Available online at www.sciencedirect.com
SCIENCE DIRECT
J. Parallel Distrib. Comput. 64 (2004) 498–506
Journal of Parallel and Distributed Computing
http://www.elsevier.com/locate/jpdc

On the interconnection of causal memory systems

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

[a] Universidad Rey Juan Carlos, 28933 Móstoles, Spain
[b] Universidad Politécnica de Madrid, 28031 Madrid, Spain
[c] Universitat Jaume I, 12071 Castellón, Spain

Received 6 July 2001; revised 28 January 2003; accepted 11 March 2004

Abstract

In this paper, we look at the interconnection of propagation-based causal Distributed shared memory (DSM) systems. We present extremely simple protocols to interconnect two such systems (possibly implemented with different algorithms), that only require the existence of a bidirectional reliable FIFO channel connecting one process from each system. We show that the resulting DSM system is also causal. This result can be used to interconnect any number of DSM propagation-based causal systems, by interconnecting them in pairs with a tree topology.

Keywords: Distributed systems; Distributed shared memory; Causal consistency; Interconnection of systems

1. Introduction

Shared memory (reading and writing of shared variables) is a well-known mechanism for interprocess communication in concurrent programs. However, while the semantic of read and write operations in sequential programs is clear, the situation is different when there can be concurrent accesses to shared variables. This is more evident if the shared memory is not centralized but distributed among a number of processors, i.e. we have distributed shared memory (DSM). There has been a number of proposals and implementations of DSM systems providing different semantics, or *consistency models* [5,9].

The *causal* memory model has attracted the attention of a number of researchers because it is considered to be powerful enough to allow relatively easy programming but, at the same time, it allows inexpensive implementations. As a consequence, a number of protocols implementing the causal memory model have been proposed in the literature (see for instance [2,6,8]). Most protocols implementing causal memory, in order to increase concurrency, support *replication* of data. With replication, there are copies (replicas) of the same variables in the local memories of several processes of the system, which allows these processes to use the variables simultaneously. However, in order to guarantee the consistency of the shared memory, the system must control the replicas when the variables are updated. That control can be done by either *invalidating* outdated replicas or by *propagating* the new variable values to update the replicas.

1.1. Our results

In this paper, we explore the interconnection of causal DSM systems implemented with replication and propagation. In particular, we introduce simple protocols for interconnecting causal memory systems, possibly implemented with different propagation-based protocols. The interconnection protocols proposed only require the existence of reliable FIFO channels connecting processes from each system. We show that the resulting system is also causal.

We first study the connection of two propagation–based causal system. We assume the existence in each system of a special process, called *interconnecting system* (IS)-*process*, which will be in charge of actually running the interconnection protocol. Those IS-processes are connected by a reliable FIFO channel, which will be used to exchange the data required by the

*Corresponding author.
E-mail addresses: afernandez@acm.org (A. Fernández), ernes@eui.upm.es (E. Jiménez), vcholvi@lsi.uji.es (V. Cholvi).

0743-7315/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2004.03.007

interconnection. We present protocols that can be run by the IS-processes in order to connect both systems (we call them IS-protocols). Basically, these protocols propagate the variable updates from one system to the other. We then show that the system obtained by connecting two systems with the IS-processes, running the proposed IS-protocols, is causal. An interesting property of our IS-protocols is that the reliable FIFO channel used does not need to be available all the time. If the channel is not available during some period of time, the variable updates can be queued up to be propagated at a later time. This makes the protocol practical even with dial-up connections.

Next, we show that the interconnection scheme for two systems can be used to interconnect a larger number of systems. Hence, we show that several propagation-implemented causal systems can be interconnected with our IS-protocols to obtain a large causal system. To do so, we interconnect the original systems in pairs avoiding the creation of cycles, which results in a tree interconnection topology.

Note that the sequential memory model, which is maybe the most widely known, is in fact causal. Hence, these results also apply to it, i.e., two sequential systems (implemented, for instance, with the local read algorithm proposed by Attiya and Welch [3]) can be interconnected so that the overall resulting system is causal. Clearly, the system obtained most possibly will not be sequential. There are other stronger-than-causal memory models (e.g., the atomic memory model) to which this may apply as well.

There are mainly two reasons for interconnecting causal 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 keep 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. An example of this would be a causal system that has to be implemented on two local area networks connected with a low-speed point-to-point link. If the causal protocol used broadcasts updates, in a single system there could 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 IS-protocol via the link to connect the whole system. Then, only one message crosses the link for each variable update.

1.2. Related work

We do not know of any previous work on interconnection of DSM systems. However, in the context of message passing systems, Rodrigues and Verissimo [10], Adly and Nagi [1], and Baldoni et al. [4] have proposed architectures and protocols to implement large causally ordered message-passing systems by interconnecting smaller causally ordered message-passing systems. Since a causal DSM system can be easily implemented on a causally ordered message-passing system [8], a large causal DSM system could be obtained by implementing smaller causally ordered message-passing systems, interconnecting them as any of the above papers proposes, and then implementing the causal DSM system on the resulting large causally ordered message-passing system. However, if the processes are already grouped into causal DSM systems, as we assume in this work, the above approach does not seem to be practical anymore, since it would imply to build causal message-passing systems on top of causal DSM systems to build a larger causal DSM system. In general, the goal of these hierarchical causal ordering papers is to improve performance in large-scale environments, while ours is to interconnect existing systems.

The rest of the paper is organized as follows. In Section 2, we introduce the basic framework and provide a formal definition of causal DSM system. In Section 3, we introduce the IS-protocols we propose for interconnecting two causal DSM systems. In Section 4, we show that the union of two causal DSM systems with the IS-protocols proposed is causal. In Section 5, we show that our approach can be used to connect more than two causal DSM systems. Finally, in Section 6, we briefly study the performance of the causal DSM system obtained by the interconnection of several causal DSM systems.

2. Definitions

A *DSM system* (*system* for short) consists of a set of *application processes* that interact via a set of variables. These variables constitute the *shared memory*. All the process interactions with the memory are done through read and write operations (*memory operations*) on variables of the memory. Each memory operation acts on a named variable and has an associated value. A write operation by process i (within the system S^q), denoted $w_i^q(x)v$, stores the value v in the variable x. Similarly, a read operation, denoted $r_i^q(x)v$, reports to process i (within the system S^q) that v is stored in the variable x. 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 write operations.

An *execution* of a system S^q is the concurrent execution of all its application processes. From the execution of a process, all we care about are the memory operations it issues. A *computation* α^q of a system S^q consists of a sequence of read and write operations observed in some execution of S^q. We denote α_i^q the computation obtained by removing from α^q all read operations from processes other than i. Similarly, we denote with $\stackrel{\alpha^q}{\rightarrow}$ the order in which the operations in α^q happen. For operations of the same process i, $\stackrel{\alpha^q}{\rightarrow}$ reflects the order in which these operations have been executed by i. We now introduce the legal computation concept.

Definition 1 (Legal Computation). A computation α^q is legal if $\forall op = r_i^q(x)v(\exists op' = w_j^q(x)v : op' \stackrel{\alpha^q}{\rightarrow} op$ and $\nexists op'' = w_k^q(x)u : op' \stackrel{\alpha^q}{\rightarrow} op'' \stackrel{\alpha^q}{\rightarrow} op)$.

In order to capture "causality" (in the sense of [7]), we need to define the causal order.

Definition 2 (Causal Order). Let op and op' be two operations in a computation α^q. Then $op \prec^{\alpha^q} op'$ if some of the following holds:

1. op and op' are operations from the same process and $op \stackrel{\alpha^q}{\rightarrow} op'$.
2. $op = w_i^q(x)v$ and $op' = r_j^q(x)v$.

From this, we define the *causal order* $\prec\prec^{\alpha^q}$ as the transitive closure of the order $\prec^{\alpha^q}$. By using the causal order and the legal computation concept, we now define both causal view, causal computation, and causal system.

Definition 3. Let α^q be a computation of system S^q. We say that β_i^q is a *causal view* of α_i^q if it is a permutation of α_i^q, it is legal, and it preserves the causal order $\prec\prec^{\alpha^q}$.

Definition 4. We say that a computation α^q of system S^q is *causal* if, for each process i, the computation α_i^q has a causal view.

Definition 5. We say that the system S^q is *causal* if all its computations are causal.

We use an architecture of DSM system proposed by Attiya and Welch [6], in which the DSM is implemented by a *memory consistency system* (*MCS*). The MCS is formed by *MCS-processes* that cooperate following a distributed protocol (*MCS-protocol*) to provide the application processes with the impression of having a shared memory. Each application process is attached to one MCS-process. An application process issues read or write operations on the shared variables by sending (read or write) *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. A write call carries the value to be written and the variable in which to write it. The response to a write call is the explicit acknowledgment of the call by the MCS-process. A read call carries the variable to be read, while its response contains the value of the variable as seen by the MCS-process.

Hence, interconnecting a set of DSM systems is, in fact, interconnecting their respective MCS. We do so with an (IS). After the interconnection, the overall system has a *global MCS* formed by the MCSs of the original systems plus the IS that interconnects them. An IS is basically a set of processes (*IS-processes*), one in each system to be interconnected (one IS-process could belong to several systems), that execute some distributed protocol (*IS-protocol*) and are connected by reliable message passing FIFO channels. An IS-process is a special kind of application process. It is attached to an exclusive MCS-process, can issue read and write operations, and exchanges information with other IS-processes.

We only consider the interconnection of causal systems in which the MCS-process of the IS-process is implemented with *replication* and *propagation*. We impose that this MCS-process has a local replica of each of the variables of the shared memory. Every write operation issued by an application process is eventually propagated to this MCS-process, which updates the corresponding local replica. We assume that the interface between each IS-process and its MCS-process is extended with two *upcalls*, sent by the MCS-process to the IS-process when local replicas of variables are updated. The update of a replica due to a write operation issued by the IS-process does not generate any upcall. Otherwise, the MCS-process sends a *pre_update*(x) upcall immediately before its replica of variable x is updated with some value v and a *post_update*(x, v) upcall immediately after. (As we will see, the *pre_update*(x) upcall is not always necessary. We assume that it can be disabled by the IS-process.) When the MCS-process sends an upcall, it must block until the IS-process replies with a *response*.

In our IS-protocol, the MCS-processes of the IS-processes, when they update the replica of a variable x with a value v must operate in a way such that (a) the value s held by the local replica of x when the corresponding *pre_update*(x) upcall is sent, is not modified until the update with v is done, and this value v is not modified until the response to the *post_update*(x, v) upcall is received. Furthermore, our IS-protocol also needs to be able to issue read operations while processing these upcalls. Then, (b) these read operations must be guaranteed to finish, and (c) they must return the value s or v when issued in the processing of the *pre_update*(x) or *post_update*(x, v) upcalls, respectively. The conditions (a) and (c) are needed for the correctness of the IS-protocol, while condition (b) prevents deadlocks.

3. The IS-protocols for interconnecting causal systems

In this section, we introduce two IS-protocols for interconnecting two causal systems S^0 and S^1 so that the resulting system, S^T, is also causal. For generality, we use S^k to denote any of these systems S^0 and S^1, and $S^{\bar{k}}$ (where $\bar{k}$ implicitly means $1-k$) to denote the other. As we described in the previous section, we have one IS-process for each system S^k, denoted isp^k. Such a process is in charge of executing the IS-protocol of the corresponding system.

In essence, the IS-protocols we propose simply propagate the write operations issued in one system to the other by means of the IS-processes. We first make sure that write operations that are causally ordered in system S^k are propagated to system $S^{\bar{k}}$ in that order. Otherwise, these operations would have a different causal order in $S^{\bar{k}}$ and it could never be guaranteed that the overall system is causal. However, this condition is not enough, since it does not guarantee that causal dependencies are preserved by the propagations. For instance, suppose $w_i^k(x)v$ is issued in S^k and that after its propagation by $isp^{\bar{k}}$ some process j in $S^{\bar{k}}$ issues $r_j^{\bar{k}}(x)v$ and $w_j^{\bar{k}}(x)u$, in this order. Then, without violating the causality of S^k, some process l in S^k could issue first $r_l^k(x)u$ and then $r_l^k(x)v$, which violates the causality of the system S^T. To prevent this, we force the IS-processes to issue read operations on every value propagated among systems, which creates causal relations between write operations propagated in both directions.

We present two IS-protocols which can interact with each other. They only differ in the code executed by the IS-process, but their interface between IS-processes is the same. Each IS-process will choose which one to use depending on which class of causal MCS-protocol its system is running. We consider two classes of causal MCS-protocols, depending on whether they guarantee the following property.

Property 1 (Causal Updating). *In any computation α^k of system S^k, if application processes i and j issue the write operations $w_i^k(x)v$ and $w_j^k(y)u$, and $w_i^k(x)v \prec\prec^{\alpha^k} w_j^k(y)u$, then the MCS-process of isp^k will update its replica of x with the value v before updating its replica of y with the value u.*

We will first consider a system implemented with a causal MCS-protocol that satisfies the Causal Updating Property. (All the causal protocols we have found in the literature fall within this class.) In this case, each IS-process contains two tasks, $Propagate_{out}^k$ and $Propagate_{in}^k$. While $Propagate_{out}^k$ deals with transferring write operations issued in S^k to the system $S^{\bar{k}}$, $Propagate_{in}^k$ deals with applying within S^k the write operations transferred from the system $S^{\bar{k}}$ by $Propagate_{out}^{\bar{k}}$. To work properly, $Propagate_{out}^k$ has to guarantee that two causally ordered write operations are transferred to $S^{\bar{k}}$ following the causal order. To do so, we use a reliable FIFO ordered communication channel. Similarly, $Propagate_{in}^k$ must apply the write operations transferred from $S^{\bar{k}}$ in exactly the same order they are received.

Fig. 1 shows the code of tasks $Propagate_{in}^k$ and $Propagate_{out}^k$. Task $Propagate_{out}^k$ is activated with parameters x and v when the *post_update*(x,v) upcall is received (i.e., immediately after the local replica of variable x is updated with value v). As a result, it reads the value v from x and sends the pair $\langle x,v\rangle$ to the $isp^{\bar{k}}$ process. Recall that the updates due to write operations issued by isp^k do not generate upcalls. Then, a pair received from $isp^{\bar{k}}$ cannot be sent back. On its turn, task $Propagate_{in}^k$ is activated with parameters x and v whenever the pair $\langle x,v\rangle$ is received from the process $isp^{\bar{k}}$. As a result, its MCS-process performs a causal write operation, thus causally propagating the value v to all the replicas of variable x within S^k. Fig. 3 shows the interaction of these tasks with their environment (the MCS-process and the process $isp^{\bar{k}}$). In this first IS-protocol isp^k disables the MCS-process *pre_update* upcalls, since it does not need them.

Let us consider now the more general case in which the Causal Updating Property is not necessarily satisfied by the causal MCS-protocol of the system S^k. In this case, the IS-protocol has a new task $Pre\_Propagate_{out}^k(x)$ (see Fig. 2), which is executed immediately before the local replica of variable x in the MCS-process of isp^k is updated with a new value v. This task issues a read operation on x, $r_{isp^k}^k(x)s$, which reads the value s previously held in x. This task enforces that two causally ordered write operations are propagated by $Propagate_{out}^k$ following the causal order even if the MCS-protocol does not enforce that the replicas are updated in that order (as shown in Lemma 1 below). In Fig. 3 are

$Propagate_{out}^k(x,v)$ :: task which is activated when the $post\_update(x,v)$ upcall is received from the MCS-process.
begin
  $r_{isp^k}^k(x)v$
  send $\langle x,v\rangle$ to $isp^{\bar{k}}$
  send *response* to the MCS-process
end

$Propagate_{in}^k(x,v)$ :: task which is activated when a pair $\langle x,v\rangle$ is received from $isp^{\bar{k}}$.
begin
  $w_{isp^k}^k(x)v$
end

Fig. 1. The IS-protocol for systems that satisfy the Causal Updating Property.

$Pre\_Propagate_{out}^k(x)$ :: task which is activated when the $pre\_update(x)$ upcall is received from the MCS-process.
begin
  $r_{isp^k}^k(x)s$
  send *response* to the MCS-process
end

Fig. 2. Third task, used in systems that do not satisfy the Causal Updating Property.

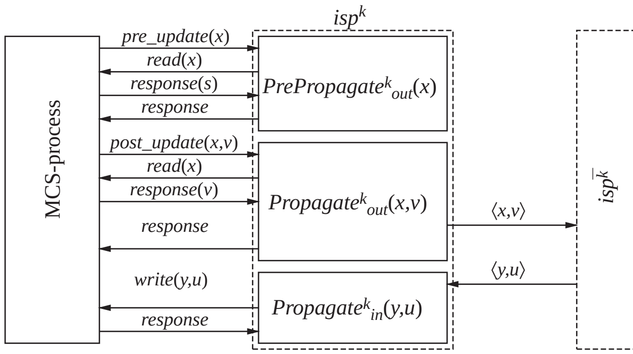

Fig. 3. Task scheme of the IS-protocols.

pictured the tasks of this IS-protocol and their interactions with the MCS-process and the $isp^{\bar{k}}$ process.

The following lemma presents the fundamental property satisfied by both IS-protocols.

Lemma 1. *In any computation α^k of system S^k (where $k \in \{0, 1\}$), if application processes i and j issue the write operations $w_i^k(x)v$ and $w_j^k(y)u$, and $w_i^k(x)v \prec\prec^{\alpha^k} w_j^k(y)u$, then $Propagate^k_{out}$ will send the pairs $\langle x, v \rangle$ and $\langle y, u \rangle$ to system $S^{\bar{k}}$ in this order.*

Proof. All we need to show is that the local replicas of x and y in the MCS-process of isp^k are updated in that order, since $Propagate^k_{out}$ sends the pairs in the same order the updates are applied. The claim trivially follows if the causal MCS-protocol used by S^k satisfies the Causal Updating Property and the IS-protocol of Fig. 1 is used, since the local replica of x is updated with v before the local replica of y is updated with u.

Now, we show by contradiction that, if we use the second IS-protocol with the new task $Pre\_Propagate^k_{out}$, then the local replicas of x and y in the MCS-process of isp^k are also updated in that order, even if the MCS-protocol does not satisfy the Causal Updating Property. Let us assume, then, by way of contradiction, that the local replica of y is updated with value u before the local replica of x is updated with value v in computation α^k. Then, if we remove from α^k all the read operations not issued by isp^k, and since the system S^k is causal, the resulting computation $\alpha^k_{isp^k}$ must have a causal view $\beta^k_{isp^k}$. From the description of the second IS-protocol and our assumption, isp^k has issued the following operations, in this order: $r^k_{isp^k}(y)t$, $r^k_{isp^k}(y)u$, $r^k_{isp^k}(x)s$, and $r^k_{isp^k}(x)v$, where t and s are the previous values of y and x, respectively (see Fig. 3). Hence, from the first condition of the definition of $\prec^{\alpha^k}$, $r^k_{isp^k}(y)u \prec^{\alpha^k} r^k_{isp^k}(x)s \prec^{\alpha^k} r^k_{isp^k}(x)v$. We have that $w_i^k(x)v \prec\prec^{\alpha^k} w_j^k(y)u$ from the statement of the lemma. Finally, from the second condition of the definition of $\prec^{\alpha^k}$, $w_j^k(y)u \prec^{\alpha^k} r^k_{isp^k}(y)u$. Since $\beta^k_{isp^k}$ must preserve the order $\prec\prec^{\alpha^k}$, the above operations on x must appear in $\beta^k_{isp^k}$ in the order $w_i^k(x)v \xrightarrow{\beta^k_{isp^k}} r^k_{isp^k}(x)s \xrightarrow{\beta^k_{isp^k}} r^k_{isp^k}(x)v$. Let us consider now the operation $w_l^k(x)s$ that writes s in x. There are three possible cases, either there is no such operation in $\beta^k_{isp^k}$, $w_i^k(x)v \xrightarrow{\beta^k_{isp^k}} w_l^k(x)s$, or $w_l^k(x)s \xrightarrow{\beta^k_{isp^k}} w_i^k(x)v$. In either case, the legality of $\beta^k_{isp^k}$ is violated and α^k cannot be causal, which is a contradiction. Hence, the local replica of x must be updated before that of y. $\square$

4. The interconnection of two systems is causal

In this section we show that the system S^T, obtained by connecting two systems S^0 and S^1 using our IS-protocols, is causal. We consider that the set of processes of S^T includes all the processes in S^0 and S^1 except isp^0 and isp^1 (they are only used to interconnect the systems S^0 and S^1).

In what follows, α^T will denote a computation of S^T observed when executing all the processes of both systems S^0 and S^1, interconnected through the IS-processes running our IS-protocols. Similarly, α^k will denote the computation of S^k observed in the same execution. Note that α^k and α^T have in common all the operations issued by processes in S^k. Furthermore, write operation $w_i^{\bar{k}}(x)v$ in α^T issued by some processes i in $S^{\bar{k}}$ appears in α^k as the write operation $w^k_{isp^k}(x)v$ issued by the process isp^k in S^k. This is so because *every* write operation issued by isp^k in α^k is, from our IS-protocols, just the propagation of a write operation issued by a process of $S^{\bar{k}}$. As we defined, α_i^T (resp. α_i^k) is the computation obtained from α^T (resp. α^k) by removing the read operations not issued by the process i. (For simplicity, we assume that processes have unique identifiers in S^T, and hence α_i^T is properly defined.)

4.1. Auxiliary lemmas

The first set of lemmas (from Lemmas 2 to 6) that follow show that if two operations of α^T are causally ordered, their corresponding operations in α^k are also causally ordered. By corresponding operation we mean the same operation if it was issued in S^k, or its propagation if it was a write operation issued in $S^{\bar{k}}$. For these lemmas we need the following definition:

Definition 6. Let op and op' be two operations in α^T such that $op \prec\prec^{\alpha^T} op'$. A *causal sequence* between op and op' is a sequence of operations $op^1, op^2, \ldots, op^m$ such that $op^1 = op$, $op^m = op'$, and $op^c \prec^{\alpha^T} op^{c+1}$ for $1 \leqslant c < m$.

Note that at least one causal sequence always exists between op and op' if $op \prec\prec^{\alpha^T} op'$. A causal sequence Seq between op and op' can be divided in n subsequences $subSeq_1$, $subSeq_2$, $\dots, subSeq_n$, such that all the operations in subsequence $subSeq_d$, $1 \geqslant d \geqslant n$, belong to the same system S^k and the operations in consecutive subsequences belong to different systems. We use $subSeq_d^k$ to express that all the operations of the dth subsequence belong to system S^k, for $1 \geqslant d \geqslant n$.

We use $first(subSeq_d)$ and $last(subSeq_d)$ to denote the first and last operation of the subsequence $subSeq_d$, respectively. Note that, in two consecutive subsequences $subSeq_d^k$ and $subSeq_{d+1}^{\bar{k}}$ of a given sequence, $last(subSeq_d^k) = w_j^k(x)v$ and $first(subSeq_{d+1}^{\bar{k}}) = r_l^{\bar{k}}(x)v$, i.e. the first operation of the later subsequence reads the value written by the last operation of the former subsequence.

Lemma 2. *Let op and op′ be two operations in* α^T *such that* $op \prec\prec^{\alpha^T} op'$. *If there is a causal sequence between op and op′ with one single subsequence* $subSeq_1^k$, *then* $op \prec\prec^{\alpha^k} op'$.

Proof. The claim follows if we show that, for any two consecutive operations op^c and op^{c+1} of $subSeq_1^k$, $op^c \prec^{\alpha^k} op^{c+1}$. Since $op^c \prec^{\alpha^T} op^{c+1}$, we must be in one of two cases (from Definition 2): (1) $op^c \xrightarrow{\alpha^T} op^{c+1}$ and both operations are issued by the same process, or (2) $op^c = w_j^k(x)v$ and $op^{c+1} = r_l^k(x)v$ (where j and l are two processes in S^k). Hence, from the respective cases of Definition 2, $op^c \prec^{\alpha^k} op^{c+1}$. □

Lemma 3. *Let op and op′ be two operations in* α^T *issued by system* S^k *such that* $op \prec\prec^{\alpha^T} op'$. *Then* $op \prec\prec^{\alpha^k} op'$.

Proof. Let Seq be a causal sequence between op and op'. We use induction on the number of subsequences of Seq to show the result. Note that this number has to be odd. In the base case, the sequence Seq has only one subsequence $subSeq_1^k$. Hence, from Lemma 2, $op = first(subSeq_1^k) \prec\prec^{\alpha^k} op' = last(subSeq_1^k)$.

Assume the claim is true for sequences with d subsequences. We show it also holds if Seq has $d+2$ subsequences. By induction hypothesis, we have that $op = first(subSeq_1^k) \prec\prec^{\alpha^k} last(subSeq_d^k)$. Note that $last(subSeq_d^k) = w_j^k(x)v$ is propagated to system $S^{\bar{k}}$ by process isp^k. Before doing so, isp^k issues the operation $r_{isp^k}^k(x)v$ (see task $Propagate_{out}^k$ of Fig. 1). Later on, isp^k propagates $last(subSeq_{d+1}^{\bar{k}}) = w_i^{\bar{k}}(y)u$ as $w_{isp^k}^k(y)u$ (see task $Propagate_{in}^k$ in Fig. 1). Then, from the definition of causal order, $w_j^k(x)v \prec\prec^{\alpha^k} r_{isp^k}^k(x)v \prec\prec^{\alpha^k} w_{isp^k}^k(y)u$ (see Fig. 4). From Lemma 2 we have that $first(subSeq_{d+2}^k) = r_s^k(y)u \prec\prec^{\alpha^k} op' = last(subSeq_{d+2}^k)$. Also, $w_{isp^k}^k(y)u \prec\prec^{\alpha^k} first(subSeq_{d+2}^k) = r_s^k(y)u$. Hence, by transitivity, $op = first(subSeq_1^k) \prec\prec^{\alpha^k} op' = last(subSeq_{d+2}^k)$. □

Let op be a write operation issued in $S^{\bar{k}}$. Let us denote by $prop(op)$ the write operation issued by isp^k as a result of propagating op to S^k.

Lemma 4. *Let op and op′ be two write operations in* α^T *issued by system* $S^{\bar{k}}$. *If* $op \prec\prec^{\alpha^T} op'$, *then* $prop(op) \prec\prec^{\alpha^k} prop(op')$.

Proof. From Lemma 3, $op \prec\prec^{\alpha^{\bar{k}}} op'$. Then, the result follows from Lemma 1, the fact that the channel connecting $isp^{\bar{k}}$ to isp^k is reliable and FIFO, and the implementation of task $Propagate_{in}^k$ (see Fig. 1). □

Lemma 5. *Let op and op′ be two operations in* α^T *issued, respectively, by systems* $S^{\bar{k}}$ *and* S^k, *such that* $op = w_i^{\bar{k}}(x)v \prec\prec^{\alpha^T} op'$. *Then* $prop(op) \prec\prec^{\alpha^k} op'$.

Proof. Let Seq be a causal sequence between op and op'. Let us assume $last(subSeq_1^{\bar{k}}) = w_j^{\bar{k}}(y)u$ and $first(subSeq_2^k) = r_l^k(y)u$. From Lemma 4, $prop(op) \prec\prec^{\alpha^k} prop(last(subSeq_1^{\bar{k}})) = prop(w_j^{\bar{k}}(y)u) = w_{isp^k}^k(y)u$. From Lemma 3 we have that $first(subSeq_2^k) = r_l^k(y)u \prec\prec^{\alpha^k} op'$. From the definition of causal order $w_{isp^k}^k(y)u \prec\prec^{\alpha^k} r_l^k(y)u$. Hence, from transitivity, $prop(op) \prec\prec^{\alpha^k} op'$. □

Lemma 6. *Let op and op′ be two operations in* α^T *issued, respectively, by systems* S^k *and* $S^{\bar{k}}$, *such that* $op \prec\prec^{\alpha^T} op' = w_i^{\bar{k}}(x)v$. *Then* $op \prec\prec^{\alpha^k} prop(op')$.

Proof. Let Seq be a causal sequence between op and op' with m subsequences. Let us assume $last(subSeq_{m-1}^k) = w_j^k(y)u$ and $first(subSeq_m^{\bar{k}}) = r_l^{\bar{k}}(y)u$. From Lemma 3, $op \prec\prec^{\alpha^k} last(subSeq_{m-1}^k) = w_j^k(y)u$. From the implementation of task $Propagate_{out}^k$ (see Fig. 1) the value u is read from y by isp^k before propagating it. Hence, from the definition of causal order, $w_j^k(y)u \prec\prec^{\alpha^k} r_{isp^k}^k(y)u$. Since $r_l^{\bar{k}}(y)u$ has to be executed after the propagation of $w_j^k(y)u$, so has to be op'. Then, $prop(op') = w_{isp^k}^k(x)v$ is executed after $r_{isp^k}^k(y)u$, and $r_{isp^k}^k(y)u \prec\prec^{\alpha^k} prop(op') = w_{isp^k}^k(x)v$ (see Fig. 5). Hence, from transitivity, $op \prec\prec^{\alpha^k} prop(op')$. □

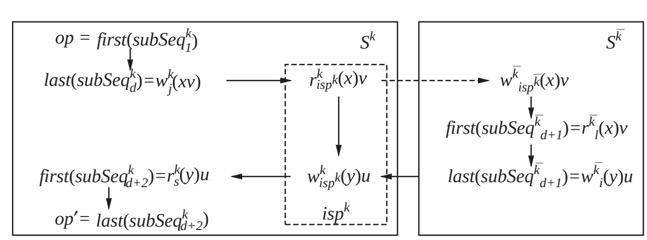

Fig. 4. Precedences for the proof of Lemma 3. Solid arrows represent causal precedences and dashed arrows represent temporal precedences.

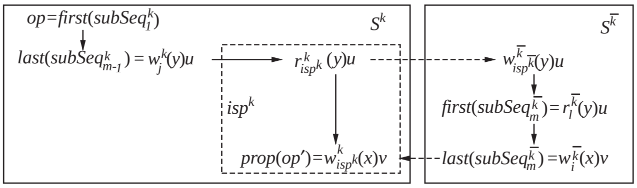

Fig. 5. Precedences for the proof of Lemma 6. Solid arrows represent causal precedences and dashed arrows represent temporal precedences.

4.2. Proof of correctness

Since S^k is a causal system, α^k has to be causal. Therefore, any α_i^k (see Definition 3) has at least one causal view. Let β_i^k be one causal view of α_i^k. Like in α^k, every write operation of the process isp^k in β_i^k is the propagation of a write operation issued by a process of $S^{\bar{k}}$. Let us denote by $orig(op)$ the original write operation propagated as write operation op by process isp^k. From β_i^k, we derive a sequence γ_i^T which we will show is a causal view of α_i^T.

Definition 7. γ_i^T is the sequence obtained by replacing in β_i^k every write operation op from isp^k by the write operation $orig(op)$.

Lemma 7. *γ_i^T is a permutation of the operations in α_i^T.*

Proof. Note that α_i^T contains all the write operations of α^T and the read operations of process i in system S^k. On the other hand, α_i^k contains all the write operations in α^T of processes in S^k, all the read operations of process i in system S^k, and the propagation by isp^k of all the write operations in α^T of processes in system $S^{\bar{k}}$. Then, the difference in their respective sets of operations is that, for each operation op issued by isp^k in α_i^k, α_i^T contains the original operation $orig(op)$.

Since β_i^k is a permutation of α_i^k by definition of causal view, both have the same operations. γ_i^T is obtained from β_i^k by replacing each op issued by isp^k by $orig(op)$. Hence, the set of operations in γ_i^T is the same as that of α_i^T. □

Lemma 8. *γ_i^T preserves the causal order $\prec\prec^{\alpha^T}$.*

Proof. Let us assume, by way of contradiction, that γ_i^T does not preserve the order $\prec\prec^{\alpha^T}$. Hence, there must be at least two operations op and op' such that $op \prec\prec^{\alpha^T} op'$ but op' precedes op in γ_i^T. Let us consider four possible cases.

Case 1: op and op' have been issued by processes of S^k. Then, from Lemma 3, we have that $op \prec\prec^{\alpha^k} op'$. Now note that since op' precedes op in γ_i^T, op' also precedes op in β_i^k, by definition of γ_i^T. Then, β_i^k does not preserve the order $\prec\prec^{\alpha^k}$. Since β_i^k is a causal view of α_i^k, we have a contradiction.

Case 2: op and op' have been issued by processes of $S^{\bar{k}}$. Since both operations are in γ_i^T, which only contains read operations from process i of system S^k, both must be write operations. Let op and op' be propagated as operations $prop(op)$ and $prop(op')$, respectively, issued by process isp^k. From Lemma 4, we have that $prop(op) \prec\prec^{\alpha^k} prop(op')$. Observe now that, by definition, operation $prop(op)$ in β_i^k is replaced by op and operation $prop(op')$ is replaced by op' to obtain γ_i^T. Then $prop(op')$ precedes $prop(op)$ in β_i^k, and hence β_i^k does not preserve the order $\prec\prec^{\alpha^k}$. Since β_i^k is a causal view of α_i^k we have a contradiction.

Case 3: op has been issued by a process of $S^{\bar{k}}$ and op' has been issued by a process of S^k. Note that op must be a write operation, since γ_i^T only contains read operations from process i of system S^k. Operation op is propagated from $S^{\bar{k}}$ to S^k as an operation $prop(op)$ issued by process isp^k. From Lemma 5, $prop(op) \prec\prec^{\alpha^k} op'$. Observe now that, by definition, operation $prop(op)$ in β_i^k is replaced by op to obtain γ_i^T. Then op' must precede $prop(op)$ in β_i^k, and hence β_i^k does not preserve the order $\prec\prec^{\alpha^k}$. Since β_i^k is a causal view of α_i^k we have a contradiction.

Case 4: op has been issued by a process of S^k and op' has been issued by a process of $S^{\bar{k}}$. Note that op' must be a write operation, since γ_i^T only contains read operations from process i of system S^k. Operation op' is propagated from $S^{\bar{k}}$ to S^k as an operation $prop(op')$ issued by process isp^k. From Lemma 6, $op \prec\prec^{\alpha^k} prop(op')$. Observe now that, by definition, operation $prop(op')$ in β_i^k is replaced by op' to obtain γ_i^T. Then $prop(op')$ must precede op in β_i^k, and hence β_i^k does not preserve the order $\prec\prec^{\alpha^k}$. Since β_i^k is a causal view of α_i^k we have a contradiction. □

Lemma 9. *γ_i^T is legal.*

Proof. By definition of causal view, β_i^k is legal. Also by definition, γ_i^T is obtained by replacing in β_i^k every write operation op from isp^k by the write operation $orig(op)$, where both op and $orig(op)$ write the same value in the same variable. Therefore, γ_i^T is legal. □

Theorem 1. *The system S^T is causal.*

Proof. Let α^T be a computation of S^T and let α_i^T be obtained from α^T. From Lemma 7, γ_i^T, as defined in Definition 7, is a permutation of the operations in α_i^T. Also, from Lemma 8, γ_i^T preserves the causal order $\prec\prec^{\alpha^T}$. Finally, from Lemma 9, γ_i^T is legal. Hence, from Definition 3, γ_i^T is a causal view of α_i^T. Since this holds for each process $i \neq isp^k$ of system S^k, for $k \in \{0, 1\}$, we have that α^T is a causal computation. Hence any computation α^T of S^T is causal, and S^T is a causal system. □

5. Generalization to several systems

The following corollary shows that our IS-protocols can be used to interconnect any number of systems to obtain a large causal system.

Corollary 1. *n propagation-based causal systems, $S^0, S^1, \ldots, S^{n-1}$, can be interconnected with our IS-protocols to obtain a system S^T causal.*

Proof. Observe that the system obtained by interconnecting two systems with our IS-protocols is a propagation-based system. We use induction on n to show the result. For $n = 1$ the claim is trivially true. Then, if we have a propagation-based causal system S' by interconnecting the systems $S^0, S^1, \ldots, S^{n-2}$, then we can interconnect S' and S^{n-1} to obtain the propagation-based causal system S^T, from Theorem 1 and the above observation. □

Note that the system S^T is obtained by connecting the original systems in pairs without forming cycles. Hence, the final interconnection topology is a tree.

6. Performance

We compare here the performance of a system obtained using our IS-protocols with the performance of a system that directly uses a causal MCS-protocol connecting all the processes. We assume that the same MCS-protocol is used in the global DSM system of reference and in each of the systems interconnected with our IS-protocols. We also assume that this protocol only implements causal consistency (and not a stronger model).

First, observe that our IS-protocols should not affect the *response time* a process observes when issuing a memory operation, since its MCS-process is not affected by the interconnection. Regarding the *network traffic*, we assume that the MCS-protocol used generates $x - 1$ messages for each write operation in a system with x MCS-processes and no message for a read operation. Then, in a global DSM system with n MCS-processes each write operation generates $n - 1$ messages. With our interconnection protocols $n + 1$ messages are generated for two systems, since we add two MCS-processes (one for each IS-process), and one message will be sent from one IS-process to the other. Generalizing these results for m systems, the number of messages for the interconnected system becomes $n + m - 1$. However, observe that if we have two systems, each one with $n/2$ processes and in different networks, in the global DSM system $n/2$ messages have to cross from one network to the other for each write operation, which can generate a bottleneck. With our protocol only one message has to cross. Note that this bottleneck problem may get worse as the number of networks increases. Finally, we consider the *latency*, which is the time until a value written is visible in any other process. For simplicity, we will discard here local computation times at the IS-processes and possible delays introduced by the conditions at the MCS-processes of the IS-processes. Then, if we have m systems, a system running the basic causal protocol has latency l, the delay of a message between two IS-processes is d, and we interconnect the systems in a star fashion, the worst case latency is $3l + 2d$.

Acknowledgments

This work has been partially supported by the Spanish Ministerio de Ciencia y Tecnología under grants TEL99-0582 and TIC2001-1586-C03-01. We want to thank Sergio Arévalo and Francisco Ballesteros for proposing us the "consistency islands" problem. We also want to thank the anonymous referees for their useful comments.

References

[1] N. Adly, M. Nagi, Maintaining causal order in large scale distributed systems using a logical hierarchy, in: Proceedings of the 12th IASTED International Conference on Applied Informatics, 1995.
[2] M. Ahamad, G. Neiger, J. Burns, P. Kohli, P. Hutto, Causal memory: definitions, implementation and programming, Distrib. Comput. 9 (1) (1995) 37–49.
[3] H. Attiya, J. Welch, Sequential consistency versus linearizability, ACM Trans. Comput. Systems 12 (2) (1994) 91–122.
[4] R. Baldoni, R. Beraldi, R. Friedman, R. van Renesse, The hierarchical daisy architecture for causal delivery, Distrib. Systems Eng. J. 6. (1999) 71–81.
[5] V. Cholvi, J. Bernabéu, Relationships between memory models, Info. Process. Lett. 90 (2) (2004) 53–58.
[6] E. Jiménez, A. Fernández, V. Cholvi, A parametrized algorithm that implements sequential, causal, and cache memory consistency, in: Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-Based Processing (Euro PDP 2002), IEEE Computer Society Press, Canary Islands, Spain, 2002.
[7] L. Lamport, Time, clocks and the ordering of events in a distributed system, Comm. ACM 21 (7) (1991) 558–565.
[8] M. Raynal, M. Ahamad, Exploiting write semantics in implementing partially replicated causal objects, in: Proceedings of the Sixth EUROMICRO Conference on Parallel and Distributed Computing, 1998, pp. 157–163.
[9] M. Raynal, M. Mizuno, How to find his way in the jungle of consistency criteria for distributed shared memories (or how to escape from minos' labyrinth), in: Proceedings of the IEEE International Conference on Future Trends of Distributed Computing Systems, Lisboa, Portugal, 1993.

[10] L. Rodrigues, P. Verissimo, Causal separators and topological timestamping: an approach to support causal multicast in large-scale systems, in: Proceedings of the 15th International Conference on Distributed Systems, 1995.

Antonio Fernández is an associate professor at the Universidad Rey Juan Carlos in Madrid, Spain, since 1998. Previously, he was on the faculty of the Universidad Politécnica de Madrid. He graduated in Computer Science from the Universidad Politécnica de Madrid in 1991, and got a Ph.D. in Computer Science from the University of Southwestern Louisiana in 1994. His research interests include data communications, computer networks, parallel and distributed processing, algorithms, and discrete and applied mathematics.

Ernesto Jiménez graduated in Computer Science from the Universidad Politécnica de Madrid (Spain) and got a Ph.D. in Computer Science from the University Rey Juan Carlos (Spain) in 2004. He is currently an associate professor at the Universidad Politécnica de Madrid.

Vicent Cholvi graduated in Physics from the Universitat de València (Spain) and received his doctorate in Computer Science in 1994 from the Polytechnic University of Valencia (Spain). In 1995, he joined the Jaume I University in Castelló (Spain) where he is currently an associate professor. His interests are in distributed and communication systems.