World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:3, No:2, 2009

Consistency Model and Synchronization Primitives in SDSMS

Dalvinder Singh Dhaliwal, Parvinder S. Sandhu, and S. N. Panda

***Abstract*—**This paper is on the general discussion of memory consistency model like Strict Consistency, Sequential Consistency, Processor Consistency, Weak Consistency etc. Then the techniques for implementing distributed shared memory Systems and Synchronization Primitives in Software Distributed Shared Memory Systems are discussed. The analysis involves the performance measurement of the protocol concerned that is Multiple Writer Protocol. Each protocol has pros and cons. So, the problems that are associated with each protocol is discussed and other related things are explored.

***Keywords*—**Distributed System, Single owner protocol, Multiple owner protocol

Open Science Index, Computer and Systems Engineering Vol:3, No:2, 2009 publications.waset.org/9612/pdf

I. Introduction

A *distributed system* is an application that executes a collection of protocols to coordinate the actions of multiple processes on a network, such that all components cooperate together to perform a single or small set of related tasks.

II. Memory Consistency Model

A memory consistency model is a contract between programmers and shared memory which specifies how the memory operations of a program will be executed. Computer scientists have proposed different memory models to enhance distributed shared memory systems. In this section, we will give an introduction to those memory models and some terminology used in this paper.

A. Strict Consistency

Strict consistency is the most stringent memory model. It requires any read operation to a memory location to return the latest write. This definition uses a global time to define what a read operation can get from the memory. This model mimics the memory behavior in a single processor.

Dalvinder Singh Dhaliwal is working with Computer Science & Engineering Department, RIMIT Institute of Engineering & Technology, Mandi Gobindgarh (Punjab), India

Parvinder S. Sandhu is Professor with Computer Science & Engineering Department, Rayat & Bahra Institute of Engineering & Bio-Technology, Sahauran, Distt. Mohali (Punjab)-140104 India (Phone: +91-98555-32004; e-mail: parvinder.sandhu@gmail.com).

S. N. Panda is is working as Principal at Regional Institute of Management and Technology, Mandi Gobindgarh (Punjab), India

.

B. Sequential Consistency

A global clock is hard to capture in a distributed system. Each processor in the distributed system may have its own local clock with a different view of time. The idea of recent time can be inconsistent in the system. proposed another model, called *sequential consistency*, to extend the idea of the strict consistency model.

C. Processor Consistency

Processor consistency [6],[8] was proposed to relax the program order constraints in the case of a write followed by a read operation to a different location. It allows the read operation to bypass the write before the write is serialized or made visible to other processors [1].

D. Weak Consistency

One family of relaxed memory models requires programmers to distinguish between data and synchronization operations. Accesses to synchronization variables are strongly ordered (for example, totally ordered), but data accesses follow a weaker order. Weak consistency was the first hybrid memory model proposed by [5], [7].

E. Release Consistency

Release consistency [6] is an extension of weak consistency. Release consistency classifies operations on shared memory into two categories, *special* and *ordinary*. *Special operations* also are classified into *sync* and *nsync*. *Sync* operations are either *release* operations or *acquire* operations. *Ordinary operations* refer to data accesses without conflicting with other operations. *Sync* accesses are used to order data accesses such that data operations may not conflict with each other. *Nsync* accesses are asynchronous data accesses. *Release* is a write synchronization operation. *Acquire* is a read synchronization operation.

F. Lazy Release Consistency.

Even though the conventional release consistency allows ordinary accesses to be postponed until a release operation is executed, it still requires all ordinary accesses to be performed with respect to all processes. [9],[10] proposed the *lazy release consistency* which allows ordinary accesses to be performed with respect to some processes. (Because the lazy release consistency was implemented as a software distributed shared memory system, the term of process is used instead of processor.)

Open Science Index, Computer and Systems Engineering Vol:3, No:2, 2009 publications.waset.org/9612/pdf

G. Message-Driven Relaxed Consistency

In order to exploit both the message passing model and the shared memory model, [11] proposed *message-driven relaxed consistency* that combines those two mechanisms into a single system. Messages carrying explicit causality annotations are exchanged to trigger memory coherence actions. In addition to shared memory, message passing is another mechanism provided by the system to exchange information among processes. Specifically, if a process sends a synchronization message to another process, the modifications in shared memory the sending process made are visible to the receiving process after the receiving process gets the message. If all messages are synchronization messages, the ordering of memory events is consist with the happened before" relation, as defined by [12].

III. Summary

The figure below represents the consistency model.

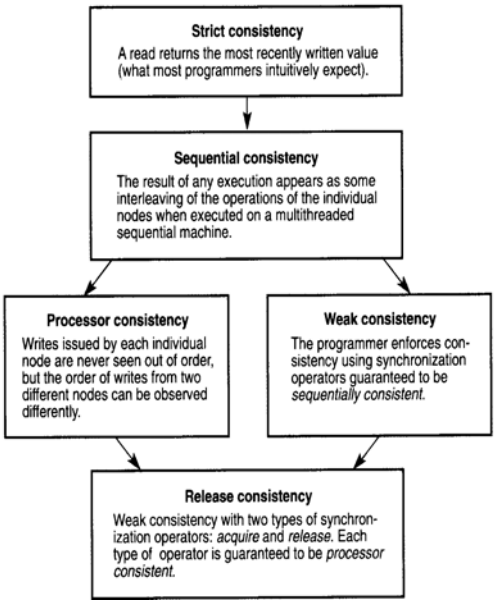


Fig. 1 Summary of consistency model

IV. Related Techniques for Implementing DSM System

In this section, brief introduction to the implementation techniques of software distributed shared memory systems are given. In the Fig. 2 shown below the term SWP represents single writer protocol & another term MWP stands for multiple writer protocol.

A. Single Owner Writer Protocol

It uses the page-based mechanism to implement a software distributed shared memory system called IVY

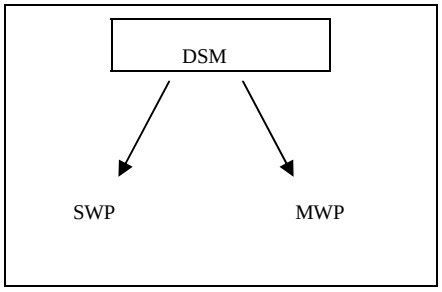


Fig. 2 DSM partition

i. IVY: It implements sequential consistency, Only one process is allowed to write the page at any point in the execution. This simply means that while all processes can read the same page but only one process can write During that write operation, the local cache of that process is set to read-write mode (if the writer does not get current version of the page, it gets the same (by invalidates all the existing copies of the page in all processes) from another process before it writes.

Concluding above, To write, all the other processes are set to be in OFF state.

B. Multiple Writer Protocol

To minimize the size of the problem, we opt MWP (as in SWP, the size of the response is equal to the size of the physical page in memory).It uses MWP release consistency.
An ordinary write operation is performed by sending the final values of the variables to all the processes. The variables are cached before the release operation is performed.

In MWP, let us suppose we have two processes called 'p' & 'q'. further suppose if p releases the control it passes over the changes to process q, and when q releases the operation, it gives the changes to p. Finally ,the caches of the pages become identical.

C. Diff

The change in the page before the release is called the diff of the page. Diff consists of two attributes, i.e, addresses of a particular change to be made and values that will actually make change happen.

Result: So when two processes wish to write the same page, then only the diff of the page is set, rather than the whole page as done in SWP. Thus page size or request size reduces.

V. Lazy Invalidate Protocol

All ordinary accesses to be performed w.r.t. all processes even though some processes do not access those variables. Tread mark proposed LIP.

The conventional release consistency requires all ordinary accesses to be performed with respect to all processes even though some processes do not even access those variables. Tread Marks [9],[14] proposed *lazy release consistency* and its implementation, called *lazy invalidate* protocol .The execution of an application in the lazy release consistency is partitioned into intervals. The intervals of different processes are partially ordered:

Open Science Index, Computer and Systems Engineering Vol:3, No:2, 2009 publications.waset.org/9612/pdf

i. Intervals in a single process are totally ordered by its program order.

ii. An interval of process *p* precedes an interval of process *q* if the interval of *q* begins with the acquire operation corresponding to the release operation that concluded the interval of *p*.

The lazy invalidation protocol is also a multiple writer protocol. It invalidates the caches of the shared memory according to *write notices*. A *write notice* is created for a written page by the writer when the process executes a release operation. Each write notice lists the information about the page number and the specific interval when the page was modified. There is a diff associated with each write notice. The diff keeps the changes of the page which the process writes since the local copy of the page in the process is made valid [15]. A process performs an acquire operation on a variable by sending an acquire request to the last process which performed a release operation on the variable. The releasing process responds to the acquiring process with a set of write notices. These write notices were created in the intervals preceding the acquiring process's new interval .The acquiring process invalidates its local copy of a page if there is a write notice for that page and the write notice was not used to invalidate the page before. When the process attempts to access an invalid cache, it first checks whether it has kept all the diffs corresponding to the write notices which invalidated the cache. If not, the process sends messages to some of the processes which created the write notices. The reason why it need not send messages to all of the processes can be shown from the following example. The interval when a write notice is created by process *p* may precede the interval of another write notice created by process *q*. Process *q* must keep process *p*'s diff because process *q* needs *p*'s diff to make the local cache valid before process *q* can write to it. The process subsequently accessing the page needs only to send a requesting message to the last writer process, process *q*. After the accessing process receives all the responses, the accessing process applies the diffs to the cache in the partial order. Then the computation resumes. All the processes keep these write notices and diffs locally until garbage collection is performed.

VI. Problems with MWP

A large number of invalidate inconsistent copies of a page exist in the system as shown in Fig. 3.

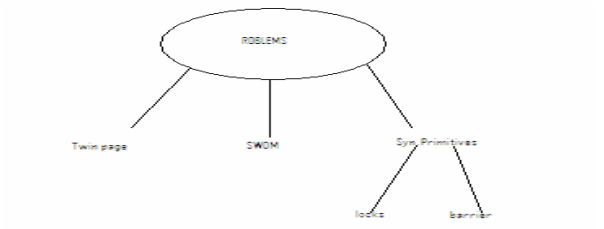


Fig. 3 Problems with MWP

VII. Synchronization Primitives In Software Distributed Shared Memory System

A. Locks

Munin,[4] uses the *probable owner* mechanism to implement locks. Each process maintains its observations about which process might own the lock. If the lock is not available locally, a requesting message is sent to the probable owner. If the probable owner does not have the lock, the probable owner forwards the requesting message to its probable owner. The message is passed along the probable owner chain to the last lock holder. If the lock is free, the last lock holder gives the lock to the requesting process and sets its observation of the probable owner to the requesting process. Tread Marks uses a distributed queue to implement a lock. Each lock has a specific manager which knows the most recent process, *p*, requesting the lock. A global waiting queue is maintained. When the manager receives a lock request from a process, *q*, it forwards this request to *p*. The manager also sets the most recent process to *q*. *p* passes the lock and invalidation information to *q* when *p* releases the lock.

B. Poor Application Programming Interfaces for Solving Synchronization Problems

The synchronization operations of *release consistency* are restricted to *release* and *acquire*, which are write and read operations on shared memory. The overhead of the strong memory consistency is not only high at run time in NOW but it is hard to program using just release and acquire. Instead of implementing a strong memory consistency model, most of contemporary software distributed shared memory systems offer higher level synchronization primitives, such as locks and barriers, and implement them by synchronization managers. In order to conform with the definition of release consistency, the developers of distributed shared memory systems usually need to spend some effort in associating locks and barriers with release and acquire operations. For example, getting a lock is an acquire operation and returning a lock is a release operation [10]. However, interpreting a barrier is not as intuitive as a lock. Therefore, using the notions of release and acquire to describe synchronization accesses is problematic. Moreover, the complexity of using these basic synchronization operations of locks and barriers to solve some synchronization problems is known to be quite complicated for programmers and prone to errors. This is a classic discussion appearing in many operating system text books, for example [13] .

C. Weaknesses of Multiple Writer Protocol

Write shared process and lazy invalidate protocols of tread marks are called multiple owner protocols. In those, a reader needs to contact some writers that own a piece of current data.

Problems with MOP

- Large number of diffs to maintain consistency of a page which is written by some processes and read by some others.
- Diff accumulation

- Garbage collection: It is performed by stopping execution of all processes and making active pages current in each process.

D. Performance Issue

The papers [3],[4] identify a class of applications that do not perform well on some distributed shared memory systems. For example, a parallel version of the travelling salesperson problem uses a branch-and-bound algorithm to find the shortest path. The algorithm uses a priority queue to store incomplete paths. The priority queue is protected by a lock. As stated in [3] the major source of overhead for these DSM versions was the amount of times spent waiting on the lock protecting the work queues. These lock waiting times are large because the DSM versions must ship the work queue, a sizable data structure, to the acquiring process before that process can perform any operation on the work queue."

A *function shipping mechanism* was proposed by [3] such that the priority queue remains attached to a specific process. Accesses to the priority queue by other processes are performed by remote procedure calls. However, there was no systematic way to incorporate such features into their system. Message-driven relaxed consistency can implement the RPC-server for the priority queue without too much overhead by creating a process which receives requests from other processes in the form of messages. However, this approach inherits the disadvantages of the message passing models, which are difficult for users to program.

REFERENCES

[1] S. V. Adve and K. Gharachorloo. Shared memory consistency models: A tutorial. Technical report, Rice University ECE, 1995.
[2] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. the NAS parallel benchmarks. Technical Report RNR-94-007, NASA, 1994.
[3] J. B. Carter. Efficient Distributed Shared Memory Based on Multi-Protocol Release Consistency. PhD thesis, Rice University, 1993.
[4] J. B. Carter. Design of the Munin distributed shared memory system. Journal of Parallel and Distributed Computing on Distributed Shared Memory, 1995.
[5] M. Dubois and C. Scheurich. Memory access dependencies in shared-memory multiprocessors. IEEE Transaction on Software Engineering, June 1990.
[6] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In the Proceedings of the 17th Annual International Symposium on Computer Architecture, 1990
[7] M. Dubois, C. Scheurich, and F. A. Briggs. Memory access buffering in multiprocessors. In Proceeding of 13th Annual International Symposium on Computer Architecture, 1986.
[8] J.R. Goodman. Cache consistency and sequential consistency. Technical Report 61, SCI Committee, 1989.
[9] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy release consistency for software distributed shared memory. In the 19th Annual International Symposium on Computer Architecture, 1992.
[10] P. Keleher. Lazy Release Consistency for Distributed Shared Memory. PhD thesis, Rice University, January 1995.
[11] P.T.Koch, R. J. Fowler, and E. Jul. Message-driven relaxed consistency in a software distributed shared memory. In Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation, 1994.
[12] L.Lamport. How to make a multiprocessor computer that correctly executes multi process programs. IEEE Transactions on Computers, 28(9):690{691, November 1979.
[13] A. S. Tanenbaum. Modern Operating Systems, chapter 2. Prentice Hall, 1992.
[14] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Raja-mony, W. Yu, and W. Zwaenepoel. TreadMarks: Shared memory computing on networks of workstations. IEEE Computer
[15] P. Keleher, A. L. Cox, S. Dwarkadas, and W. Zwaenepoel. Tread-Marks: Distributed shared memory on standard workstations and operating systems. In the 1994 Winter USENIX Conference, 1994.

Open Science Index, Computer and Systems Engineering Vol:3, No:2, 2009 publications.waset.org/9612/pdf