J. Parallel Distrib. Comput. 69 (2009) 295–306

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Interconnection of distributed memory models[☆]

Vicent Cholvi [a,*], Ernesto Jiménez [b], Antonio Fernández Anta [c]

[a] Universitat Jaume I, 12071 Castellón, Spain

[b] Universidad Politécnica de Madrid, 28031 Madrid, Spain

[c] LADyR, GSyC, Universidad Rey Juan Carlos, 28933 Móstoles, Spain

ARTICLE INFO

Article history:
Received 21 December 2007
Received in revised form
26 September 2008
Accepted 25 November 2008
Available online 10 December 2008

Keywords:
Distributed shared memory
Memory models
Interconnection systems
Distributed algorithms
Impossibility result
Correctness proofs

ABSTRACT

In this paper, we present a framework to formally describe and study the interconnection of distributed shared memory systems. Using it allows us to classify the consistency models in two groups, depending on whether they are *fast* or not. In the case of non-fast consistency models, we show that they cannot be interconnected in any way. In contrast, in the case of fast consistency models we provide protocols to interconnect some of them.

© 2008 Elsevier Inc. All rights reserved.

1. Introduction

Distributed shared memory (DSM) is a well-known mechanism for interprocess communication in distributed environments [21]. Roughly speaking, it consists in using read and write operations for interprocess communication, thus hiding the particular communication technique employed by the programmers to avoid the need to be involved in the management of messages. However, this can cause problems in systems where several processes independently and simultaneously submit reads and writes, since they can see each other's operations out of order. This problem led to the concept of *consistency models*. A consistency model is a specification of the allowable behavior of memory, and it can be seen as a contract between memory implementation and the program utilizing memory: the memory implementation guarantees that for any input it will produce some output from the set of allowable outputs specified by the consistency model, and the program must be written to work correctly for any output allowed by the consistency model. Depending on the semantics of the memory operations, a number of consistency models has been proposed in the literature (see for instance [21,7,13,11,16]).

In this paper, we study the interconnection of distributed shared memory systems. By this we mean the addition of an *interconnection system* to several existing distributed shared memory systems that implement a given consistency model in order to obtain a single distributed shared memory system that implements the same consistency model. There are two main reasons for interconnecting DSM systems with new protocols instead of using a single protocol for the whole system:

- First, in this way we can interconnect systems that are already running without changing them. They can go on using their protocols at their local level.
- Second, depending on the network topology, it could be more efficient to implement several systems and interconnect them than to have one single large system. An example of this would be a DSM system that has to be implemented on two local area networks connected with a low-speed point-to-point link. If the protocol that is used broadcasts updates, in a single system with many popular protocols there would be a large number of messages crossing the point-to-point link for the same variable update. In this case, it would seem appropriate to implement one system in each of the local area networks, and use an interconnecting protocol via the link to connect the whole system. With the appropriate interconnecting protocol, many fewer messages cross the link for each variable update.

☆ A preliminary version of this paper appeared in the Proceedings of Opodis'03 [E. Jiménez, A. Fernández, V. Cholvi, Decoupled interconnection of distributed memory models, in: OPODIS, 2003, pp. 235–246]. This work was partially supported by the Spanish Ministry of Science and Technology under Grants No. TSI2006-07799, No. TSI2004-02940 and No. TIN2005-09198-C02-01, and by the Comunidad de Madrid under Grant No. S-0505/TIC/0285.

* Corresponding address: Departamento de Lenguajes y Sistemas Informáticos, Universitat Jaume I, Campus de Riu Sec, 12071 Castellón, Spain.

E-mail addresses: vcholvi@lsi.uji.es, vcholvi@uji.es (V. Cholvi).

0743-7315/$ – see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2008.11.008

It is interesting to compare our approach with the concept of locality, defined by Herlihy and Wing [12]. Both approaches have to do with the ability to compose DSM systems. However, locality addresses composability of DSM systems with the same set of processes but disjoint sets of memory objects, while our approach studies the composability of DSM systems with the same set of memory objects but disjoint sets of processes.

A first contribution of this work is the introduction of a framework for the interconnection of memory systems and the formalization of the interactions between the existing memory systems and the interconnection system. Furthermore, we identify the *fastness* of a memory model (a concept that will be defined later in the paper) as the key property that will qualify it to be an interconnectable memory model or not.

In the case of non-fast consistency models, we show that they cannot be interconnected in any way, thus deriving that a number of popular memory models can not be interconnected (e.g., the atomic, safe, regular and sequential models [18], the PCG and PCD consistency models [10,1], the eager release model [9], the lazy release model [17], the entry model [6], the scope model [14], etc.).

In contrast, we show that several fast consistency models can, indeed, be interconnected (namely, the pRAM [20], causal [2], and cache models [10]). However, whereas the cache model can be interconnected without any restriction, we found that the other two memory models can only be interconnected when the subsystems fulfill certain restrictions. In this last situation, we give sufficient conditions and the corresponding interconnecting protocols to do so.

Regarding previous work that has been carried out on the interconnection of distributed shared systems, as far as we know, it has only been studied in [8].[1] Here, we extend the results of that paper in a number of ways. First, we consider consistency models other than the causal one (which was the only one considered in [8]). Second, we provide some impossibility results related with interconnection of consistency models. Third, we use much weaker assumptions on the systems to be interconnected.

The rest of the paper is organized as follows. In Section 2, we introduce the framework for the interconnection of systems. In Section 3, we show the impossibility of interconnection for non-fast consistency models. In Section 4, we study the interconnection of pRAM systems, in Section 5 the interconnection of causal systems, and in Section 6 we show how to interconnect cache systems. In Section 7, we briefly study the performance of the proposed interconnecting protocols. Finally, in Section 8, we present some concluding remarks.

2. System model

We consider *distributed shared memory systems* (or *systems* for short) formed by a collection of *application processes* that interact via a *shared memory* consisting of a set of *variables*. All the interactions between the application processes and the memory are performed through read and write operations (*memory operations*) on variables of the memory.

Each memory operation is applied on a named variable and has an associated value. A write operation of the value v in the variable x, denoted $w(x)v$, stores v in the variable x. A read operation of the value v from the variable x, denoted $r(x)v$, reports to the issuing application process that the variable x holds the value v. To simplify the analysis, we assume that a given value is written at most once in any given variable and that the initial values of the variables are set by using fictitious write operations.

[1] In addition, of course, to the preliminary version of this paper, appeared in OPODIS'03 [15].

Furthermore, we also consider explicit *synchronization operations*. Synchronizations can be used just to import information, as with the acquiring of a lock, or just to export information, as with the release of a lock.

In order to characterize the system model, we specify the components that form it, the *consistency model*, the *system architecture* and the *interconnecting system*.

2.1. The consistency model

Roughly speaking, a *consistency model* (also called *memory model*) is a specification of the allowable behavior of the system's operations. To formally define a consistency model, first we introduce what a system's execution is. An *execution* α of a system S consists of a set of read and write operations, as well as synchronization operations (if any), issued by the application processes that form system S. Such operations must preserve the so called *execution order*. To define this, first we introduce the *process order*.

Definition 1 (*Process Order*). Let p be a process of S and $op, op' \in \alpha$. Then op precedes op' in p's *process order*, denoted $op \prec_p op'$, if op and op' are operations issued by p, and op is issued before op'.

Definition 2 (*Execution Order*). Let $op, op' \in \alpha$. Then op precedes op' in the *execution order*, denoted $op \prec op'$, if any of the following hold:

(1) op and op' are operations from the same process p and $op \prec_p op'$.
(2) $op = w(x)v$ and $op' = r(x)v$.
(3) There is an operation $op'' \in \alpha$ such that $op \prec op'' \prec op'$.

Now, we formally define a *consistency model* as follows:

Definition 3 (*Consistency Model*). A *consistency model* M is a set formed by all executions of type M.

Obviously, for this definition to make sense, it is necessary to define what an execution of type M is in each case. The specification of particular types of executions will be dealt with later in the paper. For such a task, we need to define several related concepts.

Definition 4 (*View*). Let $\prec^o$ be an order defined on the operations of execution α, and let $\alpha' \subseteq \alpha$. A *view* β of α' preserving $\prec^o$ is a sequence formed by all operations of α' such that this sequence preserves the order $\prec^o$.

Note that if $\prec^o$ does not define a total order on α', then there can be several views of α'. We use $op \stackrel{\beta}{\rightarrow} op'$ to denote that op precedes op' in a view β. We will omit the view when it is clear from the context. We will also use $\alpha \rightarrow \alpha'$, where α and α' are sets of operations, to denote that all the operations in α precede all the operations in α'.

Definition 5 (*Legal View*). Let $\prec^o$ be an order defined on the operations of execution α, and let $\alpha' \subseteq \alpha$. A view β of α' preserving $\prec^o$ is *legal* if for each read operation $r(x)v \in \alpha'$,

(a) there is a write operation $w(x)v \in \alpha'$ such that $w(x)v \stackrel{\beta}{\rightarrow} r(x)v$, and
(b) there is no write operation $w(x)u \in \alpha'$ such that $w(x)v \stackrel{\beta}{\rightarrow} w(x)u \stackrel{\beta}{\rightarrow} r(x)v$.

2.2. The system architecture

From a physical point of view, we consider distributed systems as consisting of a set of *nodes* and *a network* that provides communication among them. The essence of this model has

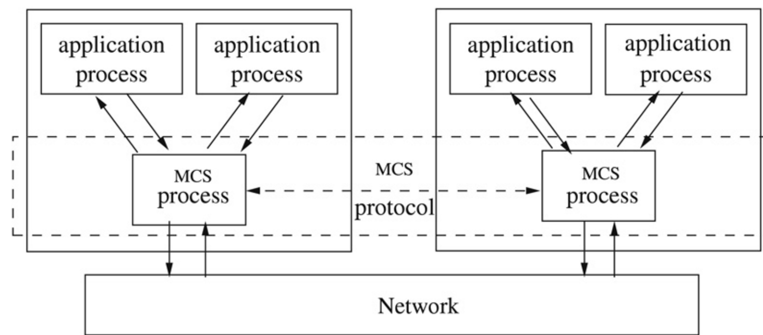

Fig. 1. System architecture.

been taken from [4]. The application processes of the system are actually executed in the nodes of the distributed system. We assume that the shared memory abstraction is implemented by a *memory consistency system* (*MCS*). The *MCS* is composed of *MCS-processes* that use local memory at the various nodes and cooperate following a distributed algorithm, or *MCS-protocol*, to provide the application processes with the impression of having a shared memory. The *MCS-processes* are executed at the nodes of the distributed system and exchange information as specified by the *MCS-protocol*. They use the communication network to interact if they are in different nodes. Each *MCS-process* can serve several application processes, but an application process is assigned to only one local *MCS-process*. For each application process p we use $mcs(p)$ to denote its *MCS-process*. An application process and its *MCS-process* have to be in the same node, as stated by the following assumption.

Assumption 1. Let p be an application process. Process p and $mcs(p)$ are in the same node.

An application process sequentially issues read/write/synchronization operations on the shared variables by sending (read/write/synchronization) calls to its *MCS-process*. After sending a call, the application process blocks until it receives the corresponding response from its *MCS-process*, which ends the operation. We assume an asynchronous model. This means that there is no bound on the amount of time instructions and message transmissions take. We do not assume synchronized clocks among processes. We also assume that no system component (processes, nodes, and networks) fails. Fig. 1 shows an example of the system architecture described above.

Regarding the consistency model implemented by a system (i.e., by its *MCS*), we follow the same approach taken when defining a consistency model:

Definition 6 (*System*). A *system* is of type M if all its executions are of type M.

Furthermore, we consider systems in which at least the last write operation on every variable must be eventually visible in every process of the system. This is a very natural property which is preserved by every system that we have found in the literature. In our terminology, it means that their *MCSs* must satisfy the following property:

Liveness property. Consider any execution α of system S. If there is only one process writing on variable x and its last operation on x was $w(x)u$, then eventually the response to any read call on x issued by any application process will contain the value u.

2.3. The Interconnection system

Interconnecting several systems involves making them to behave as though they were one single system. Using the terminology defined above, this actually means interconnecting several *MCS*s.

In our model, the load of such an interconnection will fall on an *interconnection system* (*IS*). An *IS* is a set of processes (*IS-processes*) that execute some distributed algorithm or protocol (*IS-protocol*). For simplicity in the *IS* design, we consider the existence of one *IS*-process for each *MCS* to be interconnected.

The *IS-process* of each system is an application process and, hence, it has an *MCS-process* that by Assumption 1 is in its same node. The *IS-process* uses the *MCS-process* to read and write on the shared memory of the local system. In particular, the only way a value written by an application process in some system can be read by an application process in another system is if the *IS-process* of the latter system writes it. *IS-processes* exchange information with each other (as specified by the *IS-protocol*) by using a reliable FIFO communication network. Note that, after the interconnection, the overall system has a *global MCS* formed by the *MCS*s of the original systems plus the *IS* that interconnects them. Fig. 2 presents an example of an *IS* interconnecting two systems.

Definition 7. We will say that a consistency model can be *interconnected* if for any collection of systems implementing this consistency model there is an IS-protocol that interconnects them.

In the rest of the paper we will use N to denote the number of systems to be interconnected. The systems to be interconnected will be denoted by $S^0, \ldots, S^{N-1}$, and the resulting interconnected system by S^T. The *IS-process* for each system S^k (where $k \in \{0, \ldots, N-1\}$) is denoted by isp^k. It is worth remarking that isp^k is part of the system S^k. We consider that the set of processes of S^T includes all the processes in $S^0, \ldots, S^{N-1}$ except $isp^0, \ldots, isp^{N-1}$ (since they are only used to interconnect the systems $S^0, \ldots, S^{N-1}$).

Regarding how the *IS*s operate, we note that it is necessary to guarantee that any given *IS-process* be eventually aware of the writes taking place at the *MCS-processes* that it manages, so that it could exchange such information with other *IS-processes*. This functionality can be implemented in a number of ways.

(1) Within the *IS-process*: for instance, it can be implemented by making the *IS-process* check for any updated variable, by periodically reading the whole memory. In this case, the *IS-process* will behave as a regular application process and no additional assumption is made on the *MCS-processes*.
(2) Within the *MCS-processes*: in this case the *MCS-processes* have to communicate explicitly any update to the *IS-process*. Whereas such an approach could be more efficient than the previous one, it requires that the *MCS-processes* be able to perform such a task.
(3) By using a combination of both.

In order to maintain it as general as possible, in this paper we only assume that there is an *interface* (between the *MCS* and the *IS*) that provides the above mentioned functionality,

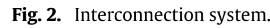

Fig. 2. Interconnection system.

without considering how it is implemented.[2] However, in order to "decouple" as much as possible the original systems and the interconnecting protocol, this interface does not allow the *IS* to contact the *MCS* (except to read or write variables). In particular, the *IS* cannot block the *MCS* (as was done in [8]). More formally, the interface guarantees the following assumption:

Assumption 2. When any *MCS-process* updates its local memory (as a result of a write operation issued by an application process), the *IS-process* will be asynchronously notified about these events (i.e., about the updated variable, the written value and the application process). Other than this, there is no *MCS*-initiated interaction between *MCS* and *IS* processes.

3. Fast vs non-fast consistency models

In this section, we show that only systems implementing *fast* consistency models can be interconnected. Formally, we define a fast consistency model as follows:

Definition 8. We say that a consistency model is *fast* if there is an *MCS-protocol* that implements it, such that memory operations only require local computations before returning control, even in systems with several nodes.

Since there are several examples of popular fast and non-fast models, this implies that the property of being fast classifies the set of memory models in a non-trivial way. The following observation will be useful to prove some subsequent results.

Observation 1. *Every IS-protocol that interconnects $N > 2$ systems can be used to interconnect 2 systems. Furthermore, every IS-protocol that interconnects 2 systems can be used to interconnect $N > 2$ systems.*

Proof. For the first part, let us consider that there is an *IS-protocol* that interconnects $N > 2$ systems through a set of N *IS-processes*. If we only have two systems, one of the two *IS-processes* can simulate $N - 2$ empty systems and their *IS-processes*. Then, we have an interconnected system of two systems.

For the second part, we use induction on i to show that i systems can be interconnected for any $i \geq 2$. For $i = 2$ the claim is trivially true. Now, assume that we can obtain a system S' by interconnecting the systems $S^0, S^1, \ldots, S^{i-2}$. The result of the interconnection is a single system. Then, the *IS-protocol* can be used to properly interconnect S' and S^{i-1}.

In what follows, we consider the interconnection of only two systems, and use this observation to generalize our results to several systems. Now, we prove that non-fast memory models cannot, in general, be interconnected.

Theorem 1. *There is no IS that guarantees the interconnection of systems implementing non-fast memory models.*

Proof. We show the result by contradiction. Assume that there is a non-fast memory model M that can be interconnected. From Observation 1, we can consider the interconnection of two systems. Therefore, let us assume there is an *IS* I that interconnects two systems implementing M. Let us first take a distributed system with two nodes. In each node we implement a system with one *MCS-process*, at least one application process, and the corresponding *IS-process*. By Assumption 1, the *MCS-process* and the application processes (the *IS-process* included) are in the same node. Then, in each of these two single-node systems each memory operation only requires local computations. Now, we use I to interconnect these two systems into a unique system implementing M. By Assumption 2, I cannot block the *MCS-processes*. Then, every memory operation in the resulting system still requires only local computations, which contradicts the fact that M is not fast.

As a consequence of this theorem, we derive that a number of popular memory models cannot be interconnected. In [4] it is shown that the sequential consistency model is not fast. Hence, it cannot be interconnected, and the same happens with the atomic consistency model and its derivations, i.e., the safe and regular memory models [19]. Similarly, Attiya and Friedman [5] have shown that the processor consistency models PCG and PCD [10, 1] are not fast, and consequently cannot be interconnected. Finally et al. [5] also proved that any algorithm for the mutual exclusion problem using fast operations must be cooperative. This implies that any synchronization operation that guarantees mutual exclusion must be non-fast. Therefore, any synchronized memory model that provides exclusive access cannot be interconnected. As a result, we have that memory models such as the eager release [9], the lazy release [17], the entry [6] or the scope [14] cannot be interconnected.

On the other hand, there is a number of consistency models that are fast and for which Theorem 1 does not apply. In the following sections we show that some of the most popular fast memory models (namely, the pRAM [20], the causal [2] and the cache [10]) can indeed be interconnected, although, in some cases, in a constrained fashion.

We will assume that these fast systems control the replicas by *propagating* the new values to update the replicas. This assumption does not significantly restrict the domain of application of our results, since all current implementations of fast models we are aware of have been obtained by using propagation.

[2] In addition to the above mentioned approaches, a local copy of the shared memory could be stored in a *protected* zone of the physical memory, so that any modification generates an interruption that informs the *IS-process* without using the *MCS-processes* However, here we do not consider this case, since it requires some "help" from the operating system.