**CSE6306 – Advanced Operating Systems**

# CONSISTENCY MODELS IN DISTRIBUTED SHARED MEMORY SYSTEMS

By
**Savitha Ramesh**
**Department of Computer Science**
**University of Texas, Arlington**

## Abstract

The main motivation behind designing a Distributed shared memory System is to achieve good performance, which will in no way be affected when the size of the system grows. While looking into achieving such a goal in a network of systems, there are a few issues that arise, most important of those in DSM being Communication overheads and Reliability

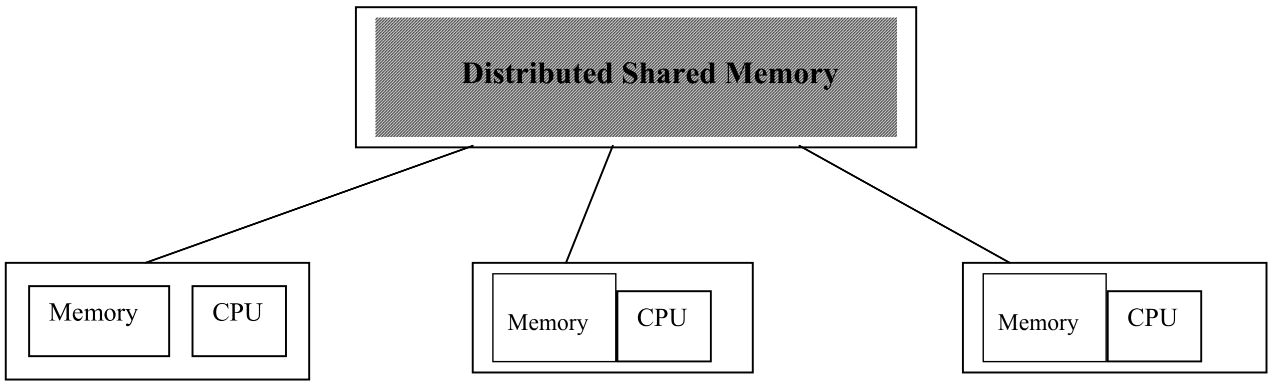
By reliability we mean, correctness of data, that is accessed and used in a Distributed shared memory System. Hence among many of the design issues in the Shared Memory Architecture of Distributed Systems, maintaining consistency proves to invite more attention. This paper discusses on the research done so far on the Consistency Aspect of DSM.

The introduction gives a good idea of the DSM and also the Consistency Issues. Then after giving a brief overview on the different types of consistency models in DSM, the paper will describe Release Consistency in detail. A comparison will be done on the various important Consistency models. We then discuss how protocols can be made adaptive depending on the application scenario. Finally in the conclusion we introduce the mixed consistency that talks about how few of these consistency models could be combined to achieve efficiency.

## 1 Introduction

A Distributed Shared Memory is a system in which the shared memory is actually physically distributed in different nodes in the network but appears to the user as a single address space. (Fig1)

**Figure 1**



## 1.1 Why Distributed Shared Memory?

In a traditional distributed system, data sharing has been done using a message-passing model. The client-server model and Remote Procedure calls are a few examples. In DSM, sharing is made possible by making the applications access data from the shared memory just as they access the virtual memory. The DSM software introduces a layer of software called the Memory mapping managers that maps the shared memory to the physical memory. Table 1 shows the reasons to go for DSM compared to message passing.

**Table 1- A Comparative Account**

| Issues | Message-passing | DSM |
|---|---|---|
| ***Programming Complexity*** | Programmers need to use explicit message-passing mechanisms to access shared data. | Easier to design and write parallel algorithms using DSM as compared to explicit message-passing mechanisms |
| ***Data Movement*** | Difficult to pass complex data structures between two processes | Complex Structures can be passed by reference. |
| ***Communication Overhead*** | Entire page or block containing the data referenced may have to be moved to the site of reference thus causing a communication overhead | Only the specific piece of data referenced needs to be moved, by taking advantage of the locality of reference. This reduces communication over head. |
| ***Cost*** | Not very cheap to build tightly-coupled multi-processor systems | Off-the-shelf hardware can be used to build DSM systems and does not require complex interfaces. Hence cheap |
| ***Memory Utilization*** | Not effectively utilized | Memory distributed in the network is effectively utilized to run programs that require large memory. |
| ***Scalability*** | The serialization point namely the common bus limits the number of processors in a tightly-coupled message passing system to a few tens of processors | A DSM system is highly scalable |

## 1.2 DESIGN ISSUES IN A DSM

The three main issues in designing a DSM system are

1. Keeping track of the location of remote data – by using a Memory mapping manager
2. Overcoming the communication overheads – by defining Granularity of data
3. Make shared data concurrently accessible and yet maintain consistency

The last issue can be termed as the consistency issue in the DSM systems. All these issues are inter-related in the sense that solution to one largely depends on the other. Hence while designing the models to achieve consistency we see that even the communication overheads and memory-mapping issues are taken care of.

### 1.2.1 CONSISTENCY ISSUES

The main idea of using a Distributed Shared Memory System is to achieve good performance by allowing concurrent access to shared data from many nodes in the network. But more than this basic requirement, the user should always be able to get the value he expects to get for a shared variable. Even among accesses made by the user there are different types such as ordinary accesses and special assesses that is access on shared data.

Moreover for performance reasons, shared data items are replicated on DSM systems for special accesses too. This might lead to potentially inconsistent data at different nodes when concurrent write access occurs on the same shared variable. In such a case, coherence is inevitable. Hence arises the issue of consistency.

There are two main options, to maintain consistency across a sequence of writes on shared data,

- ***Write-Invalidate*** – Implemented as *Multiple-reader/Multiple-writer Sharing*
- ***Write-Update*** – Implemented as *Multiple-reader/ Single-writer Sharing*

In such an attempt to maintain consistent data, there are also other issues that arise such as ***false sharing*** where two or more processes share parts of a page, but only one infact accesses each part. This leads to unnecessary invalidations in write-invalidate protocols and over-write of correct data by older versions in case of write-update protocols

Another consequent problem is ***thrashing*** especially in the case of write-invalidate where more time is spent on invalidating shared data and transferring them across the network than doing any actual work.

The consistency semantics vary from Sequential Consistency that is strict to a relaxed coherence semantics in Release Consistency. The following topics describe how different types of coherence protocol attempt to provide a solution to the identified problems in DSM.

## 2 Brief Overview on the Different Coherence Semantics in DSM

### Sequential Consistency

Following a serializable execution in concurrency control theory we say that an execution on a DSM is serializable if its result is the same as if the operations of all the processors had been executed in some sequential order, which satisfies processor, orders and is legal. A DSM system is sequentially consistent if all the executions on the DSM system are serializable. ***Ex: Ivy***

### Processor Consistency

Writes issued by each individual node are never seen out of order but the order of writes from two different nodes can be observed differently. [5]

### Weak Consistency

The programmer enforces consistency using synchronization operators guaranteed to be sequentially consistent. [5] ***Ex: Agora***

### Entry Consistency

In EC, processes must synchronize via system-supplied primitives. [7]. The synchronizations are *acquires* and *releases*. In EC, once an acquire is complete, it is ensured that the process sees the most recent version of data for the acquired synchronization variable. Exclusive locks, read-only locks and barriers are some of the synchronization primitives that should be associated with shared data. This leads to additional programming effort. ***Ex: Midway***

### Release Consistency

Weak Consistency with two types of synchronization operators such as *acquire* and *release*. Each type of operator is guaranteed to be processor consistent. [5] ***Ex: Dash***

In this paper the main emphasis is on comparing the different Release consistency Models. Though Sequential Consistency models have an advantage that DSM behaves in a way expected by the programmers, the cost involved in implementing it is high. The central manager that has to keep track of the location of each page's owner is a bottleneck and invalidation-based algorithms may give rise to thrashing. Release Consistency was implemented as an effort to reduce such DSM overheads.

## 3 RELEASE CONSISTENCY MODELS

The main idea of implementing Release Consistency Semantics is the reduced cost and also that it is tractable to programmers. In release consistency, the programmers are aware of the synchronization objects such as semaphores, locks and barriers. This strategy is a weak consistency model in the sense that memory is allowed to be inconsistent at certain points, while the use of synchronization objects help to preserve application-level consistency.

A Release - consistent memory was designed to fulfill the following requirements.

1. All *acquire* accesses must be performed before an ordinary read or write operation is allowed to perform

2. All previous read and write operations must be performed before a *release* operation is allowed to perform.
3. Acquire and Release operations should be *sequentially consistent* with respect to each other.

An example of how release-consistent can be made to appear as Sequentially consistent DSM is shown in figure 2.

**Figure 2**

```
Process 1:
        acquireLock();              //enter critical section
        X: = X + 1;
        Y: = Y+1;
        releaseLock();              //leave critical section

Process 2:
        acquireLock();              //enter critical section
        print X,Y;
        releaseLock();              //release critical section
```

In this example, the critical sections enforce consistency at the application level. If process1 acquires the lock first, process 2 will not cause any activity until it gets the lock. But in sequentially consistent memory, process 1 would block when it updated X and Y. Whereas in Release-consistent memory it will not block while operating on the shared variables. Here communication is required only when locks are releases. But for a release consistent memory it is the programmer's responsibility to indicate the read and write operations as release, acquire or non-synchronization accesses. Once the program is written with such information, it cannot differentiate between a sequential DSM and Release Consistent DSM.

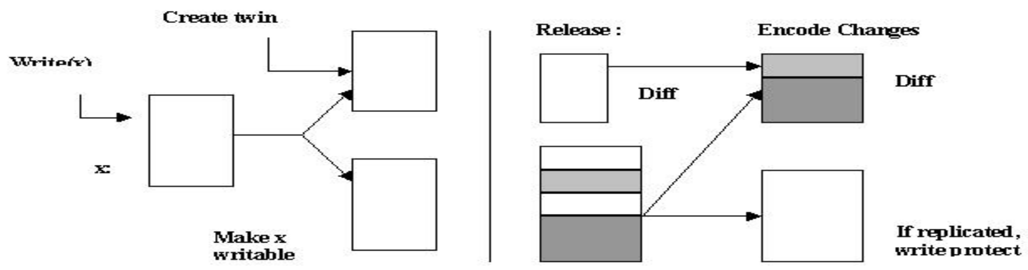
Following are few of the Software-based release Consistent Protocols

- Eager Invalidate Protocols
- Lazy Invalidate protocols
- Lazy Hybrid protocols

Before studying in detail about each of these, it is important to know how multiple-writer protocols work under release consistency.

Modifications of the local copies of a shared page are summarized as *diffs*. Initially any shared page is write-protected. When the first write access occurs, there is a protection violation. The DSM software makes a copy of the page (a twin) and removes the write-protection so that further writes to the page can occur without DSM intervention. [10] A diff is then created between the twin and the modified page, by making a run length encoded record of the modifications of the page. Diff creation is shown in figure 3.

Figure3- **Diff Creation** [10]



### 3.1 Eager-Invalidate Protocol (EI)

In an Eager Protocol, modifications to shared data are made visible globally at the time of a *release* [10]. An Eager Invalidate protocol as the name implies attempts to invalidate remote copies of a shared page if the page has been modified locally. If the page was open in a Read-only mode, then just invalidation will do. But if it was in a read-write mode, a diff, which has the modifications, is received as part of the reply from the remote node and then the local copy is invalidated. Thus diffs from different remote sites a re collected and sent to all other processors in the system which have the remote copies of the page.

To determine the remote copies that need to be invalidated, the EI protocol uses an approximate *copy set.* A copy set is a bit mask indicating which processors have a copy of the page. [10] When the remote processors communicate for invalidations, even the copy set is attached as part of the acknowledgements to the invalidate messages. This keeps the local copy sets also up-to-date. This will help in continuous learning about the remote copies of the shared page so that additional protocol rounds are done to ensure that all remote copies are invalidated.

### 3.2 The Lazy Invalidate Protocol (LI)

With a lazy protocol, the propagation of modifications is postponed *until the time of the acquire.* [10] The execution of each process is said to take place in *intervals* each represented by an interval index. Among different processors these intervals are partially ordered while on the same processor they are totally ordered. To represent the partial order, a *vector timestamp* is assigned to each interval. A processor computes a new vector timestamp at an acquire according to the pair-wise maximum of its previous vector timestamp and the release's vector timestamp [10].

Consider two processes p and q on different processors modifying some shared page.
The requirements of RC would be to follow the below steps.

- The updates of all intervals with a vector timestamp than *q's* current vector timestamp must be visible at *p* before *p* may continue past an acquire from *q*
- At an acquire, *p* sends its current vector timestamp to the previous releaser, *q*.
- Processor *q* piggybacks on the release-acquire message to *p* and *write notices* for all intervals named in *q*'s vector timestamp except the one it received from *p*.
- The page is invalidated when a *write notice* arrives.

*Write notice* only indicates that a page has been modified but does not contain the modifications. To obtain the modifications made to the page, *diffs* need to be created. (As explained previously) Diffs are created when modifications are requested by a processor or a write notice arrives for a dirty page [10]. Only when an acces miss occurs due to an attempt to access an invalidated page, the diffs created from all intervals before the faulting interval are retrieved and applied to the page. This line of operation largely reduces the number of messages.

## 3.3 The Lazy Hybrid Protocol (LH)

The basic difference between Lazy Invalidate and Lazy Hybrid protocols are that in Lazy Hybrid protocols, few pages are even updated at the time of an acquire. LH attempts to exploit temporal locality by assuming that any page accessed by a processor in the past will probably be accessed by that processor again in the future. [10] Hence if sharing patterns are known a priori , this protocol may be used to optimize the communication required.

Here a copy set is used by each processor to keep track of the pages accessed by other processors. The copy set also helps the in deciding whether a diff must be sent to a remote location. There were different possibilities to determine the set but from heuristics it was observed that it is the best to look at every diff pertaining to a write notice that is sent. For each such write notice, if the releasing processor has the diff and the acquiring processor is in the local copy set for that page, the diff is appended to the lock grant message.

## 3.4 Protocol Trade-Offs [10]

To choose between these Release Consistent protocols, a complex trade-off has to made between

- Number of Access misses
- Number of messages
- The Amount of Data
- Lock acquisition time
- Protocol overhead

**Number of Access misses**

Number of access misses is more in the case of EI, which may result due to false sharing Then next comes LI, which may experience more access misses than LH.

**Number of Messages**

EI requires more messages then LI and LH. Because in a lazy model that involve locks communication is between two synchronizing processes whereas release in an eager system requires invalidations to be sent even to processes that are not involved in the synchronization.

**Amount of Data**

LI and LH usually exchange data in the form of diffs, while EI moves entire page. Hence amount of data is more in the case of EI.

**Lock Acquisition time**

In EI, to release a lock the processor all the invalidations are sent and acknowledged. This results in lock acquisition latency. In LI and LH, lock transfer involves communication between two processes at a time. In LH, updates are appended to the lock grant message. The time taken to generate and process this data might cause some latency in acquiring the locks.

**Protocol Overhead**

EI is less complex than LI and LH because unlike the other two protocols all state concerning the modified page can be discarded once a release is made in EI. LH and LI create more diffs than in EI.

## 4 COMPARISON OF CONSISTENCY MODELS

A comparison has been made on three main Consistency Protocols based on the experiments conducted by researchers in Princeton University and University of Wisconsin, Madison. The protocols compared are:

- Sequential Consistency model (SC)
- Single Writer LRC protocol (SW-LRC)
- Home-based LRC protocol (HLRC)

### 4.1 Description of the system considered for comparison [13]

- Coherence granularity at block level (64, 256, 1024, 4096 bytes)
- Each block has a *home* which will be the first node that "touches" the block
- A page's home node ID is kept in a distributed table and cached in a local table
- Applications considered

  ***LU*** - Performs blocked LU factorization of a dense matrix
  ***FFT*** - High-performance kernel to handle a matrix of processors
  ***Ocean*** - Simulates eddy current in ocean basin-challenge is memory allocation for data in each sub-grid
  ***Water-Nsquared*** - Simulation of a system of water molecules that are allocated to an array of processors and challenge is to handle updates at each processor.
  ***Volrend*** – Application to render 3-D volume data into an image, challenge is the partition of tasks

***Water-Spatial*** – Same as Water-Nsquared but with different data structures and different algorithms
***Raytrace*** - To handle complex scenes in computer graphics, challenge is When distributed task queues are used for task stealing
***Barnes*** – Simulation of interaction among a system of particles over a number of time steps

**Table2 – Comparison of Different Consistency Models [13]**

| **Parameters** | **SC** | **SW-LRC** | **MW-HLRC** |
|---|---|---|---|
| ***Co-existence of Readers and Writers*** | Permitted to have either a single-writer or one or more readers; readers and writers never co-exist a the same time [13] | Can have multiple readers but only a single writer can co-exist with them | Multiple writers and Multiple readers allowed |
| ***Consistency Guarantees*** | Delay until Write is globally performed | Writes propagated at next acquire | Writes propagated at next acquire |
| ***On a Write-Fault*** | Read-only copies invalidated. | Only ownership of the page is migrated on a write-fault | As multiple writes are allowed, twins and diffs are created for keeping consistency |
| ***Protocol Overhead*** | High | Less | Higher than SW-LRC |
| ***Speedup based on Granularity*** | At 64-byte granularity SC outperforms LRC, Suffers performance losses at coarse granularity | At large granularities, read-write false-sharing more probable, but SW-LRC performs better than SC because invalidations are delayed until next acquire. | Here at large granularities, even write-write false sharing is taken care of. Also number of write misses are minimized compared to other protocols. |

## 5 ADAPTIVE DSM PROTOCOLS

In 1997, Cristiana Amza et al., published a paper titled " Software DSM protocols that Adapt between Single writer (SW) and Multiple writer (MW)". The adaptivity of the system is based on the nature of the scenario. The decision to choose between Single writer and Multiple writer protocol is done by analyzing if a particular page exhibits Write-Write False Sharing (WFS) or Write Granularity (WG).

### 5.1 Background

As the name implies, a Single Write protocol allows only a single writable copy of a page at any given time. [11] whereas in Multiple Write Protocols many writable copies of a page can co-exist. In SW protocol, the processor currently holding write privileges to the page is called the *owner.* Each page has a *version number*, which is incremented every time ownership is acquired. Experiments were conducted on an 8-node SPARC cluster connected by 155 Mbps ATM network. The adaptive protocols were compared with MW only or SW only implementations.

### 5.2 The Adaptive Protocols

Even within a single application, there may be pages, which behave differently when it comes to sharing. The nature of sharing of a page can be dynamically analyzed and the page can adapt to a suitable protocol between SW and MW protocol to achieve good performance.

**When to go for MW protocols?**

When a page is detected to exhibit Write-Write False sharing (WFS) problem, it will be best to perform with MW protocol. In MW protocol, more than one processor can simultaneously modify a page and the modifications can be lazily merged in the sense, until the next synchronization operation and this reduces the effect of false sharing.

**Detecting Write-Write False sharing in Single Write mode**

It is not efficient to use SW protocol when Write-Write false sharing is observed. In such a scenario, the page can switch to MW protocol. The principle behind the detection of WFS is:

*There is no Write-Write False Sharing if and only if the processor taking a write fault and trying to get ownership knows the correct location of the owner and the correct version number for the page.* [11]

**When to go for SW protocols?**

A page switches to SW mode from MW mode when the Write Granularity is above a threshold value. The researchers say that they use a threshold value to decide whether or not to use diffs [11].

**Detecting Absence of Write-Write False Sharing in MW mode**

Again to go for SW mode, absence of Write-Write False sharing has to be detected.
The principle is:
*There is no write-write false sharing if there is a write notice for the page that dominates all other write notices.* [11]

The steps to ensure such detection are as follows:

- Each processor piggybacks information indicating whether the page is perceived as write-write falsely shared or not according to the write notices they receive.
- The writer updates its local information whenever a diff request comes in
- By collecting such information, an approximate copy set is maintained.
- A processor can detect that WFS has stopped when a processor sees a new owner notice and no concurrent secondary write notices.
- Also at a barrier, when a processor detects a write notice for a page that dominates all other write notices, it can infer that WFS has stopped.

## 6 Discussions and Conclusion

After studying the different Release Consistency models and also the Adaptive protocols, it is observed that many issues can be over come by going in for an adaptive model. It is observed that performance of any consistency model largely depends on the data access pattern and synchronization behavior of an application. But the important inference is that, for smaller page sizes SC outperforms any LRC. But for coarse granularities, LRC seems to perform better than SC. The research conducted by Cristiana Amza showed promising results for Adaptive protocols. A state variable was associated with each processor to indicate which protocol best suited a page in SW or MW mode. With the experiments the following observation (Table 3) were made on a few parameters.

**Table3 – A Comparative Account**

| Observed parameters | Non-Adaptive Protocols | Adaptive Protocols |
|---|---|---|
| ***Execution Times*** | Speed up is low | Speedup is comparatively high |
| ***Memory Overhead*** | Except in SW mode, where diffs and twins are not created, memory over head is significant | In MW mode, when WFS is more, memory over head is more. |
| ***Communication*** | For SW, ownership requests may involve forwards that contributes to the contribution overhead | For WFS + WG protocols, ownership related messages are double the number of ownership requests |

In 1994, a paper was published by researchers at University of California at Santa Barbara. This paper talks about Mixed Consistency. In this model, two kinds of weak memory consistency conditions are combined. They are causal memory and pipelined random access memory. Also conditions under which mixed consistency leads to the same results as sequentially consistent memory were investigated. A new platform called Maya was developed to evaluate different memory consistencies by running different applications on it. Currently research is going on in investigating other applications to evaluate the performance in a mixed consistent DSM environment.

## References

[1] Mixed consistency  Divyakant Agrawal , Manhoi Choy , Hong Va Leong , Ambuj K. Singh
Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing August 1994

[2] Kai Li and Paul Hudak, Memory Coherence in Shared Virtual Memory Systems, ACM Transactions on Computer Systems, Vol. 7, No. 4, November 1989

[3] Ajay Mohindra and Umakishore Ramachandran, A Comparative Study of Distributed Shared Memory Design Issues, GIT-CC-94/95, August 1994.

[4] Sarita V. Adve,  Kourosh Gharachorloo, WRL Research Report 95/7: Shared Memory Consistency Models: A Tutorial, September 1995.

[5] B Nitzberg and V Lo, Distributed Shared Memory: A Survey of Issues and Algorithms, IEEE Computer August 1991, pp. 52-60.

[6] Pete Keleher, Lazy Release Consistency for Distributed Shared Memory, PhD thesis of Rice University, Huston, Texas, January, 1995.

[7] S. Adve et al. A Comparison of Entry Consistency and Lazy Release Consistency Implementations. In IEEE HPCA, February 1996.

[8] John Hennessy, Mark Heinrich and Anoop Gupta, Cache-Coherent Distributed Shared Memory: Perspectives on Its Development and Future Challenges, Proceedings of the IEEE, VOL. 87, No. 3, March 1999.

[9] Masaaki Mizuno, Michel Raynal, James Z. Zhou, Sequential Consistency in Distributed Systems, Proc. of the Int'l Workshop on Theory and Practice in Distributed Systems, October 1994.

[10] Pete Keleher, Alan L. Cox, Sandhya Dwarkadas, and Willy Zwaenepoel. An evaluation of software-based release consistent protocols. Journal of Parallel and Distributed Computing, 29(2):126--141, September 1995.

[11] Cristiana Amza, Alan L.Cox, Sandhya Dwarakdas and Willy Zwaenepoel Software DSM protocols that Adapt between Single Writer and Multiple Writer 1997

[12] Pete Keleher, Sandhya Dwarkadas, Alan Cox, and Willy Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In Proceedings of the 1994 Winter Usenix Conference, pages 115--131, January 1994.

[13] Yuanyuan Zhou, Liviu Iftode, Jaswinder Pal Singh, Kai Li : Relaxed Consistency and Coherence Granularity in DSM Systems: A Performance Evaluation : Published in the proceedings of the 6th ACM symposium on Principles and Practice of Parallel Programming, 1997

[14] Cristian Amza, Alan Cox, Karthick Rajamani, and Willy Zwaenepoel, Tradeoffs between False Sharing and Aggregation in Software Distributed Shared Memory, Proceedings of ACM SIGPLAN Conference on Principles and Practices of Computer Programming, 1997

[15] J. Carter, J. Bennet and W. Zwaenepoel, Techniques for reducing Consistency-Related Communication in Distributed  Shared Memory Systems, ACM Transactions on Computer Systems, Vol. 13, No. 3, , August 1995.