INTERNATIONAL JOURNAL OF ENHANCED RESEARCH IN SCIENCE TECHNOLOGY & ENGINEERING
VOL. 2 ISSUE 2, FEB.-2013 ISSN NO: 2319-7463

Defining Hybrid Distributed Shared Memory Consistency Models on Unified Framework

Pankaj Kumar[1], Krishna Kumar[2]
[1]Dept. of Computer Science, SRMCEM, Lucknow
[2]Research Scholar, CMJ University, Meghalaya
[1]pk79jan@rediffmail.com, [2]kkgkp@hotmail.com

Abstract: DSM (Distributed Shared Memory) system combines the advantages of shared memory parallel computer and distributed system. The value of DSM is depended upon the performance of MCM (Memory consistency Model). Lots of Consistency Model are defined by a wide variety of source including architecture system, application programmer etc. But basically it is combined in two types: uniform model and hybrid model.
In this paper we described only hybrid models for DSM. As hybrid model consider read, write as well as synchronization operation. So the paper reviews and discusses the different memory consistency models of distributed shared memory which are based on read, write as well as synchronization operation.

Keywords: DSM, MCM.

1. INTRODUCTION

A Distributed Shared Memory (DSM) system provides application programmers the illusion of shared memory on top of massage passing distributed system, which facilitates the task of parallel programming in distributed system. DSM is technique for making multicomputers easier to program by simulating a shared address space on them. In simple way we can say that DSM represents a successful hybrid of two parallel computer classes i.e. shared memory and distributed memory. It provides the shared memory abstraction in system with physically distributed memories and consequently combine the advantages of both approaches [3, 4, 6, 9].

A memory consistency model, or memory model, for a multiprocessor specifies how memory behaves with respect to read and write operations from multiple processors [3, 5]. With respect to the programmer's point of view, the model enables correct reasoning about the memory operations in a program. From the system designer's point of view, the model specifies acceptable memory behaviors for the system. As such, the memory consistency model influences many aspects of system design, including the design of programming languages, compilers, and the underlying hardware. In order to enhance performance, multiprocessors tend to implement sophisticated memory structures. These memories may replicate data through constructs such as caches and write buffers. Furthermore, the time required to access a data object may vary between processes and between objects. Any of these architectural features allow processes to have inconsistent views of memory, which, in turn, can result in unexpected program outcomes [5, 10, 15].

A memory consistency model is a set of guarantees describing constraints on the outcome of sequences of interleaved and simultaneous operations. Fewer guarantees allow more performance optimizations but yield machines that are very complex to understand and program. It is thus essential to provide multiprocessor programmers with a precise description of the memory model of the underlying machine. Several memory consistency models have been described in the literature. These descriptions arise from a wide variety of sources including architecture, system, and database designers, application programmers, and theoreticians. These descriptions use different types and degrees of formalism and hence are difficult to compare. Others are informal and sometimes ambiguous. There is no single unified formalization that describes the memory models addressed in the literature or provided by several existing machines.

2. MEMORY CONSISTENCY IN DSM

The consistency model of a DSM system specifies the ordering constraints on concurrent memory accesses by multiple processors, and hence has fundamental impact on DSM systems' programming convenience and implementation efficiency [18].

DSM allows processes to assume a globally shared virtual memory even though they execute on nodes that do not physically share memory. The DSM software provide the abstraction of a globally shared memory in which each processor can access any data item without the programmer having to worry about where the data is or how to obtain its value In contrast in the native programming model on networks of workstations message passing the programmer must decide when a processor needs to communicate with whom to communicate and what data to be send. For programs with complex data structures and sophisticated parallelization strategies this can become a daunting task [7, 19].

On a DSM system the programmer can focus on algorithmic development rather than on managing partitioned data sets and communicating values. The programming interfaces to DSM systems may differ in a variety of respects. The memory model refers to how updates to distributed shared memory are rejected to the processes in the system. The most intuitive model of distributed shared memory is that a read should always return the last value written unfortunately the notion of the last value written is not well defined in a distributed system [3, 18, 19].

The memory consistency model can be categorized into parts one which is based on read and write memory operation called as uniform model and the other which is based on synchronization operation also called hybrid model. The synchronization operations are mapped to corresponding operations provided by concurrency control [2, 6, 12].

3. PROPOSED STRUCTURAL HYBRID MODEL

A hybrid model consider read, write as well as synchronization operation. Processes want to restrict the order on which memory operation should be performed. Using this fact, hybrid memory consistency model guarantee that processors only have a consistent view of the memory at synchronization time. This allows a great overlapping of basic memory accesses that can potentially lead to considerable performance gain. Defining the Hybrid model is more complex then the uniform model because of memory operation, order of operation & relate to operation.

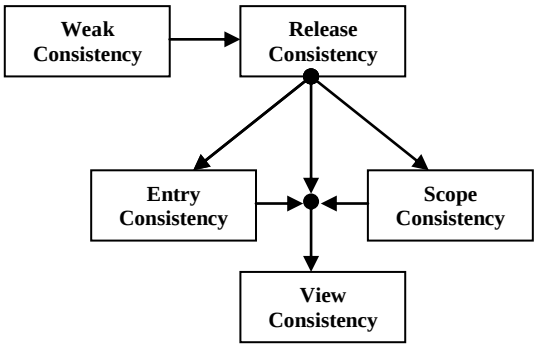

Figure 1. Structural Hybrid Model

We have taken following memory consistency model of distributed shared memory for our structural hybrid model:

a. Weak consistency model (WC)
b. Release consistency model (RC)
c. Eager release consistency (ERC)
d. Lazy release consistency (LRC)
e. Entry consistency (EC)
f. Scope consistency (ScC)
g. View based consistency (VC)

4. DESIGN AND DEVELOPMENT

The following design issues were examined for the said proposed model

4.1 Transparency

It is transparent and it has the single system image view. It works in the different transparency concepts, with Location Transparency: the user can not know where the most recent value is. For Migration Transparency: the user will not feel the existence of other users in the system using the same shared object if he has read only access, but he will feel it if he has write access to the shared object. Finally, Parallelism Transparency: can only be achieved in read operation.

4.2 Flexibility

In general it is flexible, since the micro kernel is used for interposes communication and helps memory management.

4.3 Reliability

It considered being reliable for the availability concept, since the fraction of time the system is used in asking for the counter value and receiving the answers is not too large if the system contains a small number of machines. For the Security concept, no other machine can access the shared object if it does not have authorization. For the Fault Tolerance concept, the system can work if one or more of the machines have crashed.

4.4 Performance

It shows performance in general with small number of machines in the system, but it may need a big bandwidth if the system has a large number of machines. If more than one machine asks for the shared object at the same time, the performance may become lower. Generally, in this model consistency is achieved over the performance.

VOL. 2 ISSUE 2, FEB.-2013 ISSN NO: 2319-7463

4.5 Scalability

It may be not very scalable for a large system.

5. MCM IN STRUCTURAL MODEL

In this section we define and describe the different consistency models used in the proposed structural model:

5.1 Weak Consistency

The Weak Consistency (WC) model proposed in 1986 was the first WSC model. Rather than requiring an update to be propagated to and executed at other processors immediately, WC requires that all previously-generated updates be propagated to and executed at all processors before a synchronization primitive is allowed to be executed. Thus propagation of updates can be postponed until a synchronization primitive is to be executed. WC can achieve time selection by propagating updates to other processors only at synchronization time, rather than at every update time. With time selection, updates can be accumulated and only the final results are propagated in batches at synchronization time [3, 5].

In this way, the number of messages in WC implementations can be greatly reduced compared to that in strict SC implementations. WC requires programmers to use explicit primitives, such as acquire and release, for synchronization. No data race on ordinary data objects is allowed in the program. If a program meets these requirements, WC can guarantee Sequential Consistency for it. A memory system is weakly consistent if it enforces the following restrictions [12, 17, 18]:

1. Accesses to synchronization variables are sequentially consistent and

2. No access to a synchronization variable is issued in a processor before all previous data accesses have been performed and

3. No access is issued by a processor before a previous access to a synchronization variable has been performed.

Weak consistency is depending upon execution history **(Hpi+w+sync)** of write (w) and synchronization (sync) operation of process P_i. So for operation $\mathbf{O_1}$ and $\mathbf{O_2}$, a weak consistency can be defined as

1. $\forall o1, o2$: **if type (O_1) =type (O_2) =sync** and $\exists$ $_{\mathbf{Hpi+w+sync}}$ between $\mathbf{O_1}$ and $\mathbf{O_2}$ where $o1 \xrightarrow{Hpi+w+sync} o2$ then $o1 \xrightarrow{WC} o2$.

2. $\forall o1, o2$: if $o1 \xrightarrow{cb} o2$ then $o1 \xrightarrow{WC} o2$

3. $\forall o1, o2$: **if processor (O_1) = processor (O_2) =P_i** and $o1 \xrightarrow{PO} o2$ $o1 \xrightarrow{WC} o2$

$\forall o1, o2, o3$: if $o1 \xrightarrow{WC} o2$ and $o2 \xrightarrow{WC} o3$ then $o1 \xrightarrow{WC} o3$

5.2 Release Consistency

It is one of the popular hybrid models. Release consistency is a relaxation of weak ordering where competing accesses are called special accesses [12]. It is defined by Gharachorloo et al. is a refinement of WC in the sense that competing accesses are divided into acquire, release, and non-synchronizing accesses. Competing accesses are also called special to distinguish them from noncompeting, ordinary accesses. Non-synchronizing accesses are competing accesses that do not serve a synchronization purpose [18]. An acquire access works like a synchronizing access under WC, except that the fence delays future accesses only. Similarly, a release works like a synchronizing access under WC, except that the fence delays until all previous accesses have been performed.

5.2.1 Eager Release Consistency

The Eager Release Consistency (ERC) model improves WC by removing the update propagation at acquire time. It requires that all previously-generated updates must be propagated to and executed at all processors before a release is allowed to be executed. So update propagation can be postponed until a release is to be executed. ERC takes time selection one step further than the WC model by distinguishing two different synchronization primitives: acquire and release, which are the entry and exit of a critical region respectively. ERC requires that updates be propagated to other processors only at release time [4, 6,].

In other words, ERC is more time-selective than the WC model by propagating updates only at the exit of a critical region, instead of at both the entry and exit of a critical region as in the WC model, thus further reducing the number of messages in the system. ERC has the same programmer interface as WC; though it removes update propagation at acquire time. It can guarantee Sequential Consistency for data-race-free programs that are properly labeled. A formal proof of this conclusion is provided in reference.

5.2.2 Lazy Release Consistency

The Lazy Release Consistency (LRC) model does not require the update propagation at release time. It postpones the update propagation until a processor calls an acquire at which time it knows which processor is the next one to need the updates. So LRC requires that before any access after an acquire is allowed to be executed, all previously-generated updates must be propagated to and executed at the processor executing the acquire [4, 5, 22].

The Lazy Release Consistency (LRC) model improves the ERC model by performing both time selection and processor selection. LRC can achieve time selection similar to ERC, except the update propagation is further postponed until another processor has successfully executed an acquire. LRC can achieve processor selection by postponing the update propagation until acquire time. At successful acquires, the DSM system is able to know precisely which processor is the next one to access the shared data objects, so updates can be propagated only to that particular processor (or no propagation at all if the next processor is the current processor). By sending updates only to the processor that has just entered a critical region; more messages can be reduced in the LRC model. LRC has the same programmer interface as WC and ERC. It can guarantee Sequential Consistency for data-race free programs that are properly labeled.

5.3 Entry Consistency

The entry consistency [EC] model is even weaker than RC. However, it imposes more restrictions on the programming model. EC is like RC except that every shared variable needs to be associated with a synchronization variable. A synchronizing variable is either a lock or a barrier. The association between a variable and its synchronization variable can change dynamically under program control. Note that this, like slow memory, is a location relative weakening of a consistency model. This has the effect that accesses to different critical sections can proceed concurrently, which would not be possible under RC [12, 18].

Another feature of EC is that it refines acquire accesses into exclusive and non-exclusive acquisitions. This, again, increases potential concurrency as nonexclusive acquisitions to the same synchronization variable can be granted concurrently. However, unlike RC, entry consistency is not prepared to handle chaotic (disordered) accesses.

The Entry Consistency (EC) model 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). When a processor acquires a synchronization data object, only the updates of the data objects that are associated with the synchronization data object are propagated to the processor. More precisely we say, EC requires that before any access after an acquire is allowed to be executed, all previously-generated updates of data objects that are associated with the corresponding synchronization data object, must be propagated to and executed at the processor executing the acquire. EC achieves the same time selection and processor selection as LRC, since updates are propagated only to the next processor calling an acquire [3, 4, 6]. EC achieves data selection by only propagating updates of data objects that are associated with a synchronization data object. The association, provided by the programmer, helps the EC model remove the propagation of some updates useless to a processor. With additional data selection, EC can be more efficient than LRC. In addition to requiring a program to be data-race free and properly labeled, EC requires the programmer to annotate the association between ordinary data objects and synchronization data objects in the program. If the association is correct, EC can guarantee Sequential Consistency for data-race-free programs; otherwise, Sequential Consistency is not guaranteed. The annotation of the association is normally regarded as an extra burden on the programmer.

5.4 Scope Consistency

The Scope Consistency (ScC) 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. ScC requires that before any access after an acquire is allowed to be executed, all previously-generated updates of data objects that belong to the corresponding scope, must be propagated to and executed by the processor executing the acquire. Like EC, ScC only propagates the updates of data objects that are in the current consistency scope. The difference is that a consistency scope can automatically establish the association between critical regions and data objects. For non-critical regions, however, scopes have to be explicitly annotated by the programmer. ScC achieves the same time selection, processor selection and data selection as EC. So it can offer the same performance advantages as EC if the scopes are well detected or annotated in the program [4, 22].

ScC improves the programmer interface of EC by requiring programmers to associate scopes with code sections, instead of data. For critical regions, scopes can be automatically associated; but the programmer has to annotate the scopes explicitly for non-critical regions. If the annotation is not correct, ScC can not guarantee Sequential Consistency for the program. ScC also requires the program to be data-race free and properly labeled.

5.5 View-Based Consistency

The View-based Consistency (VC) model is proposed to achieve data selection transparently without programmer annotation. A view is a set of ordinary data objects that a processor has the right to access in a data-race-free program. A processor's view changes when it moves from one region to another by calling acquire and release. VC requires that before a processor is allowed to enter a critical region or a non-critical region, all previously-generated updates of data objects that belong to the corresponding view, must be propagated to and executed at the processor. To selectively update data objects, VC uses view, while EC uses guarded shared data and ScC scope [4, 15].
However, the view in VC is different from in EC and the scope in ScC. Both and scope are static and fixed with a particular synchronization data object or a critical region. Even if some data objects are not accessed by a processor in a critical region, they are updated simply because they are associated with the lock or the critical region. The view in VC is dynamic and may be different from region to region. Even for the regions protected by the same lock, the views in them are different and depend on the data objects actually accessed by the processor in the regions. VC achieves the same time selection and processor selection as LRC. It can be more selective than EC and ScC in terms of data selection. VC has the same programmer interface as LRC, ERC, and WC. It can guarantee Sequential Consistency for data race- free programs that are properly labeled. To achieve data selection transparently, VC relies on techniques for automatic view detection.

6. DISCUSSION & CONCLUSION

Defining memory consistency models formally makes it easier to compare and relate them. The fig. shows the relationship among the different models of uniform model. Each rectangle shows the possible results that can be produced under the unified framework.

By the fig we can easily see that among the hybrid model, weak consistency is the strongest one. It imposes that all shared memory accesses previous to a synchronization access must be performed before the synchronization access performs and that no shared data access must be issued until all previous synchronization access are performed.

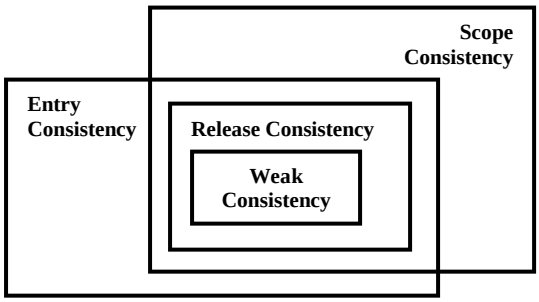

Figure 2 Relation b/w models of hybrid model

Release consistency is a model that divides the unique synchronization access of weak consistency into two distinct accesses: release & acquire. The first condition of weak consistency refers to only release access in release consistency while the second one refers only to acquire accesses. That's why weak consistency is strictly stronger than release consistency. While both entry and scope comparable. Consistency is strictly weaker than release consistency.

In this paper we have discussed the memory consistency model of distributed shared memory. As the memory consistency model can be implemented uniformly or in hybrid manner but we have discussed only hybrid memory model in detail. We discussed the basic characteristics of each models of hybrid model. We designed a structure of hybrid model which shows the relationship between the models of hybrid model. Another model can be developed by the wrapping the concept of uniform model to the concept of hybrid model.

7. ACKNOWLEDGEMENT

I would express my gratitude to **Prof. D. N. Kakkar,** Director, Sahara Arts & Management Academy for his sincere support and motivation. I can not forget the cooperation and guidance of my Ph.D. Guide Prof. Bal Gopal, Dean Computer Applications, Integral University, Lucknow. I would also thank to my entire colleague which gave me a lot of encouragement for writing this paper.

8. REFERENCE

[1]. L. Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs", IEEE Transaction Computers, vol. C-28, no. 9, pp. 690-691 September 1979.
[2]. S. Weber, P.A. Nixon and B Tangney," A flexible Frame work for Consistency Management in Object Oriented Distributed Shared Memory", Department of Computer Science, Trinity College, Ireland, October 13, 1998.
[3]. Paul Krzyzanowski "Distributed Shared Memory and Memory Consistency Models" Rutgers University – CS 417: Distributed Systems ©1998, 2001.
[4]. Z. Huang, C. Sun and M. Purvis "Selection-based Weak Sequential Consistency Models for Distributed Shared Memory" Departments of Computer & Information Science University of Otago, Dunedin, New Zealand , School of Computing & Information Technology Griffith University, Brisbane, Australia.
[5]. Lisa Higham, Jalal Kawash and Nathaly Verwaal, "Define and Comparing Memory Consistency Model" ©1997 ISCA, Proceeding of PDCS'97.
[6]. Abdelfatah Aref Yahya and Rana Mohamad Idrees Bader "Distributed Shared Memory Consistency Object-based Model", Journal of Computer Science 3 (1): 57-61, 2007 ISSN1549-3636© 2007 Science Publications.
[7]. J. Silcock "A Consistency Model for Distributed Shared Memory on RHODOS among Shared Memory Consistency Models" Deakin University, 1997.
[8]. John B.Carter, John K. Bennett and Willy Zwaenepoel "Techniques for Reducing Consistency-Related Communication in Distributed Shared Memory Systems"Rice University, TOCS95.
[9]. Changhun Lee "Distributed Shared Memory" Proceedings on the 15th CISL Winter Workshop Kushu, Japan ¢ February 2002.
[10]. Sarita V. Adve, Member, IEEE, Vijay S. Pai, Student Member, IEEE, and Parthasarathy Ranganathan, Student Member, IEEE "Recent Advances in Memory Consistency Models for Hardware Shared Memory Systems" proceedings Of The Ieee, Vol. 87, No. 3, March 1999.
[11]. Albert Meixner and Daniel J. Sorin "Dynamic Verification of Memory Consistency in Cache-Coherent Multithreaded Computer Architectures" Duke University, Department of Electrical and Computer Engineering, Technical Report #2006-1, April 18, 2006.
[12]. Alba Cristina Magalhães Alves de Melo "Defining Uniform and Hybrid Memory Consistency Models on a Unified Framework" Proceedings of the 32nd Hawaii International Conference on System Sciences -1999 IEEE.
[13]. Jason F. Cantin, Student Member, IEEE, Mikko H. Lipasti, Member, IEEE, and James E. Smith, Member, IEEE "The Complexity of Verifying Memory Coherence and Consistency" IEEE Transactions On Parallel And Distributed Systems, Vol. 16, No. 7, July 2005.

[14]. Robert C. Steinke and Gary J. Nutt "A Unified Theory of Shared Memory Consistency" Journal of the ACM, Vol. V, No. N, Month 20YY, Pages 1–47 2002.
[15]. Z. Huang, C. Sun and M. Purvis "A View-based Consistency Model based on Transparent Data Selection in Distributed Shared Memory" Technical Report OUCS-2004-03.
[16]. Ing. Thomes Seidmann," Distributed Shared memory in Modern Operating System" Ph.D. Thesis, Slovak University of Technology, January, 2004.
[17]. Jalal Y. Kawash" Limitations and Capabilities of Weak Memory Consistency Systems" Ph.D. Thesis Calgary, Alberta January, 2000.
[18]. D. Mosberger: "Memory consistency models", Operating Systems Review, 17(1):18-26, Jan. 1993.
[19]. Benny Wang-Leung Cheung, Cho-Li Wang and Francis Chimoon Lau, "Migrating-Home Protocol for Software Distributed Shared Memory", Journal of Information Science and Engineering, 2000.
[20]. Z. Huang, C. Sun and M. Purvis, "View-based Consistency for Distributed Shared Memory", Departments of Comp. & Infor. Science University of Otago Dunedin, New Zealand.
[21]. Liviu Iftode, Jawinder Pal Singh, Kai Li, " Scope Consistency: A bridge between Release Consistency and Entry Consistency" Proceeding of 8th annual symposium of parallel algorithms and architecture June 1996.
[22]. Pete Keleher, Alan L_ Cox and Willy Zwaenepoel "Lazy Release Consistency for software Distributed Shared Memory" Rice University, March 1992.