A Suite of Formal Definitions for Consistency Criteria in Distributed Shared Memories

Michel Raynal
IRISA, Campus de Beaulieu
35042 Rennes Cedex (France)
raynal@irisa.fr

André Schiper
EPFL, Dept d'Informatique
1015 Lausanne (Switzerland)
schiper@di.epfl.ch

Abstract

A shared memory built on top of a distributed system constitutes a *Distributed Shared Memory* (DSM). If a lot of protocols implementing DSMs in various contexts have been proposed, no set of homogeneous definitions has been given for the many semantics offered by these implementations. This paper provides a suite of such definitions for atomic, sequential, causal, PRAM and a few others consistency criteria. These definitions are based on a unique framework: a parallel computation is defined as a partial order on the set of read and write operations invoked by processes, and a consistency criterion is defined as a constraint on this partial order. Such an approach provides a simple classification of consistency criteria, from the more to the less constrained one. This paper can also be considered as a survey on consistency criteria for DSMs.

Key words: shared memory, distributed system, consistency criteria, partial order.

1 Introduction

Since the end of the eighties, the Distributed Shared Memory abstraction (a shared memory built on top of a distributed system) is receiving more and more attention. One of its very first implementation has been done in the IVY system designed by Li and Hudak [10]. The distributed shared memory abstraction (DSM for short) has many advantages. First, at the application level, DSM frees the programmer from the underlying support as he has to consider only the well known *shared variables programming* paradigm to design a solution to his problem, independently of the system that will run his program (be it a centralized shared memory or a distributed one). Additionally, this facilitates his programming task as a lot of problems (especially related to numerical analysis or image processing) are easier to solve by using the shared variables paradigm than by using the message passing one. Second, at the system level, DSM makes transparent transport of programs, load balancing and process migration.

So, numerous protocols implementing a DSM on top of a distributed memory parallel machine or on top of a distributed system have been proposed. References [15] and [16] survey systems offering a DSM to their users. DSM implementations have common points with multiprocessor caches, networked file systems and distributed databases. Basically, the shared memory is supported by local memories of processors and copies of a data item can simultaneously be present in several local memories. Due to characteristics of the distributed context (asynchronous communications, existence of several copies, etc.), some protocols implementing a shared memory on top of a distributed system offer to users a shared memory whose semantics is lightly different (sometimes in a very subtle way) from the classic semantics associated with a centralized shared memory, namely the *atomic* semantics.

Semantics of a shared memory is expressed by a *consistency criterion*. Such a criterion defines the value returned by every read operation invoked by a process. In nearly all DSMs [15, 12, 16], this consistency criterion is not formally defined and has to be deduced from the protocol implementing the shared memory. This makes study of properties of DSMs difficult and facilitates neither their understanding nor their comparison.

We propose, in this paper, a set of formal definitions for the following consistency criteria: atomic consistency, sequential consistency, causal consistency, PRAM consistency and a few others. These definitions consider a shared memory computation as a partial order on the set of read and write operations issued by processes, and a particular consistency criterion is expressed as a constraint that the partial order has to satisfy[1]. A protocol implementing a DSM with some consistency criterion C has to ensure all computations will satisfy the associated constraint. Such an approach has several advantages. First, as these definitions are independent of particular implementations, they exhibit intrinsic properties associated with consistency criteria; so, this approach follows the abstract data type one by clearly distinguishing the semantics of the "object" offered to users (a shared memory with some semantics) from particular implementations. Second, the set of definitions given in this paper constitutes a hierarchical suite in the following sense: as they all are expressed by using the same formalism, it is possible to order them (from the more to the less constrained); consequently it is easy to see what are the additional constraints required by one consistency criterion with respect to another by comparing their positions within the hierarchy.

The paper is divided into 5 main sections. Section 2 presents the basic shared memory model. Then, Sections 3, 4, 5, and 6 give formal definitions for sequential, atomic, causal and PRAM consistency, respectively. Basic principles of protocols implementing these criteria are also given. The interested reader will consult [18] that offers an in-depth study of the topic addressed in this paper.

2 Shared Memory Model

2.1 Notations

A shared memory system is composed of a finite set of sequential processes $P_1, \ldots, P_n$ that interact via a finite set X of shared objects. Each object $x \in X$ can be accessed by read and write operations. A write into an object defines a new value for the object; a read allows to obtain a value of the object. A write of value v into object x by process P_i is denoted $w_i(x)v$; similarly a read of x by process P_j is denoted $r_j(x)v$ where v is the value returned by the read operation; op will denote either r (read) or w (write). For simplicity, as in [13, 3, 19], we assume all values written into an object x are distinct. Moreover, the parameters of an operation are omitted when they are not important. Each object has an initial value; it is assumed that this value has been assigned by an initial fictitious write operation.

[1] Moreover, it is worth noting that this set of formal definitions is based on very few (and simple) mathematical notions, namely: partial order, linear extension, suborder and legality (of read operations).

2.2 Histories

Histories are introduced to model the execution of shared memory parallel programs. The *local history* (or local computation) $\widehat{h}_i$ of P_i is the sequence of operations issued by P_i. If $op1$ and $op2$ are issued by P_i and $op1$ is issued first, then we say $op1$ precedes $op2$ in P_i's process-order, which is noted $op1 \rightarrow_i op2$. Let h_i denote the set of operations executed by P_i; the local history $\widehat{h}_i$ is the total order $(h_i, \rightarrow_i)$.

An *execution history* (or simply a history, or a computation) $\widehat{H}$ of a shared memory system is a partial order $\widehat{H} = (H, \rightarrow_H)$ such that:

- $H = \bigcup_i h_i$
- $op1 \rightarrow_H op2$ if :
  - *i)* $\exists\ P_i\ : op1 \rightarrow_i op2$ (in that case, $\rightarrow_H$ is called *process-order* relation),
  - or *ii)* $op1 = w_i(x)v$ and $op2 = r_j(x)v$ (in that case $\rightarrow_H$ is called *read-from* relation),
  - or *iii)* $\exists op3\ : op1 \rightarrow_H op3$ and $op3 \rightarrow_H op2$.

Two operations $op1$ and $op2$ are *concurrent* in $\widehat{H}$ if we have neither $op1 \rightarrow_H op2$ nor $op2 \rightarrow_H op1$.

2.3 Legality

A read operation $r(x)v$ is *legal* if: (i) $\exists\ w(x)v\ : w(x)v \rightarrow_H r(x)v$) and (ii) $\nexists\ op(x)u\ : (u \neq v) \wedge (w(x)v \rightarrow_H op(x)u \rightarrow_H r(x)v)$. A history $\widehat{H}$ is legal if all its read operations are legal.

The legality concept is the key notion on which are based our definitions of shared memory consistency criteria. In a legal history no read operation can get an overwritten value. In the following sections, the definition of every consistency criterion follows the same pattern:

- First, according to the consistency criterion considered, one or several histories are defined from the computation $\widehat{H}$,
- Then, $\widehat{H}$ is claimed to satisfy the consistency criterion if and only if this (these) associated history (-ies) is (are) legal.

3 Sequential Consistency

3.1 Definition

Sequential consistency has been proposed by Lamport in 1979 to define a correctness criterion for multiprocessor shared memory systems [9]. A system is sequentially consistent with respect to a multiprocess program, if "*the result of any execution is the same as if (1) the operations of all the processors were executed in some sequential order, and (2) the operations of each individual processor appear in this sequence in the order specified by its program*".

This informal definition states that the execution of a program is sequentially consistent if it could have been produced by executing this program on a monoprocessor system. More formally, we define sequential consistency in the following way.

Definition. Sequential consistency.
A history $\widehat{H} = (H, \rightarrow_H)$ is *sequentially consistent* if it admits a linear extension[2] in which all reads are legal.

As an example let us consider the history $\widehat{H_1}$ (Figure 1). Each process P_i, (i=1,2), has issued three operations on the shared objects x and y. The write operations $w_1(x)0$ and $w_2(x)1$ are concurrent. It is easy to see that $\widehat{H_1}$ is sequentially consistent by building a legal linear extension $\widehat{S}$ including first the operations issued by P_2 and then the ones issued by P_1. It is also easy to see that the history $\widehat{H_2}$ (Figure 2) is not sequentially consistent, as no legal linear extension of $\widehat{H_2}$ can be built.

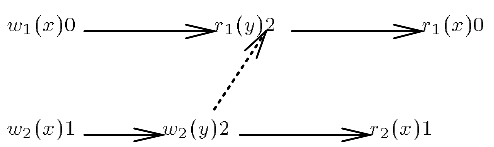

Figure 1: A sequentially consistent history $\widehat{H_1}$

3.2 Protocols

Various cache-based protocols implementing sequential consistency have been proposed in the context of parallel machines [2, 5, 14]. In most of these protocols, every processor local memory contains a copy of the whole shared memory. Read operations are local, while write operations issued by processes are totally ordered.

In [2] and in the *fast read* protocol of [5], this total order is built by an underlying *atomic broadcast* primitive (messages sent with this primitive are delivered in the same order to each procesor [6]). Read operations issued by a process are appropriately scheduled by its processor in order to ensure their legality.

4 Atomic Consistency

4.1 Definition

Atomic consistency is the "oldest" consistency criterion and the one that is the most encountered in distributed systems. It is more constraining than sequential consistency as it adds to sequential consistency the following constraint: any two non-overlapping operations must appear in their real-time order within $\widehat{H}$.

Expressed in the previous model this means that executions of operations can no longer be considered as instantaneous. In order to take into account the real-time occurrence of operations, a *real-time precedence* relation, denoted $\prec_{RT}$, is defined in the following way. Let e_i^s and e_j^t be two operations belonging to H; if e_i^s was terminated before (with respect to physical time) e_j^t began, then we have, by definition: $e_i^s \prec_{RT} e_j^t$. Relation $\prec_{RT}$ is a partial order relation: two operations overlapping in real-time are not ordered.

Definition. Atomic Consistency.
A history $\widehat{H} = (H, \rightarrow_H)$ is *atomically consistent* if it admits a linear extension $\widehat{S} = (H, \rightarrow_S)$ (i) whose all reads are legal (i.e., $\widehat{S}$ is sequentially consistent) and (ii) which is a linear extension of $(H, \rightarrow_{RT})$ (i.e., $e_i^s \prec_{RT} e_j^t \Rightarrow e_i^s \rightarrow_S e_j^t$).

As soon as reads are legal (point *i* of the definition), they return the "last" value of a variable. The fundamental difference between sequential consistency and atomic consistency lies in the meaning of the word "last". In the case of sequential consistency "last" refers to logical time, while it refers to physical time in the case of atomic consistency (point *ii* of the definition).

The interested reader will find in [13, 5] a theory of atomic consistency. In [7], under the name of *linearizability*, atomic consistency theory is generalized to objects.

[2]A linear extension $\widehat{S} = (S, \rightarrow_S)$ of a partial order $\widehat{H} = (H, \rightarrow_H)$ is a topological sort of this partial order, *i.e.*, (i) $S = H$, (ii) $op_1 \rightarrow_H op_2 \Rightarrow op_1 \rightarrow_S op_2$ ($\widehat{S}$ maintains the order of all ordered pairs of $\widehat{H}$) and (iii) $\rightarrow_S$ defines a total order.

4.2 Protocols

The most representative protocol implementing atomic consistency on top of distributed memory parallel machines is the Li-Hudak's one [10]. This protocol uses an invalidation approach. Each data (a page in this protocol) is owned by a process, namely the last process that wrote into it. When a process, wants to read a page for which it has not a copy, it sends a request to the manager of this page that forwards this request to the current owner. When the owner receives such a request, it sends a copy of the page to the requesting process and invalidates its write access right associated with the page. When a process wants to write a page, it sends, through the manager of the corresponding page, a request to the current owner; when receiving such a request, the owner first invalidates all -except his own- copies it has previously disseminated, and then sends its copy to the requesting process. After this, the requesting process is the new owner of the page, and no one else has a copy of the page. These mechanisms ensure atomicity (*i.e.* mutual exclusion) between any couple of read and write operations, and any couple of write operations.

5 Causal Consistency

5.1 Definition

Causal consistency has first been introduced by Ahamad *et al.* in 1991 [3], and then studied by several authors [4, 19]. It defines a consistency criterion strictly weaker than sequential consistency, and allows for a wait-free implementation of read and write operations in a distributed environment (*i.e.* causal consistency allows for cheap read/write operations).

With sequential consistency, all processes agree on a same legal linear extension $\widehat{S}$. The agreement defined by causal consistency is weaker. Given a history $\widehat{H}$, it is not required that two processes P_i and P_j agree on the same ordering for the write operations which are not ordered in $\widehat{H}$. The reads are however required to be legal.

The set of operations that may affect a process P_i are the operations of P_i plus the set of all write operations issued by other processes. Let $\widehat{H_i}$ be the sub-history of $\widehat{H}$ from which all read operations not issued by P_i have been removed[3].

[3] More formally, $\widehat{H_i}$ is the sub-relation of $\widehat{H}$ *induced* by the set of all the writes of H and all the reads issued by P_i.

Definition. Causal consistency.
Let $\widehat{H} = (H, \rightarrow_H)$ be a history. $\widehat{H}$ is *causally consistent* if, for each process P_i, all the read operations of $\widehat{H_i}$ are legal.

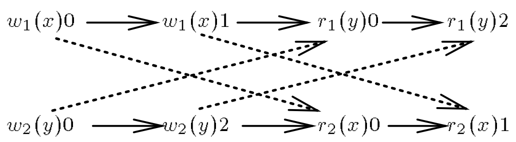

Figure 2: A causally consistent history $\widehat{H_2}$

Said another way, in a causally consistent history, all processes see the same partial order on operations but, as processes are sequential, each of them might see a different legal linear extension of this partial order.

So, in a causally consistent history, no read operation of a process P_i can get a value that, from P_i's point of view, has been overwritten by a more "recent" write. As an example consider history $\widehat{H_2}$ (Figure 2). This history is causally consistent as all its read operations are legal. The history $\widehat{H_3}$ (Figure 3) is not causally consistent as the read operation $r_3(x)1$ issued by P_3 is not legal: $w_1(x)1 \rightarrow_H r_3(x)2 \rightarrow_H r_3(x)1$. Said another way: when P_3 has issued its first read operation on x (namely $r(x)2$), it has got the value 2, and consequently for this process, the value 1 of x has logically been overwritten.

Actually, when considering read and write operations as equivalent to receive and send operations in message passing systems, causal consistency is equivalent, in the shared memory model, to causal ordering [6, 17] for the delivery of messages in the message passing model.

5.2 Protocols

[4] presents an implementation of causal memory and studies programming with such a memory. It is shown that, when executed on a causally consistent memory, concurrent-write free[4] or data-race free[5] parallel programs behave as if the underlying shared memory was sequentially consistent. This is particularly interesting as these programs, intended to be executed on a sequentially consistent memory, remain

[4] A program is *concurrent-write free* if, in all its executions, all its write operations are totally ordered.

[5] A program is *data-race free* if all accesses to each variable follow the readers-writer discipline [1].

correct when executed on a "less synchronized memory".

This approach has been generalized in [19] where a synchronization condition, called MSC, less restrictive than concurrent-write freeness or data-race freeness, is proposed. This condition can be used when executing, on a causally consistent memory, a program designed for a sequentially consistent memory. For the computation to be correct, it is only necessary to add on top of the causally consistent memory, a protocol implementing the MSC condition. Such a condition is interesting as it provides a layered approach to design a sequentially consistent memory: the library can contain (i) a first protocol implementing a causally consistent memory, and (ii) a second one implementing the MSC condition.

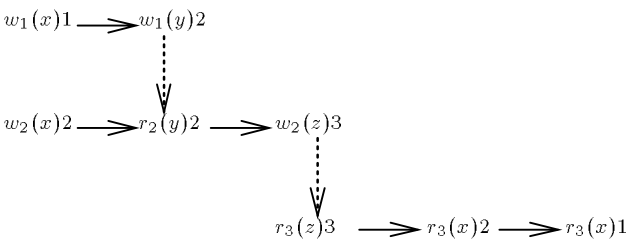

Figure 3: A non causally (but PRAM) consistent history $\widehat{H_3}$

6 PRAM Consistency

PRAM (Pipelined RAM) consistency [11] is a consistency criterion weaker than causal consistency. The difference, in the shared memory model, between PRAM consistency and causal consistency is the same as the one between FIFO ordering and causal ordering for message deliveries in the message passing model [6, 17]. PRAM and FIFO are only concerned by "direct relations" between pairs of "adjacents" processes and do not take into account transitivity due to intermediary processes. More precisely, in a message passing system with FIFO ordering, two messages sent to a same process by two distinct senders can be delivered in any order, even if the send events are causally related [8] (this is not the case with causal ordering: if the send events are causally related, messages must be delivered in their sending order to the destination process). In the same way, in a PRAM consistent shared memory system, two updates of objects by two distinct processes can be know in any order by a third one[6] (this is not the case in a causally consistent shared memory : if $w_k(x)u \rightarrow_H w_j(x)v$, a process P_i reading x can never get v and then u). As an example consider history $\widehat{H_3}$ which is not causally consistent (Figure 3). This history is PRAM consistent: as $w_1(x)1$ and $w_2(x)2$ have been issued by distinct processes, values 1 and 2 of x can be known by P_3 in any order.

Let $\widehat{H}$ be a history and let $\widehat{H'} = (H', \rightarrow_{H'})$ be a history defined from $\widehat{H}$ in the following way ($\widehat{H'}$ differs from $\widehat{H}$ only in point *iii* defining transitivity -see Section 2.2- where $\rightarrow_i$ is used instead of $\rightarrow_H$):

- $H' = H$ (so $H' = \bigcup_i h_i$)
- $op1 \rightarrow_{H'} op2$ if :

  i) $\exists\ P_i\ : op1 \rightarrow_i op2$
  (*process-order* relation),

  or *ii)* $op1 = w_i(x)v$ and $op2 = r_j(x)v$
  (*read-from* relation),

  or *iii)* $\exists op3\ : op1 \rightarrow_i op3$ and $op3 \rightarrow_i op2$.

Let $\widehat{H_i'}$ be the sub-history of $\widehat{H'}$ from which all read operations not issued by P_i have been removed (Figure 4 depicts the sub-history $\widehat{H_3'}$ associated with the computation $\widehat{H_3}$ described in Figure 3). $\widehat{H}$ is PRAM consistent if, for each P_i, all read operations of $\widehat{H_i'}$ are legal. This definition of PRAM consistency shows that its difference, with respect to causal consistency, lies only in the nature of the transitivity considered (point *iii* of their definitions).

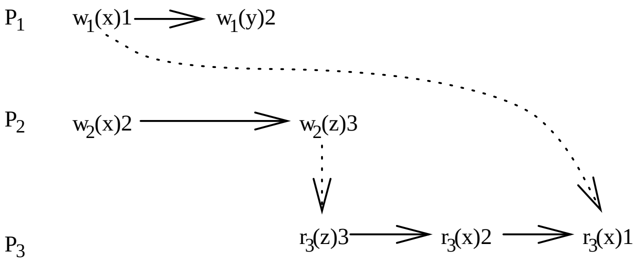

Figure 4: The sub-history $\widehat{H_3'}$

[6] Of course, only one of the updates can be know, if updates overwrite the value previously written by other processes.

7 Conclusion

Numerous protocols implementing distributed shared memory systems have been designed. In most of them, the semantics (consistency criterion) they offer to users is defined only by the description of the protocol and not in an abstract way. This makes it difficult the study of their properties and the appreciation of their differences. In this paper, we provided a suite of formal definitions for the most encountered consistency criteria, namely atomic, sequential, causal and PRAM. These definitions are not bound to particular implementations and are based on a unique framework. This, not only eases their understanding and their comparison, but should facilitate the design of a generic protocol which could be customized to the specific need of each user.

References

[1] S.V. Adve and M.D. Hill. Weak ordering - a new definition. *Proc. 17th Annual ISCA (Int. Symposium on Computer Architecture)*, pp. 2-20, 1990.

[2] Y. Afek, G. Brown, and M. Merritt. Lazy caching. *ACM Transactions on Programming Languages and Systems*, 15(1):182–205, 1993.

[3] M. Ahamad, J.E. Burns, P.W. Hutto, and G. Neiger. Causal memory. in *Proc. 5th Int. Workshop on Distributed Algorithms (WDAG-5)*, pages 9–30. Springer Verlag, LNCS 579, 1991.

[4] M. Ahamad, P.W. Hutto, G. Neiger, J.E. Burns, and P. Kohli. Causal memory: definitions, implementations and programming. *Distributed Computing*, 9:37-49, 1995.

[5] H. Attiya and J.L. Welch. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems*, 12(2):91–122, 1994.

[6] K. Birman and T. Joseph. Reliable communications in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47-76, 1987.

[7] M. Herlihy and J. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, 1990.

[8] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[9] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, C28(9):690–691, 1979.

[10] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4),:321-359, 1989.

[11] R.J. Lipton and J.S. Sandberg. PRAM: a scalable shared memory. Tech. Report CS-TR-180-88, Princeton University, Sept. 1988.

[12] W.G. Levelt, M.F. Kaashoek, H.E. Bal and A.S. Tanenbaum. A comparison of two paradigms for distributed shared memory. *Software Practice and Experience*, 22(11):985-1010, 1992.

[13] J. Misra. Axioms for memory access in asynchronous hardware systems. *ACM Transactions on Programming Languages and Systems*, 8(1):142–153, 1986.

[14] M. Mizuno, M. Raynal, and J.Z. Zhou. Sequential Consistency in Distributed Systems. *Proc. Int. Workshop "Theory and Practice in Dist. Systems"*, Dagstuhl, Germany, Springer-Verlag LNCS 938, (K. Birman, F. Mattern and A. Schiper Eds), pp. 227-241, 1994.

[15] B. Nitzberg and V. Lo. Distributed shared memory: a survey of issues and algorithms. *Computer*, 24(8):52-60, 1991.

[16] J. Protic, M. Tomašević and V. Milutinović. A survey of distributed shared memory systems. *Proc. 28th Annual Hawaii Int. Conf. on System Sciences*, Vol. I (Architecture), pp. 74-84, 1995.

[17] M. Raynal, A. Schiper and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Information Processing Letters*, 39:343–350, 1991.

[18] M. Raynal and A. Schiper. A Suite of Formal Definitions for Consistency Criteria in Distributed Shared Memories. Irisa Research Report 968, 27 pages, Dec. 1995.

[19] M. Raynal and A. Schiper. From causal consistency to sequential consistency in shared memory systems. *Proc. 15th Int. Conf. FST&TCS (Foundations of Software Technology and Theoretical Computer Science)*, Bangalore, India, Sringer-Verlag LNCS Series 1026, (P.S. Thiagarajan Ed.), pp. 180-194, dec. 1995.