opertion
WRQ	1	P1	A	Write
WRQ	2	P3	A	Write
UURQ	3	P1	A	Wait
UC	4	P3	A	Updated
RRQ	5	P2	A	Read

request type will be treated as Read Request (RRQ) (Table 1), if it is a WRITE operation, Write Request (WRQ) (Table I) is considered. If a node find itself as a obsolete block writer, it will discard its modification over that block and send an Urgent Update Request (UURQ) (Table 1) to the later block writer informing the node to send the updated block. After proper modification, the later node finds that already an UURQ is waiting for updated block, it will replicate and migrate the updated block to the requestor node and sends an Update Complete (UC) (Table I) message to the LUC matrix.

When any other node wants to perform READ operation over the same block but does not contain the replicated copy of the block, will search in the LUC matrix for the last updated block owner and replicate and migrate the block for READ. One of the main features of LUC Model is, only the last updated block will exist in the system and any obsolete write operations will be discarded according to Thomas write rule [11]. This mechanism should relief us from the cumulative updates problem. Moreover, the logical clock counter not only ensures sequential consistency but also reduces memory and time consumption from two phase locking mechanism [3].

V. CASE STUDY

In this section, we focuses on how our framework can be used. LUC is the main algorithm that ensures both the view consistency and timestamp ordering. First of all the generic VC algorithm has been discussed and then step by step how LUC algorithm works is described.

A. How View Consistency Works

In VC model the view moderator uses a generic VC algorithm to ensure consistency. Firstly, the replica and view timestamp should be read from the corresponding file by which a view entity is generated [Equation 1, 2].

$$viewTS,\ viewReplica \Leftarrow ReadViewEntry(fId) \quad (1)$$

$$viewEntry \Leftarrow viewTS,\ viewReplica \quad (2)$$

If the view entity is empty which indicates the node has no replica of the desired data, a new replica is created from First available location using Replicate and Migrate Block (RMB) approach [Equation 3].

$$newReplica \Leftarrow FastReplica(File,\ viewEntry) \quad (3)$$

If the view entity is non empty, a new replica is created from later replica which is actually the view of the next file operation [Equation 4].

$$newReplica \Leftarrow LaterReplica(File,\ viewEntry) \quad (4)$$

After completing the file operation the current timestamp is collected and being checked with the previous view timestamp. If the current timestamp is the later one which indicates the last modification of data, information is written into the view entity [Equation 5].

$$WriteViewEntry(fId,\ fileTS,\ newReplica) \quad (5)$$

Algorithm 1 View Consistency Algorithm

$fId, replica \Leftarrow File$
$viewTS, viewReplica \Leftarrow ReadViewEntry(fId)$
$viewEntry \Leftarrow viewTS, viewReplica$
if $viewEntry \neq NULL$ **then**
  $newReplica \Leftarrow LaterReplica(File, viewEntry)$
else
  $newReplica \Leftarrow FastReplica(File, viewEntry)$
end if
$data, fileTS \Leftarrow FileOperation(fId, newReplica)$
if $fileTS > viewTS$ **then**
  $WriteViewEntry(fId, fileTS, newReplica)$
end if
$return \quad data$

B. How LUC Algorithm Works

LUC algorithm utilizes both the VC algorithm and TO alogorithm to maintain SC. LUC algorithm performs two separate actions for READ and WRITE operations. In WRITE operation Thomas Write rule is used for obsolete WRITE.

Initially the LUC matrix remains empty [Equation 6].

$$LUC_{matrix} = \phi \tag{6}$$

and the counter initiates with 1 [Equation 7].

$$C(n) = 1 \tag{7}$$

The incoming operations on DSM are categorized into 3 categories namely, READ, WRITE and UPLOAD(Initial WRITE). If a READ operation occurs, READ algorithm is performed. If a WRITE operation occurs, WRITE algorithm is performed else the block is uploaded into DSM. Finally the counter is incremented with 1 [Equation 8].

$$C(n) = C(n) + 1 \tag{8}$$

Algorithm 2 LUC Algorithm

Require: $LUC_{matrix} = empty$
$C(n) \Leftarrow 1$
for all $Operations$ **do**
  if $READ$ **then**
    $READ$ Algorithm
  else if $WRITE$ **then**
    $WRITE$ Algorithm
  else
    $UploadBlock$ to DSM
  end if
  $C(n) \Leftarrow C(n) + 1$
end for

C. How READ Algorithm Works

First of all READ algorithm searches the LUC matrix for any request type containing UC or Update Complete [Equation 9].

$$RQ_{type} = UC \tag{9}$$

If no record exists, the requested node replicate and migrate the block from the original uploaded node and inserts a record containing request type RRQ or Read Request in the LUC matrix [Equation 10].

$$LUC_{matrix} \Leftarrow RQ_{type}(RRQ) \tag{10}$$

If any record exists, the requested node search LUC matrix for any request type containing WRQ or Write Request [Equation 11].

$$RQ_{type}(WRQ) \tag{11}$$

If any record exists, that record indicates currently writing over the block. So, the requested node inserts an UURQ type request or Urgent Update Request in the LUC matrix [Equation 12] and wait for updated block from the modifying node.

$$LUC_{matrix} \Leftarrow RQ_{type}(UURQ) \tag{12}$$

Now if the last modified node is the requesting node itself, it starts READ operation and inserts a entry containing request type RRQ or Read Request in the LUC matrix [Equation 13]. Otherwise, replicate and migrate the block from the last modified node and insert a record containing request type RRQ or Read Request in the LUC matrix.

$$LUC_{matrix} \Leftarrow RQ_{type}(RRQ) \tag{13}$$

Algorithm 3 READ Algorithm

repeat
  $BSearch(LUC_{matrix})$
until $RQ_{type} = UC$
if $Found$ **then**
  if $TS \neq (n-1)$ **then**
    repeat
      $BSearch(LUC_{matrix})$
    until $RQ_{type} = WRQ$
    if $Found$ **then**
      $LUC_{matrix} \Leftarrow RQ_{type}(UURQ)$
      $break$
    end if
  end if
  if $Node = own$ **then**
    $LUC_{matrix} \Leftarrow RQ_{type}(RRQ)$
    $StartRead$
  else $\{Node = other\}$
    RMB
    $LUC_{matrix} \Leftarrow RQ_{type}(RRQ)$
    $StartRead$
  end if
else
  RMB from Initial Uploaded Node
  $LUC_{matrix} \Leftarrow RQ_{type}(RRQ)$
  $StartRead$
end if

D. How WRITE Algorithm Works

First of all, WRITE algorithm searches the LUC matrix for any request type containing UC or Update Complete [Equation 14].

$$RQ_{type} = UC \tag{14}$$

If no record exists, the requested node replicates and migrates the block for writing from the original uploaded node and inserts a record containing request type WRQ or Write Request in the LUC matrix [Equation 15].

$$LUC_{matrix} \Leftarrow RQ_{type}(WRQ) \tag{15}$$

If any record exists, the requested node search LUC matrix for any request type containing WRQ or Write Request [Equation 16].

$$RQ_{type}(WRQ) \tag{16}$$

If any record exists, that record indicates currently writing over the block. So, the requested node inserts an UURQ type request or Urgent Update Request in the LUC matrix [Equation 17] and wait for updated block from the modifying node.

$$LUC_{matrix} \Leftarrow RQ_{type}(UURQ) \tag{17}$$

Now if the last modified node is the requesting node itself, it starts WRITE operation and inserts a entry containing request type WRQ or Write Request in the LUC matrix. Otherwise, replicate and migrate the block from the last modified node and insert a record containing request type WRQ or Write Request in the LUC matrix [Equation 18].

$$LUC_{matrix} \Leftarrow RQ_{type}(WRQ) \tag{18}$$

Now, WRITE algorithm search the LUC matrix again for any request type containing UURQ or Urgent Update Request [Equation 19].

$$RQ_{type}(UURQ) \tag{19}$$

If any record exists, the current node replicate and migrate the block to the requested node submitting request type UURQ or Urgent Update Request and inserts a entry for UC or Update Complete request type [Equation 20].

$$LUC_{matrix} \Leftarrow RQ_{type}(UC) \tag{20}$$

E. How LUC Model Handles Obsolete Write Operation

According to Thomas Write Rule, If a later node completes the write operation before the earlier node, over the same block but different view, an obsolete write operation performs. From [Figure 4], the sequence of operation of LUC model is discussed. DSM1 wants to perform a write operation of a block that belongs to DSM2. So replica is transfered to DSM1. In the meantime DSM3 wants to perform write operation over the same block so a replica is also been transfered to DSM3. timestamp of DSM1 for write operation is lower than DSM3, so it is expected to complete the write operation for DSM1 before DSM3 does. If DSM3 completes the operation before DSM1 then an obsolete write performs and DSM1 is notified about that. When DSM1 completes the operation DSM manager send a request to DSM3 to send the updated block to DSM1 because DSM1 performs an obsolete write. Then DSM3 sends the updated replica to DSM1 and both DSM1 and DSM3 hold the last updated view of data without any conflict and increasing concurrency. Now if DSM2 wants to perform a read operation over the same block, it needs to get the updated view of the same block that holds by DSM3. So again DSM3 can provide last updated replica to DSM2 and only the last updated view of data remains in the system. No cumulative updates generated by any nodes.

Algorithm 4 WRITE Algorithm

```
repeat
  BSearch(LUC_matrix)
until RQ_type = UC
if Found then
  if TS ≠ (n − 1) then
    repeat
      BSearch(LUC_matrix)
    until RQ_type = WRQ
    if Found then
      LUC_matrix ⇐ RQ_type(UURQ)
      break
    end if
  end if
  if Node = own then
    LUC_matrix ⇐ RQ_type(WRQ)
    StartWrite
  else {Node = other}
    RMB
    LUC_matrix ⇐ RQ_type(WRQ)
    StartWrite
  end if
  repeat
    BSearch(LUC_matrix)
  until RQ_type = UURQ
  if Found then
    C(n) ⇐ C(n) + 1
    RMB to the Requested Nodes
    LUC_matrix ⇐ RQ_type(UC)
  end if
else
  RMB from Initial Uploaded Node
  LUC_matrix ⇐ RQ_type(WRQ)
  StartWrite
end if
```

VI. Performance Analysis

It is difficult to provide a generic equation or a mathematical expression to measure the performance of any consistency models, but in general for measuring the network traffic or transferred bytes for any consistency model a general mathematical expression can be expressed as follows.

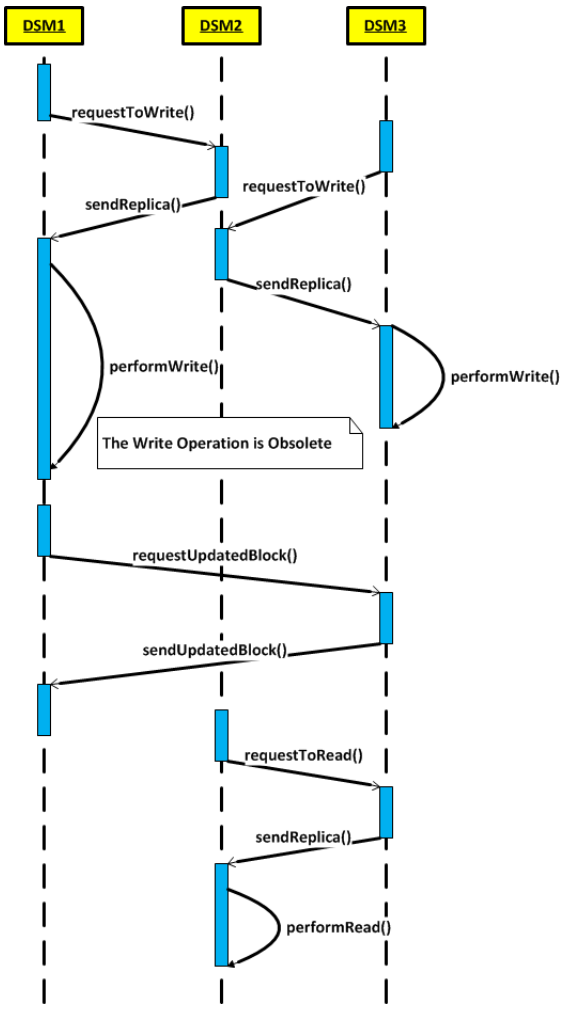

Figure 4. Sequence Diagram of LUC Model for READ and WRITE Operation

A. Sequential Consistency Model

For READ operation, SC model transmits an ACK message (M) to all existing nodes (n) in DSM and transmit respective block size (B′) for read in network traffic. So the equation for transferred bytes is

$$f_r(x) = \sum_{i=1}^{n-1} M_i + B' \tag{21}$$

For WRITE operation, SC model transmits both ACK message (M) and respective block size (B) for write to all existing nodes (n) in DSM. So the equation of transferred bytes is

$$f_w(x) = \sum_{i=1}^{n-1} M_i + \sum_{i=1}^{n} B_i \tag{22}$$

B. Release Consistency Model

For READ operation, RC model transmits an ACK message (M) to all relevant nodes (n′) in DSM and transmits respective block size (B′) for read in network traffic. In addition if any cumulative updates exist (p) for any node, that also transmits. So the equation of transferred bytes is

$$f_r(x) = \sum_{i=1}^{n'-1} M_i + B' + \sum_{i=0}^{p} B_i \tag{23}$$

For WRITE operation, RC model transmits ACK message (M) to all relevant nodes (n′) in DSM and transmits respective block size (B) for write in DSM. In addition if any cumulative updates exist for any node, that also transmits. So the equation of transferred bytes is

$$f_w(x) = \sum_{i=1}^{n'-1} M_i + \sum_{i=1}^{p} B_i \tag{24}$$

C. Last Update Consistency Model

For READ operation, LUC model transmits an ACK message (M) to all relevant nodes (n′) in DSM and transmit respective block size (B′) for read in network traffic. So the equation of transferred bytes is

$$f_r(x) = \sum_{i=1}^{n'-1} M_i + B' \tag{25}$$

For WRITE operation, LUC model transmits both ACK message (M) to all relevant nodes (n′) in DSM and transmits respective block size (B) for write in DSM. So the equation for transferred bytes is

$$f_w(x) = \sum_{i=1}^{n'-1} M_i + B \tag{26}$$

In the above equations, it is considered that the bytes transfered for write operation is considerably greater than the bytes transfered for read operation and bytes transfered for read operation is considerably greater than the bytes transfered to DSM manager for acknowledgement.

$$B \gg B' \gg M$$

and also the total number of nodes is considerably greater than the selective nodes of the network containing the same block.

$$n \gg n'$$

.

VII. Experimental Result Analysis

For comparing the performance between two existing consistency models and the LUC model, two major factors have been chosen. These are Network Traffic and Response Time for accomplishment of the operations [2]. Initially, Sequential Consistency (SC) model and Release Consistency (RC) model have been developed and then the proposed Last Update Consistency (LUC) model has been developed. The Network Traffic Analysis and Response Time Analysis depend on different factors, like in Response Time Analysis the nature of the system (Homogenous/ Heterogeneous) affects more. So, Homogenous system has been used where all nodes have the same components.

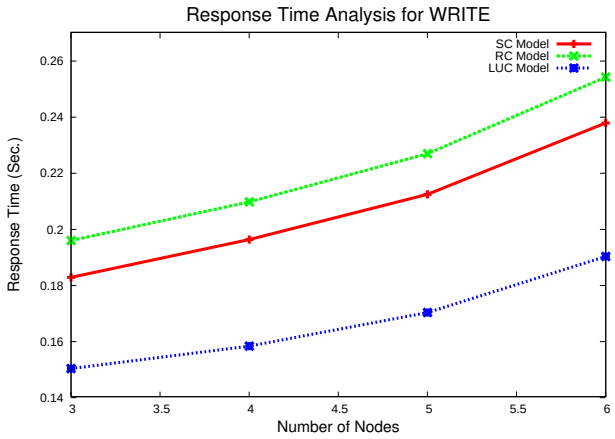

Figure 5. Response Time Performance Analysis for WRITE Operation

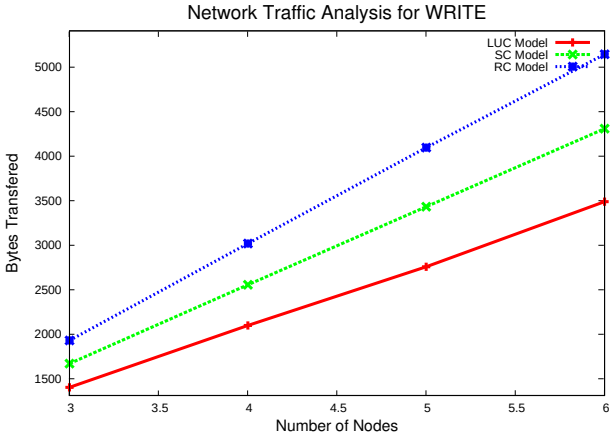

Figure 6. Network Traffic Performance Analysis for WRITE Operation

A. System Environment Description

A cluster of 6 identical 3.06 GHz Intel Xeon servers running Linux version 2.6.25 and connected through a gigabit switch have been used to evaluate our LUC approach. Each node contained individual DSM manager module for memory mapping. All nodes had identically similar processing power and memory. For developing the system a wired distributed network was chosen where all nodes were connected by bus topology. The implementation is done at the user level, without modification to the operating system kernel. Furthermore, this model does not rely on any particular compiler. Instead, the implementation relies on (user-level) memory management techniques to detect accesses and updates to shared data. In order to address the performance problems with earlier DSM systems, the LUC implementation focuses on reducing the amount of communication necessary to keep the distributed memories consistent.

B. Simulation Environment Description

For simulating the proposed model NS-2 simulator has been used. 2Mbps duplex links has been used between the source and destination nodes. The number of nodes was set to 50. Each node has used a DropTail queue, of which the maximum size was 10. UDP was chosen as agent and CBR for Application/Traffic. The duration for simulation was 5 sec.

C. Response Time Analysis

The graph [Figure 5] shows that the RC model shows the worst performance compare to other models, because of its cumulative update nature. RC model's performance degradation is proportional to the network density. The graph also shows that the LUC model performs better in this experiment because of its lower resource using strategy.

D. Network Traffic Analysis

The graph [Figure 6] shows that the RC model shows the worst performance compare to other models, because of its cumulative update nature. RC model's performance degradation is proportional to the network density. The graph also shows that the LUC model performs better in this experiment because of its lower resource using strategy.

E. Average Throughput Analysis

The throughput of the simulation increases because, file transfer rate increase as the size of the file increase. The higher throughput indicates higher data transmission of the network and maximum utilization. From the Figure 7, we find that the average throughput increases for both models linearly as the file size increases. Usually the throughput fluctuates over period of time. But the average throughput indicates the network traffic load over a system. Initially when the file size was small the total bytes transmitted by the system was low. When the block size increases the transmitted bytes also increase. In RC model the slope of the curve increased due to propagation of the unnecessary cumulative updates. On the other hand, in LUC model the gradient was lower than RC model due to minimization of the network traffic. Comparatively, LUC reduces the network traffic than RC model by not propagating unnecessary updates to DSM. But in the both cases throughput increase linearly with respect to block size due to network saturation in Ethernet.

F. Average Delay Time Analysis

Average delay time is a vital indicator for performance measuring between different DSMs. Although the average delay time for different models were expected to be different, the deviation in aveage delay time for different models in simulation environment was almost nil. The delay time of the simulation increases because of file size increase. The average delay time should provide a profound knowledge of time reduction of a system. From the Figure 8, we find that the average delay time increase for both models spontaneously as the file size increase from 2048 to 4096. But the average delay time was not increased significantly after that. The reason behind the

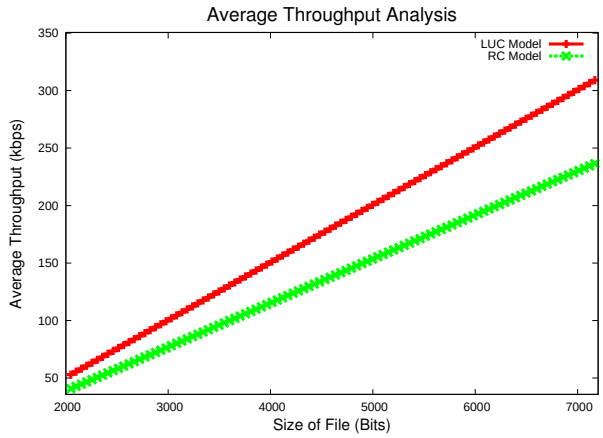

Figure 7. Average Throughput Analysis for LUC and RC Model in Simulator Environment

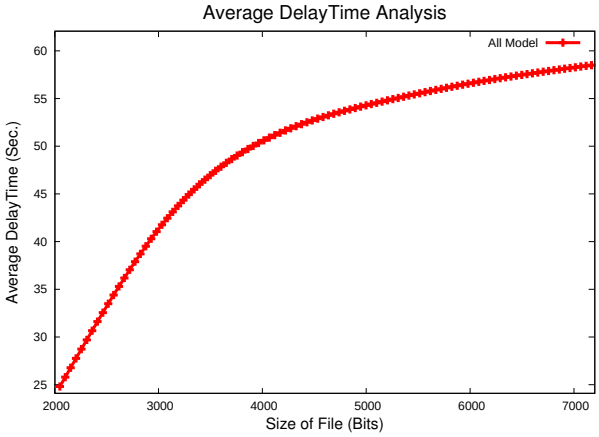

Figure 8. Average Delay Time Analysis for LUC and RC Model in Simulator Environment

curvature behavior of Average Delay Time is, initially when the file size increased the number of replicas of a specific block was limited. After sequence of operation the number of replicas existed to almost all the nodes. So, initially when the file size increased the Average Delay Time was also increased. After several operations when the replicas are available by most of the nodes only the diffs or the modified portion of the block was transmitted so the curve did not extend linearly although the file size increased gradually. Although the average delay time for all models are same, the average jitter curve should not be same as average delay time.

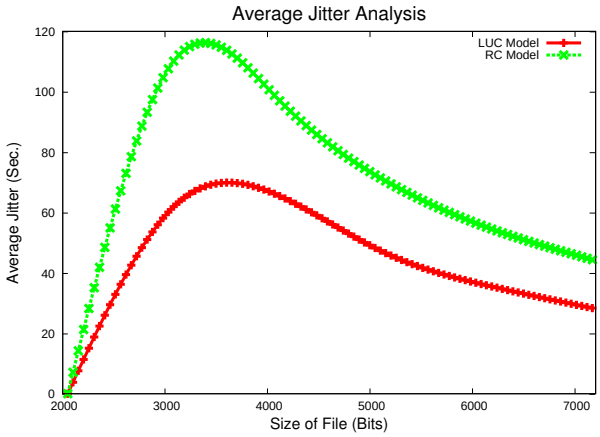

Figure 9. Average Jitter Analysis for LUC and RC Model in Simulator Environment

G. Average Jitter Analysis

Jitter is the time variation of a periodic signal in electronics and telecommunications. Jitter is used as a measure of the variability over time of the packet latency across a network. For measuring the performance of a DSM jitter can be a good indicator. If jitter is increased due to communication in DSM, the goodput of the system drops. So, in LUC model that uses timestamp ordering, might face challenges with respect to consistency. In existing models the increment of jitter not only hampers the concurrency but also increases the probability of thrashing. From the Figure 9, initially jitter increases spontaneously because average delay time increases gradually. But subsequently the jitter degrades when delay time failed to increase greatly. The major improvement of LUC model over existing RC model has been found in jitter analysis which was absent in average delay time analysis. In average delay time the improvement of LUC model was hidden due to excessive computation for ensuring consistency by providing more concurrency. But in jitter analysis LUC model scores better than RC model because of its underlying timestamp ordering, concurrent access and propagation of only the updated information by ensuring view consistency.

H. Improvement Analysis

From the Figure 10, the improvement of LUC model over SC model in Response Time shows an exponential curve having the increamental improvement of 21.73% , 24.11% , 24.82% and 25.11% with respect to scalability issue. The average improvement is 23.9525% .

From the Figure 11, the improvement of LUC model over RC model in Response Time shows an exponential curve having the increamental improvement of 30.47% , 32.59% , 33.35% and 33.68% with respect to scalability issue. The average improvement is 32.5264% .

From the Figure 12, the improvement of LUC model over SC model in Network Traffic shows a linear curve having the increamental improvement of 19.11% , 21.89% , 22.92% and 23.56% with respect to scalability issue. The average improvement is 21.8690% .

From the Figure 13, the improvement of LUC model over RC model in Network Traffic also shows a linear curve having the increamental improvement of 37.79% , 44.01% , 46.77% and 47.56% with respect to scalability issue. The average improvement is 44.0314% .

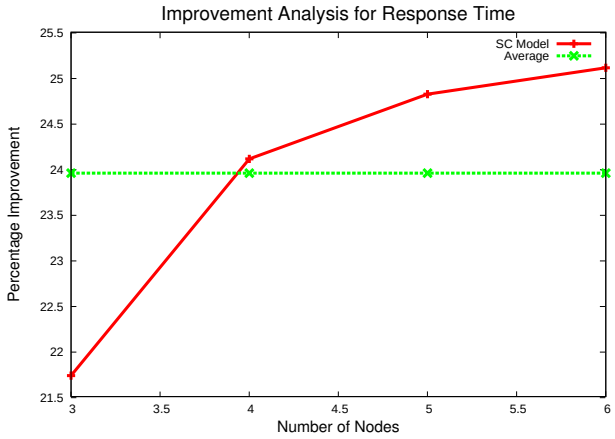

Figure 10. Improvement of LUC model compare to SC model in Response Time for WRITE Operation

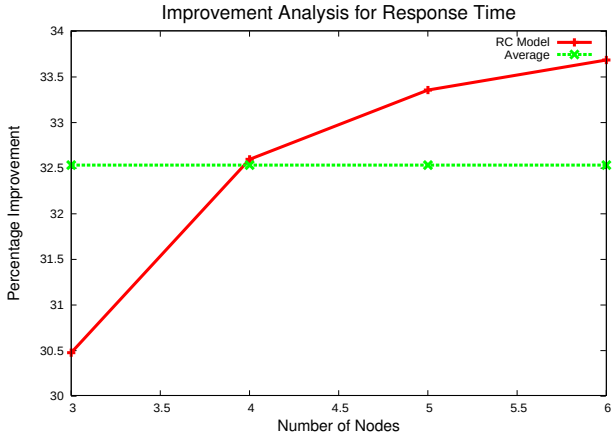

Figure 11. Improvement of LUC model compare to RC model in Response Time for WRITE Operation

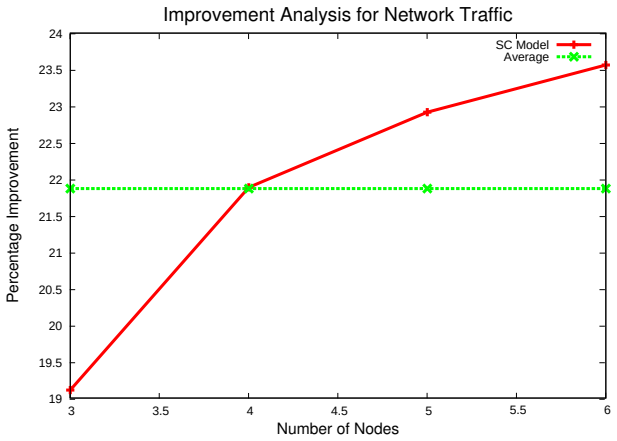

Figure 12. Improvement of LUC model compare to SC model in Network Traffic for WRITE Operation

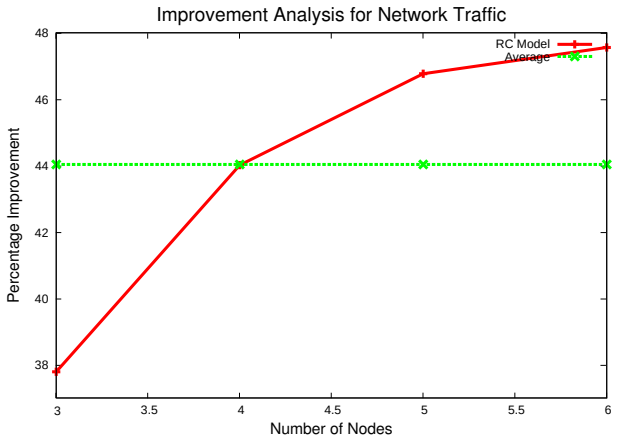

Figure 13. Improvement of LUC model compare to RC model in Network Traffic for WRITE Operation

VIII. Discussion

In this section the improvement of LUC model over RC and SC model in different factors is discussed.

The improvement of LUC model over SC model in Response Time shows an average improvement of 23.9525%. Secondly, the improvement of LUC model over RC model in Response Time shows an average improvement of 32.5264% . So, LUC model increases improvement of 8.5739% over RC model than SC model. RC model decreases its performance in response time for excessive locking mechanism(both READ and WRITE) and cumulative update propagation. SC model is free from cumulative update propagation so it scores better than RC model. Thirdly, the improvement of LUC model over SC model in Network Traffic shows an average improvement of 21.8690% . Finally, the improvement of LUC model over RC model in Network Traffic shows an average improvement of 44.0314% . So, LUC model increases improvement of 22.1624% over RC model than SC model. RC model decreases its performance in network traffic for cumulative update propagation. SC model is free from cumulative update propagation so it scores better than RC model. LUC model has neither cumulative updates nor excessive locking mechanisms and shows the best result in both response time analysis and network traffic analysis. In simulation environment, LUC model showed its strength by improving throughput to around 24% and jitter to around 40% over RC model. The reason behind the improvement of LUC model in simulation study was free from cumulative updates and excessive locking. Table II showed the performance improvements of LUC model in different aspects over different models in a compact form.

IX. Conclusion and Future Work

The major contribution of this paper over the concept DSM is to provide a new view consistency model named as LUC model that not only reduces the network traffic but also reduces the response time and excessive locks. LUC model successfully shows its improvements on different aspects, such as network traffic, throughput, delay time and jitter. LUC model decreases about 22% and 44% of Network traffic compared to those of SC and RCmodelrespectively. Moreover, LUC model takes 24% and 32.5%lesserin Response time than that of SC and RCmodelrespectively. Although the response time

TABLE II.
SUMMARY OF PERFORMANCE IMPROVEMENTS OF LUC MODEL

Model	Response Time	Network Traffic
SC	23.9525%	21.8690%
RC	32.5264%	44.0314%

and bytes transferred have been mitigated by this model, the programmers annotation and the code complexity of the LUC model increased notably. This is the trade off for LUC that the network traffic and response time has been optimized but the difficulty to implement and uses of local memory increases notably. However, the Network Traffic and Response time are two of the most important issues in Distributed Systems,where LUC model proves its superiority overothers [2]. The improvement of LUC model should open a new applicable entity in software engineering. LUC model can be adopted by software development firms for CVS(Concurrent Version System). The view consistency, high degree of concurrency and optimized resource utilization should make it usable as a substitute of existing CVSs. For using this model as CVS, each project in real world should be treated as an object or an entity of the DSM.

In this research,only the page based DSM is considered, but thereis a room to improve the LUCmodel by using Object based DSM [7]. We will use the modified critical segment transmission into the DSM like TradeMarks [8], which should improve the proposed model further. Moreover the behavior of the LUC model over grid and cloud computing environment will be observed where the scalability issue of DSM should be proven for LUC model. Avoidance of thrashing can be achieved by reducing false sharing. So in future, we should emphasize more on achieving stability by reducing false sharing and enhancing scalability of LUC model. The future of this research has been directed to those particular areas.

ACKNOWLEDGMENT

We thank Maruf Hasan for numerous discussions concerning this work and the reviewers for their detailed comments.

REFERENCES

[1] Z. Huang, M. Purvis, P. Werstein, "View-Oriented Update Protocol with Integrated Diff for View-based Consistency," *5th International Symposium on Cluster Computing and the Grid, IEEE Computer Society*, vol. 33, pp. 873-880, 2005

[2] Z. Huang, M. Purvis and P. Werstein, "Performance Evaluation of View-Oriented Parallel Programming," *34th International Conference on Parallel Processing, IEEE Computer Society*, vol. 33, pp. 251-258, 2005.

[3] Quazi Ehsanul Kabir and Mamun Hidenori Nakazato, "Timestamp Based Optimistic Concurrency Control," *IEEE Region 10 TENCON*, pp. 1-5, 2005.

[4] Z. Huang, C. Sun, M. Purvis and S. Cranefieldt, "View-based Consistency and False Sharing Effect in Distributed Shared Memory," *Operating Systems Review*, vol. 35, no. 2, pp. 51-60, 2001.

[5] Ashvin Goel, Calton Pu and Gerald J. Popeky, "View Consistency for Optimistic Replication," *17th IEEE Symposium on Reliable Distributed Systems*, vol. 33, pp. 36-42, 1998.

[6] John P Ryan and Brian A. Coghlan, "Distributed Shared Memory in a Grid Environment," *John Von Neumann Institute for Computing. NIC Series*, vol. 33, pp. 129-136, 2006.

[7] Deo Prakash Vidyarthi and Kirti Rani, "Distributed Shared Memory for Object Allocation in DCS," *International Journal of Information Technology and Management*, pp. 87-91, 2006.

[8] C. Amza, "TreadMarks: Shared memory computing on networks of workstations," *IEEE Computer*, vol. 29, no. 2, pp. 18-28, Feb. 1996.

[9] B.N. Bershad, "Shared Memory Parallel Programming with Entry Consistency for Distributed Memory Multiprocessors," *CMU Technical Report*, CMU-CS-91-170, Sep. 1991.

[10] B.N. Bershad, "The Midway Distributed Shared Memory System," *Proc. of IEEE COMPCON Conference*, pp. 528-537, 1993.

[11] R. H. Thomas, "A Majority Consensus Approach to Concurrency Control," *ACM Transactions on Database Systems*, vol. 4, no. 2, pp. 180-219, 1979.

[12] Z. Huang, C. Sun, M. Purvis, and S. Cranefield, "View-based Consistency and Its Implementation," *Proc. of the First IEEE/ACM Symposium on Cluster Computing and the Grid*, pp. 74-81, IEEE Computer Society, 2001.

[13] P. Keleher, "Lazy Release Consistency for Distributed Shared Memory," *Ph.D. Thesis*, Dept of Computer Science, Rice University, 1995.

[14] C. Sun, Z. Huang, W. J. Lei, and A. Sattar, "Towards Transparent Selective Sequential Consistency in Distributed Shared Memory Systems," *Proc. of the 18th IEEE International Conference on Distributed Computing Systems*,pp. 572-581, Amsterdam, May 1998.

[15] K. Gharachorloo, D. Lenoski, J. Laudon, "Memory consistency and event ordering in scalable shared memory multiprocessors," *Proc. of the 17th Annual International Symposium on Computer Architecture*, pp. 15-26, May 1990.

[16] L. Iftode, J.P. Singh and K. Li, "Scope Consistency: A Bridge between Release Consistency and Entry Consistency," *Proc. of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1996.

[17] K. Li, P. Hudak, "Memory Coherence in Shared Virtual Memory Systems," *ACM Trans. on Computer Systems*, vol. 7, pp. 321-359, Nov. 1989.

Rubaiyat Islam Rafat is currently a researcher at University of Dhaka, Bangladesh. He received his MS degree in information technology from Institute of Information Technology, University of Dhaka, Bangladesh, in 2009 and BS degree in computer science and engineering from Ahsaullah University of Science and Technology, Bangladesh, in 2005.

His research interest includes security and cryptography, distributed systems and data mining.

Kazi Sakib is currently a Post Doctorate fellow at University of Bradford, UK. He received his Ph.D. degree in computer science from the RMIT University, Melbourne, Australia in 2008, his MS and BS degree in computer science from the University of Dhaka in 2001 and 1999 respectively.

He is currently an Assistant Professor of Institute of Information Technology, University of Dhaka, Bangladesh. His current research interest includes sensor networks, the performance, availability, and security of distributed systems.