Towards the Hierarchical Group Consistency for DSM systems : an efficient way to share data objects

Laurent Lefèvre and Alice Bonhomme
LIP / INRIA RESO, ENS-Lyon, France
laurent.lefevre@inria.fr

Abstract

We present a formal and graphical comparison of consistency models, based on the programmer point of view. We conclude that most consistencies can be categorized into 3 models depending on their flexibility degree (none, 1 or 2). We propose a new consistency model that provides these 3 flexibility degrees : the Hierarchical Group Consistency (HGC) and present its deployment inside the DOSMOS DSM system.

Keywords : *Distributed Shared Memory, consistency models, DOSMOS*

1. Introduction

DSM systems are now a well recognized alternative for the deployment of large class of applications. Their main challenge is to manage data consistency while keeping good performances. During last decade, a lot of consistencies have been proposed [2,3,7,10]. However, their definition is made from the designer or hardware point of view. Each definition is thereby often dependent on the DSM designer context. By using large scale clusters (hundred or thousand of nodes), DSM have to face the scalability problem. How to provide scalable solutions for applications needing a virtual shared space with a large number of computing nodes ? In particular, one issue that DSM systems have to address concerns minimizing replication (ie reducing the number of messages required to keep the memory consistent) and maximizing availability (ie increasing the number of local accesses) of shared data (objects).

Our goal is to propose a new taxonomy for consistency models, that can be used by a programmer that deals with the implementation of a distributed application on top of a cluster of machines. This taxonomy is programmer-centric. For this, we use the framework introduced by *Hu et all.* [9], and we show, that from the programmer point of view, there are only 3 main consistency models. The other ones are just various implementations of those models.

Based on this observation, we propose a graphical representation of those models. This representation is based on different degrees of flexibility of the consistency : flexibility about consistent moment (like Release Consistency) or about consistent data (like Entry Consistency). This representation outlines that so far, there is no consistency model with flexibility about consistent processes.

Hence, we suggest the Hierarchical Group Consistency (HGC). Consistency is only maintained inside a group of processes rather than between all processes. Furthermore, for each group, various consistency rules can be applied, depending on the type of sharing performed inside a group. This group structure is particularly interesting for heterogeneous clusters and allows the programmer to adapt the consistency management to the application (depending on the sharing degree, for example) and to the execution cluster (depending on the communication performances, for example). The Hierarchical Group Consistency has been implemented in the DOSMOS system [5,12]. Like in most DSM systems based on weak consistency, DOSMOS provides Acquire and Release operations to set critical sections. However, with HGC, DOSMOS also allows the programmer to specify which data to share and between which processes.

This paper is organized as follows : Section 2 rapidly presents the programmer-centric framework used to define consistency models, and introduces a formal comparison of existing models. Then Section 3 is devoted to a graphical 3D representation of the 3 consistency models. Next, Section 4 presents the Hierarchical Group Consistency model. Section 5 describes the hierarchical group deployment in the DOSMOS system. Section 6 concludes this paper and presents future directions.

2. Three Memory Consistency Models from the Programmer Point of View

Research on data consistency has always been a hot topic because of its central position between parallelism, operating systems, distributed systems, etc. However, if lot of consistencies have been defined, the formal context of the consistency concept has not been clearly specified. As a consequence, comparing consistencies remains a difficult challenge [1,8]. In this section, we focus on the consistency concept based on the programmer point of view. That means that a consistency model is defined based on how it is perceived by the programmer rather than how it is managed or implemented by the system. We base our work on the formal framework introduced by *Hu et all.* [9] in order to compare different models. Section 2.1 introduces this framework and defines the most well known consistency within this framework. Section 2.2 shows how to compare these models with this framework.

2.1. Formal definition of consistency models

In their article [9], *Hu et all* define a memory model as follows :

Definition 1 *A* **memory consistency model** M *is a two-tuple* (C_M, SYN_M) *where* C_M *is the set of possible memory accesses (read, write, synchronization) and* SYN_M *is an inter-processes synchronization mechanism to order the execution of operations from different processes.*

The execution order of synchronization accesses determines the order in which memory accesses are perceived by a process. Accordingly, for each program, there are several possible executions. A program execution is defined as follows.

Definition 2 *An* **execution** *of the program PRG under consistency model M, denoted as* $E_M(PRG)$*, is defined as an ordering of synchronization operations of the program.*

With the ordering of synchronization operations, the execution of all related operations are also ordered. Thus, we define the synchronization order of an execution.

Definition 3 *The* **synchronization order** *of an execution* $E_M(PRG)$ *under consistency model* M*, denoted as* $SO_M(E_M(PRG))$*, is defined as the set of ordinary operation pairs ordered by the synchronization mechanism* SYN_M *of* M.

Hence, for any consistency model M, we can define C_M and $SO_M(E_M(PRG))$. C_M deals with how the programmer has to program, and SO_M gives the rules used to generate the result.

The basic Atomic Consistency (AC), the Sequential Consistency (SC) [11], the Release Consistency (RC) [7], the Lazy Release Consistency (LRC) [1], the Eager Release Consistency (ERC) [6], the Entry Consistency (EC) [3] and the Scope Consistency (SsC) [10] can all be defined within this framework. Furthermore, based on those definition, we easily outline that RC, LRC and ERC have the same definition. That means that LRC and ERC are different implementation of the RC model. In effect, from the programmer point of view, the results are the same for both implementation. Similarly, we state that SsC is a particular case of EC. As a result, in the following, we only consider AC, SC, RC and EC models.

Finally, *Hu et all.* define a correct program as follows :

Definition 4 *A program PRG is said correct for the consistency model* M*, iff for any possible execution* $E_M(PRG)$*, all ordinary conflicting accesses pairs are ordered by either the program order (PO) or by the synchronization order of execution* SO_M.

2.2. Formal Comparison of consistency models

In order to compare consistency models, we define the concept of models equivalence.

Definition 5 M_1 *and* M_2 *are said* **equivalent** *iff :*

- $C_{M_1} = C_{M_2}$,

- a correct program PRG for M_1 is also correct for M_2,
- if 2 compatible executions $E_{M_1}(PRG)$ and $E_{M_2}(PRG)$ give the same result.

$E_{M_1}(PRG)$ and $E_{M_2}(PRG)$ are said compatible executions if there does not exist (u, v), 2 synchronization operations such that $(u, v) \in E_{M_1}(PRG)$ and $(v, u) \in E_{M_2}(PRG)$.

Theorem 1 *The Atomic Consistency model and the Sequential Consistency model are equivalent.*

Proof : By definition, $C_{AC} = C_{SC}$. Furthermore, AC and SC provide a global order for all read/write operations. Consequently, they are both correct for any program. Finally, for any execution E_{AC}, all accesses are ordered. Yet, E_{SC} only orders operations concerning data accessed several times during the execution. Thus, E_{SC} is included into E_{AC}. For the remaining operations that are in E_{AC} and not in E_{SC}, they concern distinct data. The result is therefor the same whatever the execution order. AC and EC are equivalent. More generally speaking, we refer to them as the *Strong Consistency Model*.

Theorem 2 *The Release Consistency model and the Entry Consistency model are not equivalent.*

Proof : Let us give a counterexample.

Let us consider the program and execution shown here. If for EC, x is not associated to l_1, then PRG is correct for EC (accesses to x are ordered by the synchronization order), and gives the result ($a = 0$, $b = 1$). However, for RC, PRG is also correct, but the result is ($a = 1$, $b = 1$). EC and RC are not equivalent.

Corollary 1 *The Strong Consistency model, the Release Consistency model and the Entry Consistency model are three different consistency models.*

Proof : The strong consistency model does not define any synchronization variables. It is consequently not equivalent to EC or RC.

3. Graphical comparison of the three consistency models

Once we have outlined 3 distinct models, we suggest to compare them with a graphical taxonomy based on user point of view. Considering a correct program for a model M, this taxonomy allows us to answer the following question : *At each moment t of a program execution, if a given process p_i accesses a shared data d, will he read the last value written in the shared memory by another process p_j?* In case of a positive answer, we say that at moment t, process p_i is consistent with the process p_j about the data d.

Then, we represent each model with a 3D visualization of the answer to this question by using various parameters (t, d, p). Hence, each model is represented by a volume made up of the triplets (t, d, p) for which the answer is yes. Basically, (t, d, p) is defined with respect to 3 axes :

When (t) : for times intervals of running application;

What (d) : for shared memory space;

Who (p) : for application processes.

As shown in Fig.1, the axis discretization is based on shared objects pattern accesses.

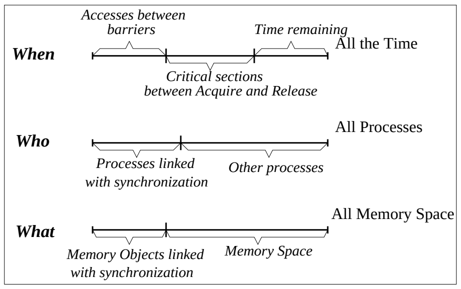
Accesses between barriers
Time remaining
When
All the Time
Critical sections between Acquire and Release
Who
All Processes
Processes linked with synchronization
Other processes
What
All Memory Space
Memory Objects linked with synchronization
Memory Space

FIG. 1 – *When, who, what axis.*

Accordingly, if we normalize the axes, a plain cube means that a process, for all running execution, for all shared data, reads last value written by any other process. The strong consistency model perfectly fits in that definition and graphical representation (Fig. 2). Figures 3 and 4 graphically represent weak consistencies. With Release Consistency, an access performed between Acquire and Release operations is consistent with other processes on all accessible data. In a barrier case, all the processes have the same view of the shared memory. For the remaining execution, there are some conflict risks for any accessible data. The Entry Consistency model gives a slightly different result from the previous one. In effect, if for the barriers, the behavior is the same, it is rather different for an access in a critical section surrounded by an Acquire and Release operation. The process is consistent with the other processes only for the shared data associated to this synchronization.

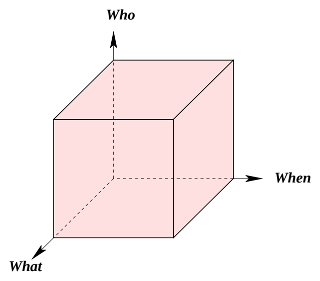

FIG. 2 –. Strong consistency

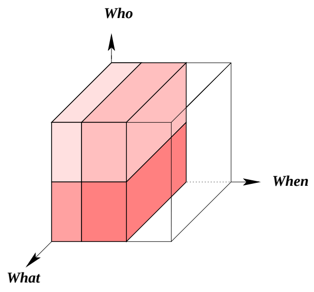

FIG. 3 –. Release consistency

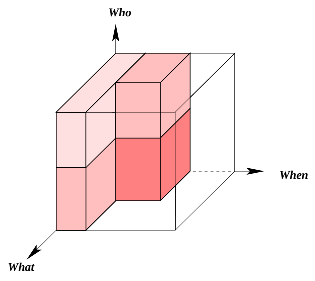

FIG. 4 –. Entry consistency

4. Towards a new consistency model? The Hierarchical Group Consistency (HGC)

Moving from a strong towards a weak consistency consists of the *When* axis decrease. The Entry Consistency model also relaxes the *What* axis. At this stage, only one dimension remains fixed : the processes one. It would be pertinent to make this axis flexible. For an application, all processes do not use all shared data. Moreover, in order to provide scalable solutions, it could be interesting to synchronize with barriers a subset of running processes.

Thus, we propose a new consistency model which groups together processes working on same synchronization variables. Such a model has the graphical representation shown in Figure 5.

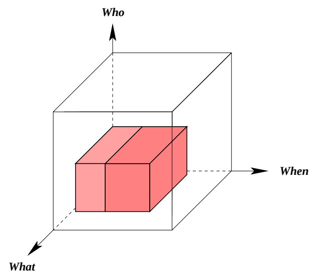

FIG. 5 – *Graphical representation of a new model with 3 flexible axes*

Basically, the HGC model is similar to EC since data are associated to a synchronization variable. Furthermore, processes are also associated to this synchronization variable. Thus, let consider a data modification performed inside a critical section managed by the synchronization variable l. Then, those modifications are forwarded only if they concern data associated to l, and only to processes associated to l. Furthermore, in the HGC model, it is possible to perform some synchronization barrier for only a subset of the processes. Thus, the HGC model can be defined as follows :

Definition 6 *The Hierarchical Group Consistency model is defined by :*

- $C_{HGC} = \{read(x), write(x), Acq(l), Rel(l), Sync(l)\}$
- $(u, v) \in SO_{HGC}(E_{HGC}(PRG))$ *iff* $\exists$ *a synchronization variable* l *to which* u *and* v *are associated such that:*
  u *is performed before* $Rel(l)$ *and* v *is performed after* $Rel(l)$.
  OR u *is performed before* $Sync(l)$ *and* v *is performed after* $Sync(l)$.

Theorem 3 *The Hierarchical Group Consistency model is not equivalent to the Release Consistency model, nor to the Entry Consistency model.*

Proof : HGC introduces a new synchronization operation (the barrier restricted to a synchronization variable) consequently, $C_{HGC} \neq C_{RC}$ and $C_{HGC} \neq C_{EC}$. Thus, those models are not equivalent.

HGC limits coherence management costs of a shared data to some dedicated and explicitly associated processes. Global communications are restricted to processes really needing to have up to date copies of shared data.

By this way, the Hierarchical Group Consistency allows to mix high availability of shared data combined with weak replication [4].

5. Implementation aspect

We implement the Hierarchical Group Consistency inside the DOSMOS framework (Distributed Shared Objects MemOry System). DOSMOS is based on 3 kind of processes : Application Processes (run application code), Memory Processes (manage memory access and consistency inside a group) and Link Processes (manage memory consistency between groups)..

The policy of DOSMOS system is to manage the consistency of an object only within the group of processes that frequently use this object. Thus DOSMOS introduces a hierarchical view of one's application processes by creating groups of processes that frequently share the same object.

We define a group as a pair (G, O) where O is a set of objects and G a set of Application Processes sharing those objects. Each group has a group manager (Link Process : LP) responsible for the inter-group communications. Each group management is independent from the others.

Concerning read and write operations to shared data, we classify the accesses into two categories : the intra-group accesses (the accessing process belongs to object's group) and the inter-group accesses between two distinct groups. Figure 6 shows an example of object accesses using group consistency. In this example, we have 2 sites A and B. In each site, processes share the same object (Y for site A and X for site B). The objects X and Y do not have the same consistency management : Release Consistency for Y and Lazy Release Consistency for X. In this context, we describe three different actions : First, one process of site A writes Y, then this process sends the new value of Y to the other members of the site A. The second action illustrates an inter-group access. One process of site B wants to read the object Y. Since it does not belong to Y's group, it sends its request to its group manager that forwards it to the group manager of site A. This one forwards again the request to the right process in the group that sends the Y value to the requesting process. The last action concerns an acquire of X : since X is managed in Lazy Release Consistency, the acquirer asks the last acquirer for the new value of X.

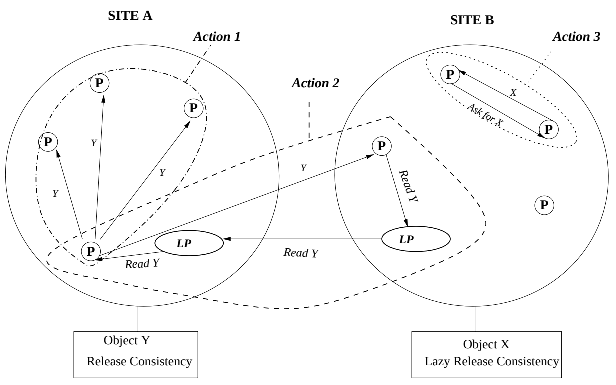

FIG. 6 –. Three actions supporting the group consistency : an intra-write in Release Consistency (action 1), an inter-read (action2), and an acquire in Lazy Release Consistency (action 3)

For an intra-group access, we distinguish two cases :

– If it is the first access for the process to that object, the group manager sends it a copy of this object;

– Else, the process already has a local copy. Then, depending on the consistency implementation of the group, the process will directly work on this copy or not.

For an inter-group access, the accessing process doesn't have a copy and will never have one during the whole execution of the application. It simply asks for the value of this object (for a read) or sends the new value (for a write) to the group manager.

6. Conclusion

This paper presents an original taxonomy of memory consistencies based on the programmer approach. This classification shows the interest of a new consistency relaxing un-useful memory management costs. The Hierarchical Group Consistency proposes to group together processes that frequently access the same shared data. This model allows a gain of performances without denying DSM programmability. The Group Consistency model implemented in DOSMOS is an original approach for improving DSM Systems. It allows the system to be adapted to the environment under which it performs (type of processors, communication network, applications patterns, objects size....) Depending on the environment, the policy concerning replication of objects will not be the same and consequently the group structure will be different. The hierarchical Group Consistency is well suited for large scale systems and could perfectly fit for multi-cluster and Grid applications. This model fits perfectly the requirements of various kind of networks. Depending on the latency, the bandwidth or other criteria, we can design an accurate group structure in order to reach the best compromise between replication and availability suited for the studied network. We are currently dynamic hierachical groups which should easily permit load balancing.

REFERENCES

[1] S. V. Adve, A. L. Cox, S. Dwarkadas, R. Rajamony, and W. Zwaenepoel. A comparison of entry consistency and lazy release consistency implementations. In *Proc. of the 2nd IEEE Symp. on High-Performance Computer Architecture (HPCA-2)*, pages 26–37, February 1996.

[2] S. V. Adve and K. Gharachorloo. Shared memory consistency models: A tutorial. *IEEE Computer*, 29(12):66–76, December 1996.

[3] Brian N. Bershad, Matthew J. Zekauskas, and W. A. Sawdon. The midway distributed shared memory system. In *38th IEEE International Computer Conference (COMPCON Spring'93)*, pages 528–537, February 1993.

[4] Alice Bonhomme and Laurent Lefèvre. How to combine strong availability with weak replication of objects? In *ECOOP98 : 12th Conference on Object Oriented Programming : Workshop on Mobility and Replication*, Bruxelles, Belgium, July 1998.

[5] Lionel Brunie, Laurent Lefèvre, and Olivier Reymann. High performance distributed objects for cluster computing. In *1st IEEE International Workshop on Cluster Computing (IWCC '99)*, pages 229–236, Melbourne, Australia, dec 1999. IEEE Computer Society Press.

[6] John B. Carter, John K. Bennet, and Willy Zwaenepoel. Implementation and performance of MUNIN. *ACM - Operating Systems Review*, 25(5):152–164, 1991.

[7] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *16th Annual Symposium on Computer Architecture*, pages 15–26, May 1989.

[8] L. Higham, J. Kawash, and N. Verwaal. Defining and comparing memory consistency models. In *Proc. of the 10th Int'l Conf. on Parallel and Distributed Computing Systems (PDCS-97)*, October 1997.

[9] W. Hu, W. Shi, and Z. Tang. A framework of memory consistency models. *Journal of Computer Science and Technology*, 13(2), March 1998.

[10]L. Iftode, J. P. Singh, and K. Li. Scope consistency: A bridge between release consistency and entry consistency. In *Proc. of the 8th ACM Annual Symp. on Parallel Algorithms and Architectures (SPAA'96)*, pages 277–287, June 1996.

[11]L. Lamport. How to make a multiprocessor that correctly executes multiprocess programs. *IEEE Trans. on Computers*, C-28(9):690–691, September 1979.

[12]Laurent Lefèvre and Olivier Reymann. Combining low-latency communication protocols with multithreading for high performance dsm systems on clusters. In *8th Euromicro Workshop on Parallel and Distributed Processing*, pages 333–340, Rhodes, Greece, Jan 2000. IEEE Computer Society Press.