Journal of Computer Science 3 (1): 57-61, 2007
ISSN1549-3636
© 2007 Science Publications

Distributed Shared Memory Consistency Object-based Model

Abdelfatah Aref Yahya and Rana Mohamad Idrees Bader
Faculty of Science and Information Technology, Al-Zaytoonah University of Jordan
P.O.Box 130, Amman 11733 Jordan

Abstract: A novel model that describes consistency in shared memory was developed, presented and discussed. The new object-based model handles errors of inaccuracy and misrepresentation in distributed shared memory process. The issue of misalignment was also covered.

Key words: Distributed, communication, object, memory, consistency

INTRODUCTION

Distributed shared memory (DSM) systems represent a successful hybrid of two parallel computer classes: shared-memory multiprocessors and distributed computer systems. They provide the shared memory abstraction in systems with physically distributed memories and consequently combine the advantages of both approaches[1].

A consistency model is essentially a contract between the software and the memory. It says that the software agrees to obey certain rules and the memory promises to work correctly. If the software violates these rules, all bets are off and correctness of memory operation is no longer guaranteed[2,3].

Different techniques have been suggested to keep the consistency of the shared data. The techniques are differentiated according to synchronization. The consistency models that do not use synchronization are also be known as "Strong consistency models"[2,4]. There are several models of this type, such as:

* Strict consistency: Absolute time ordering of all shared accesses matters.
* Sequential consistency: All processes see all shared accesses in the same order.
* Causal consistency: All processes see all casually related shared accesses in the same order.
* PRAM consistency: All processes see a "write" from each processor in the order they were issued. Writes from different processors may not always be seen in the same order.
* Processor consistency: PRAM consistency + memory coherence.

On the other hand the synchronization consistency models are known as "Relaxed memory consistency"[2,5,6]. Examples of such models are:

* Weak consistency: Shared data can only be counted on to be consistent after synchronization is done.
* Release consistency: Shard data are made consistent when a critical region is exited.
* Entry consistency: Shared data pertaining to a critical region are made consistent when critical region is entered.

In this study the problem of deviation and offset in a distributed shared memory (DSM) is solved using a novel object-based process and algorithm. The environment of the system in this model is a pure distributed machine, since each processor works as a server and once again works as a client, depending on the operation it will perform.

Design and development: The following design issues where examined for this model:

a. 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.
b. Flexibility: In general it is flexible, since the micro kernel is used for interposes communication and helps memory management.
c. Reliability: it considered to be 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.
d. 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 ask for the shared object at the same time, the performance may become lower. Generally, in this model consistency is achieved over the performance.
e. Scalability: It may be not very scalable for a large system.

Corresponding Author: Abdelfatah Aref Yahya, Faculty of Science and Information Technology, Al-Zaytoonah University of Jordan, P.O.Box 130, Amman 11733 Jordan

Value	Counter	Busybit

Fig. 1: Structure of the shared variable object

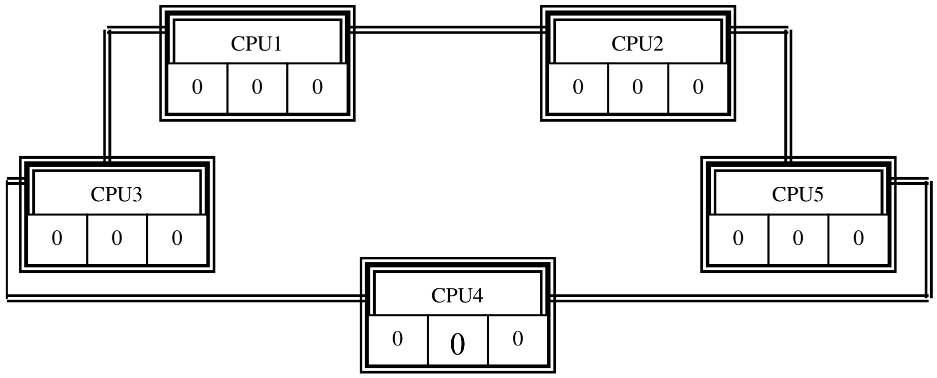

Fig. 2: The system in the initial state

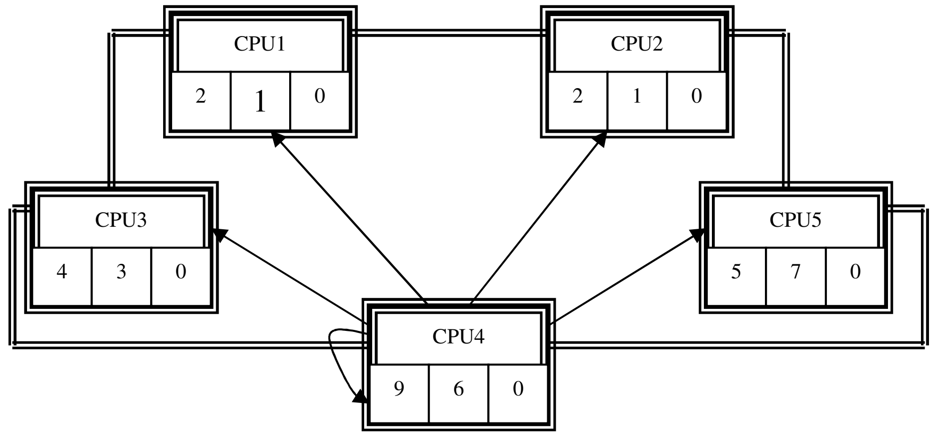

Fig. 3a: CPU4 asks other CPUs for their counter values, looking for the highest counter

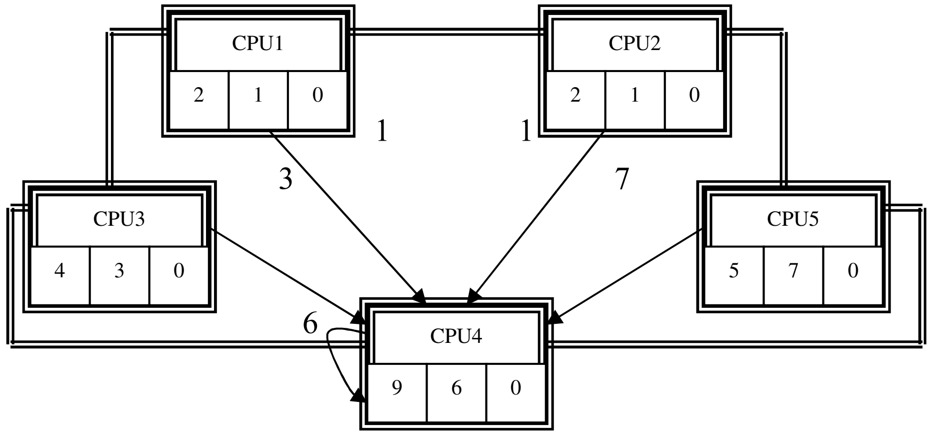

Fig. 3b: Each CPU5 returns its counter value to CPU 4

The object: In this model, the contract between the software and the memory says that the structure of the shared object is a record. The first field is the structure of the object itself. In this field, the value and the type of the object are defined. The second field, which is important to the consistency model, holds the counter value. This field indicates the number of modifications to this object. From this counter value, the most recent value can be determined by taking the value from the object associated with the highest value of the counter. A third field was defined to indicate the status of the object, whether it is busy or not. This field is denoted by busybit. Figure 1 shows the structure of this object.

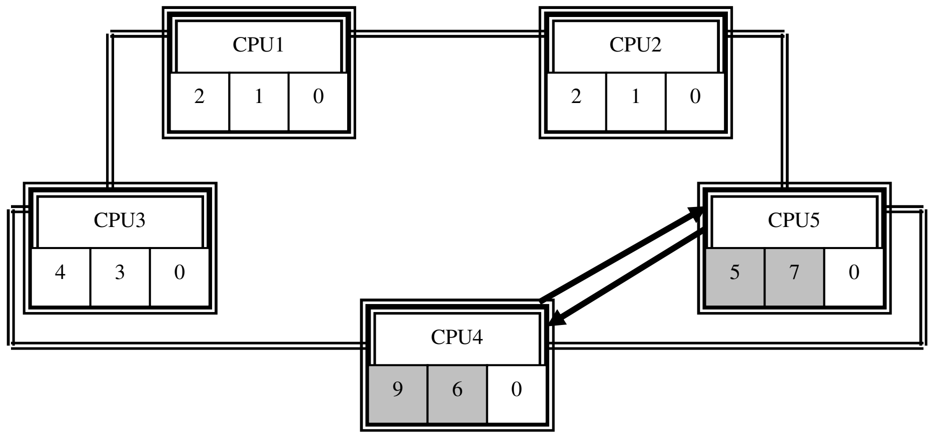

Fig. 3c: CPU4 asks CPU5 for its object value and then CPU5 answers it

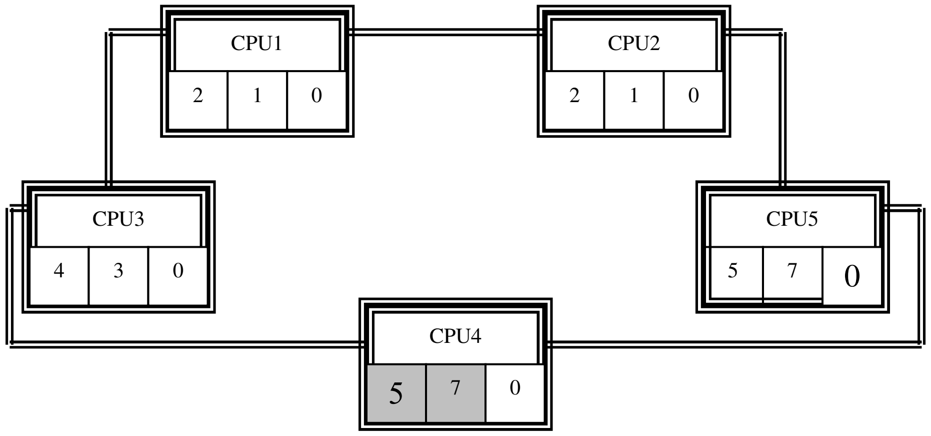

Fig. 3d: Final step. CPU4 changes its object value to 5, and its counter to 7, keeping the most recent value of the object. The busybit in CPU 4 becomes 1, indicating that it is busy

The system: In the proposed model, the system has (n) machines. The user of the system can determine the value of (n). Each machine has its own memory. These (n) machines are connected with each other through a network and each machine access the shared object. Each machine works as a server and as a client depending on the operations it performs.

All machines in the system have this shared object, where the initial value of the object and counter is zero. The busybit will also be initials zero (busybit = 1 means it is busy, busybit = 0 means it is free). Figure 2 shows the system in the initial state.

If one of the machines (say M1) wants to access the shared object, it should search for the highest value of the counter in all machines including itself. It may have one of the following cases:

(I) If all other machines have the same value or less than what M1 has, this means it has the most recent value (recall that the highest value of the counter returns the most recent value of the object).
(II) If another machine found is to be the one which has the highest value (say M2), then M1 should do the following:

a. A remote access occurs using the IP address of M2.
b. M1 checks the busybit in M2 if it is 0 or 1.
c. If busybit is 0, M1 asks for the value of the objects and the counter value in order to have the most recent value, otherwise it should wait and try again.

Figure 3a-d, show an example of the process of searching for the highest counter value and the return of the most recent value to maintain memory consistency. Consider in this example that the object is of type integer and CPU4 is attempting to write to this object.

There are three types of operations: a read only (RO), write only (WO) and read and write (RW). The type of an operation affects the status of the object to be free or not. Any object, M1, should determine the type of access it wants to have. Here is a brief description of how these operations are interpreted.

* Read only (RO): M1 has the most recent value and the value of counter and the busybit = 0. This means that this object is not busy, so any other machine can read this object in parallel with.
* Write only (WO): M1 has the most recent value and the value of counter and the busybit = 1. This means that this object is busy, so no other machine can use it as shown in Fig. 2d.

- Read and Write (RW): also M1 has the most recent value and the value of counter and the busybit = 1. This means that this object is busy and no other machine can use it as shown in Fig. 2d.

In (WO) and (RW) cases, after M1 finishes accessing the object it increments the counter by 1 and the busybit becomes 0 once again.

RESULTS

The Direct result of this work is an algorithm that deals with all the mentioned problems in existing systems. The algorithm answers the main question:

What happened if two machines ask for the most recent value at the same time? The solution comes from time principal: the one, which asked first, will gain access to it; the second machine will try again. If the system is large, this should be solved by partitioning the large system into subsystems, according to the number of machines and then the subsystems can apply the object-based model among the machines that receive the results of the subsystems for the first time[7].

DSM object-based algorithm

- We have N machines.
- If a machine (M1) wants to access the object, first it must determine the operation type.
- If the operation is Read Only (RO):
- M1 will search for the highest counter value in the system in order to have the most recent value of the object.
- It will get the object value from that machine which has the highest counter value.
- Set the busybit to 0, so any other machine in the system can use the object.
- If the operation is Write Only (WO):
- M1 will search for the highest counter value in the system in order to have the most recent value of the object as in (RO).
- It will get the object value from that machine which has the highest counter value.
- Set the busybit to 1, to prevent any other machine from accessing this object, by this it can access the object exclusively and store the new value into it.
- At the end, it adds 1 to the counter value and sets the busybit to 0.
- If the operation is Read And Write (RW), the same procedure as in point 4 will be executed, but M1 may store a new value into object instead of the old value.

DISCUSSION AND CONCLUSION

Memory consistency is one of the most important topics in Distributed Systems[8,9], since data must be as consistent as possible to keep the work in the distributed system acceptable and correct. In a general model, synchronization variables or synchronization operations are considered to give better performance and more utilization of the system, since they try to preserve consistency as much as possible.

Object-based model is classified as a "synchronization model". It uses a counter and a busy bit to maintain consistency, differ in that from the strong restrictive models and relaxed models. It may have less performance than the relaxed models, but it insures the consistency of shared objects.

In conclusion, a new object-Based DSM model is developed with the following characteristics[10,11]:

1. Pure consistent: any machine wants the shared object; it always has the most recent value of it. If one machine asks for (RO), say (M1) and then after a while another machine asks for (WR) or (WO), say (M2), then there is no problem in value consistency. Since M1 has really the most recent value before M2 starts, even when M2 change the value of the object, the value of M1 counter will give a clear view that it has the value before M2 and also it indicates that it does not have the most recent value now.
2. If one of the machines crashes, there will be no problem, because even if it has the most recent value, this value will crash with it. The machine, which has the highest counter now, will become the one, with the most recent value available to the system.
3. In the case when Busy bit = 1 and other machines want the object, they will simply try again.
4. It is efficient since it does not require a long time to ask for and receive the highest value of the counter and for the value of the object. Also, the requesting machine asks each of the other machines separately, which does not affect the work of these machines.
5. It does not waste bandwidth since there is no need to pass the most recent value to all other machines. The one that needs a value can ask for it.
6. It is designed for multi-computers and can be implemented on multiprocessors.
7. It is user-friendly and easy to be programmed.

REFERENCES

1. Scott, M.L., 2005. Shared memory computing on clusters with symmetric multiprocessors and system area networks. ACM Trans. on Computer Systems, 23: 301-335, pp: 35.
2. Kistler, M. and A. Lorenzo, 2005. Improving the performance of software distributed shared memory with speculation. IEEE Trans. on Parallel & Distributed Systems, 16: 885-896, pp: 12.
3. Huh, J., J. Chang, D. Burger and G.S. Sohi, 2004. Coherence decoupling: Making use of incoherence. Proc. 11th Intl. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS-XI), pp: 97-106.

4. Katsinis, C., 2003. Models of distributed-shared-memory on an interconnection network for broadcast communication. J. Interconnection Networks, 4: 77, pp: 26.
5. Acacio, M.E., J. Gonzalez, J.M. Garcia and J. Duato, 2002. Owner prediction for accelerating cache-to-cache transfer misses in a cc-NUMA architecture. Proc. 2002 ACM/IEEE Conf. Supercomputing (SC'02).
6. Chaudhuri, M. and M. Heinrich, 2004. SMTp: An architecture for nextgeneration scalable multi-threading. Proc. 31st Ann. Intl. Symp. Computer Architecture (ISCA'04), pp: 124-135.
7. Krewell, K., 2003. Sun weaves multithreaded future. Microprocessor Report.
8. Tam K.K., 2004. Sun's Niagara pours on the cores: Early details revealed at hot chips 16. Microprocessor Report.
9. Krewell, K., 2005. Best Servers of 2004: Where Multicore Is the Norm. In Microprocessor Report.
10. Cain, H.W. and M.H. Lipasti, 2004. Memory ordering: A value-based approach. Proc. 31st Ann. Intl. Symp. Computer Architecture (ISCA'04), pp: 90-101.
11. InfiniBand Trade Association, InfiniBand Architecture Specification, Release 1.2, Oct. 2004. Available from http://www.infinibandta.org.