IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-4, Issue-12 E-ISSN: 2347-2693

Implementation and Consistency Issues in Distributed Shared Memory

Debendranath Das[1*], Ryan Saptarshi Ray[2] and Utpal Kumar Ray[3]

[1*]Department of Information Technology, Jadavpur University, Kolkata, India
[2]Department of Information Technology, Jadavpur University, Kolkata, India
[3]Department of Information Technology, Jadavpur University, Kolkata, India

Available online at: www.ijcseonline.org

Received: 24/Nov/2016 Revised: 28/Nov/2016 Accepted: 14/Dec/2016 Published: 31/Dec/2016

***Abstract*— Presently all programmers want to perform their tasks much faster than before. So, Parallel Processing comes into the picture to satisfy the increasing demands. Till a long time, parallel programs were only written either for multiprocessing environment or multi-computing environment. However, both of these parallel processing systems have some relative advantages and disadvantages. Distributed Shared Memory (DSM) system is a new and attractive area of research which combines the advantages of both shared-memory parallel processors (multiprocessors) and distributed systems (multi-computers). However, in DSM environment there are some critical issues like memory consistency that should be handled carefully. In this paper, an overview of DSM is given after a brief description of Distributed Computing Systems. Later various implementation issues and consistency models related to DSM are shown. Then an example of a simple program is given that can be implemented in DSM environment using Open SHMEM.**

Keywords- Parallel Programming; Multiprocessing; Multicomputing; Distributed Shared Memory (DSM); Consistency Models.

I. INTRODUCTION

The two basic or conventional methods for Information Sharing in MIMD (i.e. Multiple Instruction Multiple Data) architecture are –

1> Shared Memory Approach (Used in Multiprocessor Systems)

2> Message Passing Approach (Used in Multicomputer or Distributed Systems)

Both have some advantages and disadvantages. Distributed Shared Memory (DSM) is a relatively new concept which tries to combine the advantages of both multiprocessor systems and multicomputer systems. But still DSM has some disadvantages and there are lots of important issues which must be handled very carefully in order to successfully implement DSM. Consistency is one such important issue in DSM systems, which must be addressed very carefully by both the computer architects and programmers in order to write correct parallel programs maintaining memory consistency **[1]**.

Rest of the paper is organized as follows, Section II contains a brief overview of Distributed Shared Memory (DSM), Section III discusses the various implementation issues of DSM. Section IV contains the different consistency models of DSM. Section V shows how to write programs in DSM environment using OpenSHMEM and Section VI concludes this paper with future directions.

II. DISTRIBUTED SHARED MEMORY

Multiprocessor Systems (i.e. Shared Memory approach for communication) enable simple data sharing through a uniform mechanism of reading and writing shared structures in the common memory. This system has advantages of ease of programming and portability, although these systems are not scalable. Sometimes it suffers from longer latency for memory accessing, and often the design of memory system is a very complex task **[1,3]**.

Multicomputer Systems (i.e. Message Passing approach for communication) are scalable in nature. Hence, we can obtain high computation power from these systems. Communication between processes residing on different nodes involves a message-passing model that requires explicit use of send/receive primitives. Process migration imposes problems because of different address spaces. Therefore, compared to shared-memory systems, hardware problems are easier and software problems more complex in distributed-memory systems **[3,13]**.

DSM Overview:

In early days of distributed computing, it was implicitly assumed that programs on machines with no physically shared memory obviously ran in different address spaces. In 1986, Kai Li proposed a different scheme in his PhD dissertation entitled, "Shared Virtual Memory on loosely Coupled Microprocessors" **[6]**. It opened up a new area of research that is known as Distributed Shared Memory (DSM)

[3]. In DSM systems we try to incorporate the advantages of both Multiprocessor and Multicomputer Systems.
DSM provides a virtual address space shared among different processes in loosely coupled processors. That is, DSM is basically an abstraction that integrates the local memories of different machines in a network environment into a single logical entity shared by cooperating processes executing on multiple sites. The shared memory itself exists only virtually. Due to the virtual existence of the shared memory, DSM is sometimes also called as Distributed Shared Virtual Memory (DSVM) **[1,2]**.

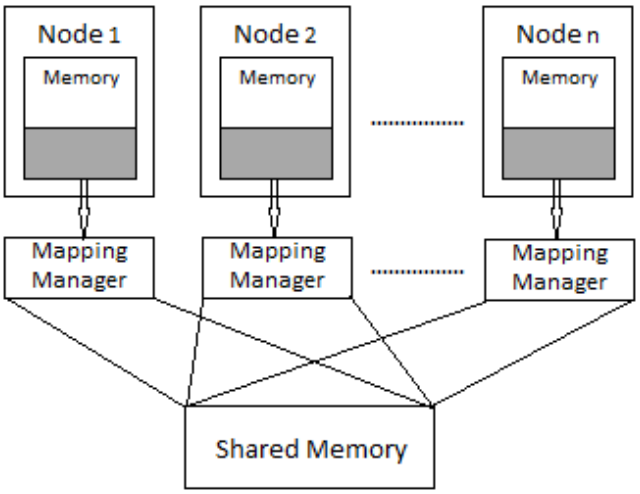

Figure 1. Distributed Shared Memory

Software DSM
Until recently the process communication in distributed systems was limited only to message passing paradigm. But some recent loosely coupled distributed memory systems have implemented a software layer on top of the message passing communication system to provide a shared memory abstraction to the programmers. The shared memory abstraction gives the system the illusion of physically shared memory and allows the programmers to use the shared memory paradigm. This is the concept of distributed shared memory where the distributed memory is not shared physically but they are shared logically by all the processors. The software layer which is used for providing the shared memory abstraction can be implemented either in an operating system kernel or in runtime library routines with proper system kernel support.

Hardware DSM
DSM can also be implemented by hardware. However, hardware implementation requires the addition of special network interfaces and cache coherence circuits to the system to make remote memory access look like local memory access. So, Hardware DSM is very expensive compared to Software implementation of DSM systems **[3]**.

Pros & Cons of DSM Systems

Pros: Because of the combined advantages of shared-memory and distributed systems, DSM approach is a viable solution for large-scale, high-performance systems with a reduced cost of parallel software development **[8]**.
Cons: Consistency can be an important issue in DSM as different processors access, cache and update a shared single memory space.

III. IMPLEMENTATION ISSUES OF DSM

Important issues involved in the design and implementation of DSM system are: **[1,8]**

- Virtual Memory and DSM
- Granularity
- Structure of Shared Memory Space
- Memory Coherence and Access Synchronization
- Data Location and Access
- Replacement Strategy
- Thrashing
- Heterogeneity

Virtual Memory and DSM
DSM is not true shared memory like tightly coupled multiprocessor system. Thus, remote memory accesses have to be reconciled with the memory manager at each node. Of course, if the basic machine architecture does not support virtual memory, then the solution could perhaps be simpler. However, if the basic architecture supports virtual memory then the DSM Management and Virtual Management(VM) have to be integrated. In particular, the local memory at each node may be considered as a large cache memory of the global distributed memory space that spans over the entire network. The DSM and VM management at each node would have to cooperate to ensure that the semantics implemented by the DSM manager and the VM manager are not compromised.

Granularity
One of the most important parameters in the design of DSM system is granularity. Granularity refers to the block size or page size of a DSM system. It indicates the unit of data sharing and the unit of data transfer across the network when a block or page fault occurs in the local memory of a site or node. Selecting proper block size is an important part in designing a good DSM system, otherwise, the system performance degrades. Due to the fact of locality of reference (especially spatial locality of reference), it is more likely that the process should refer next to a location, which is a neighbor of the current location. Hence if the page size or block size is greater, then relatively less number of times page faults will occur, which also, in turn, will save the network bandwidth and communication overhead. However, larger block size also creates some problems. For example, multiple processes at different nodes may refer to the same

page or block resulting in the frequent transfer of blocks from one node to another without actually performing the execution task, resulting in a condition called Thrashing. That is why selecting a suitable block size (i.e. granularity) is an important issue in designing a DSM system

Structure of Shared Memory Space
Structure refers to the layout of the shared data in memory. The structure of the shared memory space of DSM system is very much dependent on the applications that the DSM system is intended to support **[1]**.

Memory Coherence and Access Synchronization
If replication or copy of shared data items is allowed in a DSM system, then maintaining shared data consistency (also called memory coherence) is an important issue. This is because multiple sites or nodes may have a copy of the same shared data and if any of them modifies the data then it should be informed to all other nodes, otherwise other processes may use dirty data unknowingly. This problem is similar to multi cache scheme for shared memory multiprocessors. For solving this problem, we need to implement a good memory coherence protocol. In DSM system, concurrent access to shared data may be allowed. So, memory coherence protocol alone is not sufficient to maintain the consistency of the shared data, we also need some synchronization tools or primitives like semaphore, event flag, lock etc. for ensuring exclusive access to the shared data

Data Location and Access
To ensure data sharing in DSM system, all user processes must be able to locate and retrieve the relevant data. Therefore, a DSM system must implement some form of data block locating mechanism in order to service network data block faults.

Replacement Strategy
In DSM system, the main memory of each node can be considered as a bigger cache of the global shared address space. So, similar to caching mechanism, if a page fault (synonymous to cache fault) occurs at a specific site or node when the entire local memory is full, then it causes the replacing of existing page to accommodate the new arrival page. That is data block of local memory must be replaced by new data block. Therefore, a suitable cache replacement strategy (e.g. FIFO, LRU) is also necessary for the design of a DSM system.

Thrashing
In a DSM system, data block migrates from one node to another on demand. Therefore, if two nodes compete for write access to a single data item, the corresponding data block may be transferred back and forth at such a high rate that no real wok can get done. This is called thrashing **[1]**. DSM system should avoid this situation.

Heterogeneity
The DSM system must be designed to take care of heterogeneity so that it functions properly with machines having different architectures.

IV. CONSISTENCY MODELS

Among the various important issues of DSM systems discussed in the previous section, my focus for this paper is on Memory Coherence, which basically deals with the fact of maintaining the distributed shared memory consistent (i.e. Memory Coherence). In this section, various consistency models of DSM will be described.
Consistency requirements vary from application to application. A consistency model basically refers to the degree of consistency that has to be maintained for the shared memory data for the memory to work correctly for a certain set of applications. It is defined as the set of rules that applications must follow if they want the DSM system to provide the degree of consistency guaranteed by the consistency model **[1]**.

The memory consistency model of a shared-memory multiprocessor provides a formal specification of how the memory system will appear to the programmer, eliminating the gap between the behavior expected by the programmer and the actual behavior supported by the system.

A consistency model is essentially a contract between the software and the memory. It says that if the software agrees to obey certain rules, the memory promises to work correctly **[2]**. This model determines the order in which memory operations will appear to execute to the programmer. If we want stronger consistency model, then obviously, the degree of concurrency or parallelism gets down. So, the basic idea is to invent a consistency model that allows consistency requirements to be relaxed to a higher degree, with the relaxation done in such a way that a set of applications can function correctly.
Some major consistency models are described below **[1,2, 4].**

Strict Consistency Model
The most stringent consistency model is called strict consistency. It is defined by the following condition: "Any read to a memory location x returns the value stored by the most recent write operation to x." **[2,5]** Implementation of the strict consistency model requires the existence of an absolute global time so that memory read/write can be correctly ordered to make the meaning of "most recent" clear. However, in distributed system the existence of absolute global time is not possible and also synchronizing the local clocks of each node perfectly is impossible (due to

technical limitations). Hence implementation of strict consistency model in DSM system is practically impossible.

To understand this consistency, take an example: In Figure2, consider two processes P1 and P2. The operations done by each process are shown horizontally, with time increasing to the right. Straight lines separate the processes. The symbols W(x)a and R(y)b mean that a write to x with the value a and a read from y returning b have been done, respectively. P1 does a write to location x, storing the value 1. Later, P2 reads x and sees the 1 and this behavior is correct for a strictly consistent memory.

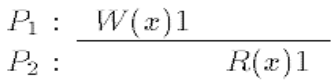

Figure 2. Strictly Consistency Memory

Sequential Consistency Model

This model was proposed by Lamport [1979] **[1]**. A shared memory system is said to be sequential consistent if all the processes see the same order of all memory access operations on the shared memory. For example, if three operations read(r1), write(w1) and read(r2) are performed in the same order on a memory address, any of the orderings of the three operations (i.e. permutation of r1, w1 and r2) is acceptable provided that all processes see the same ordering. If any of the processes see some other ordering of memory access operations than the others, then it is not a sequential consistent memory. Compared to strict consistency model, sequential consistency model is weaker, because strict consistency model only allows one ordering (r1, w1, r2) and nothing else. In a sequentially consistent system, all processors must agree on the order of observed effects.
The following is a legal execution history for SC.

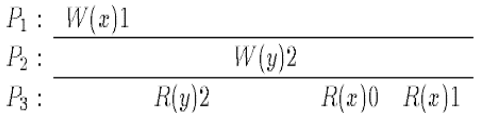

Figure 3. Sequential Consistency Memory

Note that R(y)2 by processor P3 reads a value that has not been written yet! Of course, this is not possible in any real physical system. However, it shows a surprising flexibility of the Sequential Consistency model.

Causal Consistency Model

This model was proposed by Hutto and Ahamad [1990] **[1]**. It relaxes the requirements of sequential consistency model for better concurrency. Unlike the sequential consistency model, in this model all processes see only those memory reference operations in the same order that are causally related and the memory reference operations that are not causally related may be seen in different order by different processes. A memory reference operation (read/write) is said to be causally related to another memory reference operation if the first one might have been influenced in any way by the second one. For example, if a process performs a read operation followed by a write operation on some memory address, then the write operation is causally related to the read operation because the value returned by the read operation must be dependent on the previous write operation. Obviously, in the implementation of shared memory system supporting this causal consistency model, there is a need to keep track of which memory reference operation is dependent on which other memory reference operations. This can be done by constructing and maintaining a dependency graph for the memory access operation.

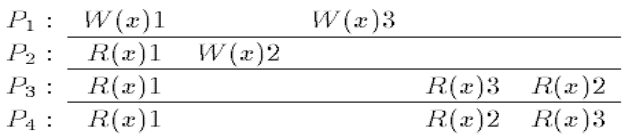

Figure 4. Causal Consistency Memory

This is a legal execution history under Causal Consistency but not under Sequential Consistency. Note that W(x)1 and W(x)2 are causally related as P2 observed the first write by P1. Furthermore, P3 and P4 observe the accesses W(x)2 and W(x)3 in different orders, which would not be legal under SC.

Pipelined Random-Access Memory Consistency Model

This PRAM consistency model, proposed by Lipton and Sandberg [1988]**[1],** provides weaker consistency semantics than the consistency models described so far. It only ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed as if all the write operations performed by a single process are in a pipeline. Write operations performed by different processes may be seen by different processes in different orders.
The following execution history is legal under PRAM Model but not under Causal Consistency Model:

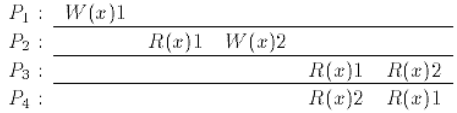

Figure 5. PRAM Consistency Memory

P3 and P4 observe the writes by P1 and P2 in different orders, although W(x)1 and W(x)2 are potentially causally related. Thus, this would not be a legal history for Causal Consistency.

Processor Consistency Model

This model was proposed by Goodman [1989]**[1].** It is similar to the previous one (i.e. PRAM consistency model), with one additional restriction imposed on it. That is memory coherence. Memory coherence means that for any memory location all processes agree on the same order of all write operations to that location. Processor consistency ensures that all write operations performed on the same memory location (no matter by which process they are performed) are seen by all processes in the same order. This requirement is in addition to the requirement imposed by the PRAM consistency model.

Weak Consistency Model

This model is proposed by Dubois [1988] **[1].** It is defined by stating the following properties: **[5]**

1. Accesses to synchronization variables are sequentially consistent.
2. No access to a synchronization variable is allowed to be performed until all previous writes have completed everywhere.
3. No data access (read or write) is allowed to be performed until all previous accesses to synchronization variables have been performed.

It is a weak consistency model that provides better performance at the cost of putting an extra burden on the programmers.

Release Consistency Model

This model provides accesses which are used to tell the memory system that a critical region is about to be entered. Release accesses say that a critical region has just been exited. These accesses can be implemented either as ordinary operations on special variables or as special operations. In either case, the programmer is responsible for putting explicit code in the program telling when to do them. For example, by calling library procedures such as acquire and release or procedures such as enter_critical_region and leave_critical_region. There are two flavors of release consistency that differ based on the program orders they maintain among special operations. The first flavor maintains sequential consistency among special operations (RCsc), while the second flavor maintains processor consistency among such operations (RCpc)**[5].** A variation of release consistency model is **Lazy Release Consistency Model** proposed by Kelehar [1992] and is more efficient than the conventional release consistency model.

V. PROGRAMMING IN DSM USING OPENSHMEM

Brief Overview of OpenSHMEM

OpenSHMEM is a standard for SHMEM library implementations which is used to write parallel programs in DSM environment. SHMEM is a communications library which we can use for Partitioned Global Address Space (PGAS) style programming **[9]**. Some important features of SHMEM are one-sided point-to-point and collective communication, a shared memory view, and atomic operations that operate on globally visible or "symmetric" variables in the program **[10].**

Simple Example Parallel Program written using OpenSHMEM:

This program simply prints the Hello World message along with its node number to the terminal.

```
#include <stdio.h>
#include "shmem.h"

#if !defined(OSHMEM_SPEC_VERSION) ||
OSHMEM_SPEC_VERSION < 10200
#error This application uses API 1.2 and up
#endif

int main(intargc, char* argv[])
{
        intproc, nproc;
        char name[SHMEM_MAX_NAME_LEN];
        int major, minor;

        shmem_init();
        nproc = shmem_n_pes();
        proc = shmem_my_pe();
        shmem_info_get_name(name);
        shmem_info_get_version(&major, &minor);

        printf("Hello, world, I am %d of %d: %s (version:
        %d.%d)\n",
        proc, nproc, name, major, minor);
        shmem_finalize();

        return 0;
}
```

A brief overview of the OpenSHMEM function calls used in the above program is given below: **[12]**

shmem_init() subroutine is used to initialize the calling PE and to reserve resources for communicating with other PEs. This subroutine performs the same function as the start_pes subroutine.

shmem_n_pes() returns the total number of PEs running in an application.

shmem_my_pe() returns the processing element (PE) number of the calling PE. It accepts no arguments. The result is an integer between 0 and npes - 1, where npes is the total number of PEs executing the current program.

shmem_info_get_name() returns the vendor defined character string of size defined by the constant SHMEM_MAX_NAME_LEN. The program calling this function prepares memory of size SHMEM_MAX_NAME_LEN.
shmem_info_get_version returns the major and minor version number of the software.

shmem_finalize() is a collective operation that ends the OpenSHMEM portion of a program previously initialized by shmem_init and releases resources used by the OpenSHMEM library. This collective operation requires all PEs to participate in the call.

VI. CONCLUSION

So, we can say that, DSM is a new paradigm for writing parallel programs that tries to take advantages of both multiprocessor systems and multicomputer systems or distributed systems, suppressing their disadvantages. However, for designing the DSM system we should take care of certain implementation issues like consistency that significantly affect the performance of the system. Software DSM is more cost effective and convenient than Hardware DSM. But it is still having some disadvantages. One such disadvantage is that programmers need to understand the various consistency models to write correct programs. OpenSHMEM programming paradigm is used to write parallel programs in DSM environment. OpenSHMEM is a library-based programming language and provides a high level of abstraction. It helps to avoid many parallel programming errors.

Further work can be done in the following fields:

a) Performance optimization of DSM systems
b) Investigation of new consistency models
c) Research can be done to explore how DSM systems can be used to solve some real life critical problem

REFERENCES

[1] Pradip K.Sinha, "Distributed Operating Systems: Concepts and Designs" published by IEEE Computer Society Press,2004.

[2] Andrew S. Tanenbaum, "Distributed Operating Systems" published by PEARSON Education, Fifth Impression,2008.

[3] Ryan Saptarshi Ray, Utpal Kumar Ray, Ashish Anand,Parama Bhaumik, "Distributed Shared Memory – A Survey and Implementation Using Openshmem" published in Int. Journal of Engineering Research and Applications, ISSN: 2248-9622, Vol. 6, Issue 2, (Part - 1) February 2016, pp.49-52.

[4] David Mosberger, "Memory Consistency Models", Paper submitted to Department of Computer Science, The University of Arizona, Tucson, AZ 85721.

[5] Radhika Gogia, Preeti Chhabra, Rupa Kumari**,** "CONSISTENCY MODELS IN DISTRIBUTED SHARED MEMORY SYSTEMS", Research Article published in IJCSMC, Vol. 3, Issue. 9, September 2014, pg.196 – 201.

[6] Kai Li, "Shared Virtual Memory on Loosely Coupled Microprocessors", PhD Thesis submitted to Yale University, September 1986.

[7] M. J. Flynn, "Computer Architecture: Pipelined and Parallel Processor Design" published by Jones and Barlett, Boston, 1995.

[8] Jelica Protic, Milo Tomasevic, Veljko Milutinovic, "A Survey of Distributed Shared Memory Systems" published in Proceedings of the 28th Annual Hawaii International Conference on System Sciences,1995.

[9] PGAS Forum,http://www.pgas.org/

[10] B. Chapman, T. Curtis, S. Pophale, S. Poole, J. Kuehn, C. Koelbel, L. Smith "Introducing OpenSHMEM, SHMEM for the PGASCommunity", published in Proceedings of Partitioned Global AddressSpace Conference, 2010.

[11] Message Passing Interface (MPI) standard, http://www-unix.mcs.anl.gov/mpi/

[12] http://openshmem.org/site/Documentation/Manpages/Browse

[13] Faizul Navi Khan, Kapil Govil, "Reliability Based Task Allocation Scheme to Enhance the Performance of Distributed Environment", Research Article published in IJCSE Vol. 2, Issue. 8, September 2014, pg.99.

Authors Profile

Debendranath Das received the degree of B.Tech in Computer Science and Engineering. from, West Bengal University of Technology, India in 2015. He is currently pursuing the degree of M.E. in Software Engineering from Jadavpur University, India from 2015 (presently attending the final year classes). His interests lie in the field of Distributed Systems,Parallel Computing, Data Structures and Algorithms, Swarm Intelligence.

Ryan Saptarshi Ray received the degree of B.E. in I.T. from School of Information Technology, West Bengal University of Technology, India in 2007. He received the degree of M.E. in Software Engineering from Jadavpur University, India in 2012. Currently he is Senior Research Fellow in the Department of Information Technology, Jadavpur University, India. He was employed as Programmer Analyst from 2007 to 2009 in Cognizant Technology Solutions. He has published 2 papers in International Conferences, 9 papers in International Journals and also a book titled "Software Transactional Memory: An Alternative to Locks" by LAP LAMBERT ACADEMIC PUBLISHING, GERMANY in 2012 co-authored with Utpal Kumar Ray..

Utpal Kumar Ray received the degree of B.E. in Electronics and Telecommunication Engineering in 1984 from Jadavpur University, India and the degree of M.Tech in Elecrical Engineering from Indian Institute of Technology, Kanpur in 1986. He was employed in different capacities in WIPRO INFOTECH LTD., Bangalore, India; WIPRO INFOTECH LTD., Bangalore, India, Client: TANDEM COMPUTERS, Austin, Texas, USA; HCL America, Sunnyvale, California, USA, Clent: HEWLETT PACKARD, Cupertino, California, USA; HCL Consulting, Gurgaon, India; HCL America, Sunnyvale, California, USA; RAVEL SOFTWARE INC., San Jose,

California, USA; STRATUS COMPUTERS, San Jose, California, USA; AUSPEX SYSTEMS, Santa Clara, California, USA and Sun Micro System, Menlo Park, California, USA for varying periods of duration from 1986 to 2002. From 2003 he is working as Assistant Professor in the Department of Information Technology, Jadavpur University, India. He has published 15 papers in different conferences and journals. He has also published a book titled "Software Transactional Memory: An Alternative to Locks" by LAP LAMBERT ACADEMIC PUBLISHING, GERMANY in 2012 co-authored with Ryan Saptarshi Ray..

