Accepted Manuscript

Interconnection of distributed memory models

Vicent Cholvi, Ernesto Jiménez, Antonio Fernández Anta

Please cite this article as: V. Cholvi, E. Jiménez, A. Fernández Anta, Interconnection of distributed memory models, *Journal of Parallel and Distributed Computing* (2008), doi:10.1016/j.jpdc.2008.11.008

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*

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.

Key words: Distributed shared memory, memory models, interconnection systems, distributed algorithms, impossibility result, correctness proofs

1 Introduction

Distributed shared memory (DSM) is a well-known mechanism for interprocess communication in distributed environments [2]. 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 have been proposed in the literature (see for instance [2–6]).

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.

It is interesting to compare our approach with the concept of locality, defined by Herlihy and Wing [7]. 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

2

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 [8], the PCG and PCD consistency models [9, 10], the eager release model [11], the lazy release model [12], the entry model [13], the scope model [14], etc.).

In contrast, we show that several fast consistency models can, indeed, be interconnected (namely, the pRAM [15], causal [16], and cache models [9]). 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 [17] [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 [17]). 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.

---

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

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.

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

4

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

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 been taken from [18]. 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
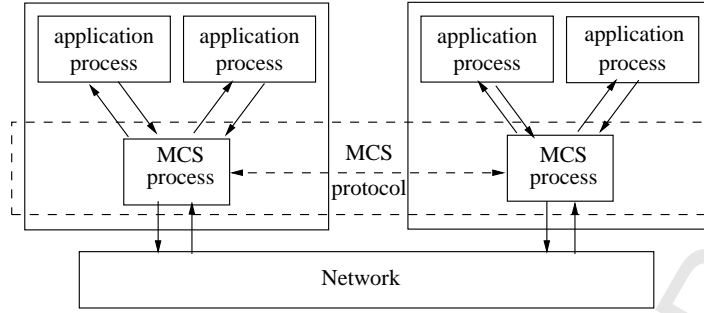
5

Fig. 1. System architecture.

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 *MCSs*.

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
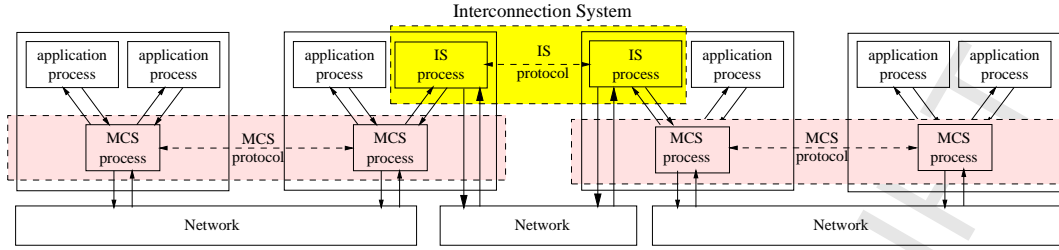
6

Fig. 2. Interconnection System.

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. Figure 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 $S^0, \cdots, S^{N-1}$, and the resulting interconnected system by S^T. The *IS-process* for each system S^k (where $k \in \{0, \cdots, N-1\}$) 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 $S^0, \cdots, S^{N-1}$ except $isp^0, \cdots, isp^{N-1}$ (since they are only used to interconnect the systems $S^0, \cdots, S^{N-1}$).

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-*

7

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, 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 [17]). 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*

---

[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 don't consider this case, since it requires some "help" from the operating system.

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, ..., 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 [18] 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 [20] have shown that the processor consistency models PCG and PCD [9, 10] are not

9

fast, and consequently cannot be interconnected. Finally, Attiya and Friedman [20] 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 [11], the lazy release [12], the entry [13] or the scope [14] cannot be interconnected.

On the other hand, there are 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 [15], the causal [16] and the cache [9]) 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.

4   Interconnection of pRAM Systems

In this section, we study the interconnection of pRAM systems [15]. In this model, every process performs all its operations locally and transmits updated values to the other processes along FIFO channels. These updates are later performed asynchronously at the remote processes. Formally, we define a pRAM system as follows:

Definition 9 (pRAM System) *A system S is pRAM if for every execution α and every process p there is a legal view β_p of α_p preserving $\prec_q$ for all q (where α_p denotes the subset of operations obtained by removing from execution α all read operations issued by processes other than p).*

Following, we show that, in general, the interconnection of pRAM systems is not possible. That is, there is no *IS* that interconnects *every* pair of pRAM systems. The proof is based on the fact that some pRAM systems may not be FIFO ordered (the formal definition of FIFO ordered system is provided below). Indeed, as we pointed at the end of Section 2, it is necessary to have a functionality that guarantees that any given *IS-process* be eventually aware of the writes taking place at the *MCS-processes* that it manages. If such a functionality is implemented within the *IS-process* (i.e., by making the *IS-process* periodically read the whole memory), it is easy to prove that pRAM systems, in general, cannot be interconnected. Basically, the idea is that when

the *IS-process* finds two variables that have been modified since the last time it read them, it cannot know in which order they were written, and may propagate them to the other systems in an incorrect order. Hence, such a functionality must be implemented within the *MCS-processes* (i.e., by making them to communicate explicitly any update to the *IS-process*, as described in Assumption 2). Depending on how this update is done, it could happen that local replicas are updated in a different order than the write operations were issued[3]. Then, the communication of the updates between the *MCS-processes* and the *IS-process* (which is performed without using read and write operations but using explicit messages) could be out of order.

In what follows in the rest of the paper, we extend our notation and use both subscripts and superscripts to respectively denote the process that performs the operations and the system where such a process is located.

Theorem 2 *There is no IS that guarantees pRAM interconnection for every pair of pRAM systems.*

PROOF. The proof is based on the fact that when some process p in S^k issues several write operations, it may update the corresponding variables in its local memory in a different order from p's process order.

Let us assume, by way of contradiction, that there is a system S^T which is the result of interconnecting two pRAM systems S^0 and S^1 through some interconnection system I. From Definition 9, we know that for every execution α^T there is a legal view β_p^T of α_p^T, for all p, preserving $\prec_q$, for all q.

Assume that we have an execution α^0 with the following sequence of write operations issued by process p of S^0: $w_p^0(x)s \prec_p w_p^0(y)l$. We know, from the Liveness Property (see Section 2.2), that there is a time t after which any read operation on x and y issued by any process in S^1 returns s and l, respectively. We now assume that after this time t the process p issues the write operations $w_p^0(x)u$ and $w_p^0(y)v$. We know, following the definition of interface's behavior, that when any *MCS-process* updates its local memory, the interface informs isp^0 about these events. Then, I can take one of the following actions:

- *Case 1: isp^1* issues $w_{isp^1}^1(x)u$ and $w_{isp^1}^1(y)v$, in this order, in S^1. In this case, if $w_p^0(x)u$ and $w_p^0(y)v$ were issued by process p in the order $w_p^0(y)v \prec_p w_p^0(x)u$ and some process q of S^1 issues the read operations $r_q^1(x)u \prec_q r_q^1(y)l$ (which is possible if the pRAM systems are not FIFO ordered), then it is impossible to form a legal view β_q^T preserving $\prec_p$. Hence, we reach a contradiction.

---

[3] An example of this behavior is the causal (and hence pRAM) algorithm of [6], in which the update of several variables are batched into a single message and applied in arbitrary order, in mutual exclusion with all read operations.

---

1   Task $Propagate_{out}^k$ :: executed upon notification from the interface to isp^k that the variable x has been updated in $mcs(isp^k)$ to value v due to a write operation issued by p

2   begin

3   if $p \neq isp^k$ then

4   send $\langle x, v \rangle$ to isp^l, $l \neq k$

5   end

1   Task $Propagate_{in}^k$ :: executed upon reception of $\langle x, v \rangle$ from isp^l, $l \neq k$

2   begin

3      $w(x)v$

4   end

Fig. 3. The pRAM *IS-protocol* for each isp^k, $k \in \{0, 1\}$.

- *Case 2:* isp^1 issues $w_{isp^1}^1(y)v$ and $w_{isp^1}^1(x)u$, in this order, in S^1. In this case, if $w_p^0(x)u$ and $w_p^0(y)v$ were issued by process p in the order $w_p^0(x)u \prec_p w_p^0(y)v$ and some process q of S^1 issues the read operations $r_q^1(y)v \prec_q r_q^1(x)s$ (which is possible if the pRAM systems are not FIFO ordered), then it is impossible to form a legal view β_q^T preserving $\prec_p$. Hence, we reach a contradiction.
- *Case 3:* isp^1 does not issue $w_{isp^1}^1(y)v$ or $w_{isp^1}^1(x)u$ in S^1. From the Liveness Property, this case is not possible.

$\square$

Despite the previous result, we have found that for certain types of pRAM systems, which we call *FIFO ordered*, it is still possible to do this.

FIFO Ordered Systems. We say a system is *FIFO ordered* if for each process p in S^k, if p issues two write operations $w(x)v \prec_p w(y)u$, then the $mcs(isp^k)$ process updates its local replica of x with the value v before updating its local replica of y with the value u.

In Figure 3, we present an *IS-protocol* that can be used to interconnect pRAM systems that are FIFO ordered. It consists of two concurrent tasks, $Propagate_{out}^k$ and $Propagate_{in}^k$. The first task, $Propagate_{out}^k$, deals with transferring write operations issued in S^k to S^l, $l \neq k$. It is activated upon notification from the interface to isp^k that the variable x has been updated to value v due to a write operation issued by the application process p. Then, $Propagate_{out}^k$ sends the pair $\langle x, v \rangle$ to isp^l, $l \neq k$. We avoid re-propagating write operations received from other systems by checking that the write operation was not issued in S^k by isp^k. The second task, $Propagate_{in}^k$, deals with applying within S^k the write operations transferred from the systems S^l, $l \neq k$. It is activated whenever a pair $\langle x, v \rangle$ is received from process isp^l, $l \neq k$. As a result, the isp^k process performs a write operation $w(x)v$, thus propagating the value v to all the replicas of variable x within S^k. We note

12

that the FIFO ordered property do not undergo substantial constraints with respect to systems that are not FIFO ordered. Theorem 3 states formally the guarantees provided by the above mentioned interconnection protocol.

Theorem 3 *The system S^T obtained by connecting N FIFO ordered pRAM systems $S^0, \cdots, S^{N-1}$ using the pRAM IS-protocol in Fig. 3 is pRAM.*

PROOF. See Appendix A.1. □

5 Interconnection of Causal Systems

In this section, we study the interconnection of causal systems [16]. In the causal model, in addition to the conditions of the pRAM executions, read operations are forced to return the value written by the lastest causally ordered operation (i.e., read operations preserve the execution order in Definition 2). Formally, we define a causal system as follows:

Definition 10 (Causal System) *A system S is causal if for every execution α and every process p there is a legal view β_p of α_p preserving $\prec$.*

As in the case of pRAM systems, here we consider systems that are implemented by using *propagation*. First of all, we have that since the pRAM model is strictly weaker than the causal model [3, 16], the result of impossibility in Section 4 is also applicable to causal systems.

Corollary 1 (from Theorem 2) *There is no IS that guarantees causal interconnection for every pair of causal systems.*

In spite of this result, in the previous section we presented an *IS-protocol* for interconnecting pRAM systems that are FIFO ordered. Thus, a question that naturally arises is whether it is possible to interconnect causal systems that are also FIFO ordered. However, here we show that this result does not apply to causal systems. That is, there is no *IS* that interconnects every pair of causal systems, even if they are FIFO ordered.

Theorem 4 *There is no IS that guarantees causal interconnection for every pair of causal systems, even if they are FIFO ordered.*

PROOF. Let us assume, by way of contradiction, that there is a system S^T which is the result of interconnecting two FIFO-ordered causal systems S^0 and S^1 with the *IS I*. From Definition 10, we know that for every execution α^T there is a legal view β_p^T of α_p^T, for all p, preserving $\prec$.

13

Assume that we have an execution α^0 with the following write operations issued by process r of S^0: $w_r^0(x)s \prec w_r^0(y)l$. From the Liveness Property, we know that there is a time t after which any read operation on x and y issued by any process in S^0 returns s and l, respectively. We now assume that after this time t the processes p and g issue the write operations $w_p^0(x)u$ and $w_g^0(y)v$, causally related to each other through read operations (detailed below in each case). Following the definition of interface's behavior, we consider that when all *MCS-processes* update their local memory, the interface communicates isp^0 about these events. Then, I can take one of the following actions:

- *Case 1:* isp^1 issues $w_{isp^1}^1(x)u$ and $w_{isp^1}^1(y)v$, in this order, in S^1. Now, some process q of S^1 issues the following read operations $r_q^1(x)u \prec_q r_q^1(y)l$. In this case, if $r_g^0(y)l \prec w_g^0(y)v \prec r_p^0(y)v \prec w_p^0(x)u$, then it is impossible to form a legal view β_q^T preserving $\prec$. Hence, we reach a contradiction.
- *Case 2:* isp^1 issues $w_{isp^1}^1(y)v$ and $w_{isp^1}^1(x)u$, in this order, in S^1. Now, some process q of S^1 issues the following read operations $r_q^1(y)v \prec_q r_q^1(x)s$. In this case, if $r_p^0(x)s \prec w_p^0(x)u \prec r_g^0(x)u \prec w_g^0(y)v$, then it is impossible to form a legal view β_q^T preserving $\prec$. Hence, we reach a contradiction.
- *Case 3:* isp^1 does not issue $w_{isp^1}^1(y)v$ or $w_{isp^1}^1(x)u$ in S^1. From the Liveness Property, this case is not possible.

□

Nevertheless and although in general the interconnection of causal systems is not possible even if they are FIFO ordered, we found that it is still possible to interconnect causal systems that are *globally ordered*.

Globally Ordered Systems We say that a system is *globally ordered* if for each two write operations $w(x)v \prec w(y)u$ issued by (maybe different) processes in S^k, each $mcs(p)$ with p in S^k updates its local replica of x with the value v before updating its local replica of y with the value u.

In Figure 4, we present an *IS-protocol* that can be used to connect causal systems that are globally ordered. It consists of two concurrent tasks, $Propagate_{out}^k$ and $Propagate_{in}^k$, like in the *IS-protocol* in Fig. 3. In fact, the $Propagate_{in}^k$ task is the same. The key difference is found in task $Propagate_{out}$, where a pair $\langle x, v \rangle$ is not sent to the other systems until all the *MCS* replicas of x have been updated.

Clearly, globally ordered systems provide stronger guarantees than FIFO ordered systems. However and similar to this latter type, they do not undergo substantial constraints with respect to systems that are not globally ordered. Theorem 5 states formally the guarantees provided by the above mentioned protocol.
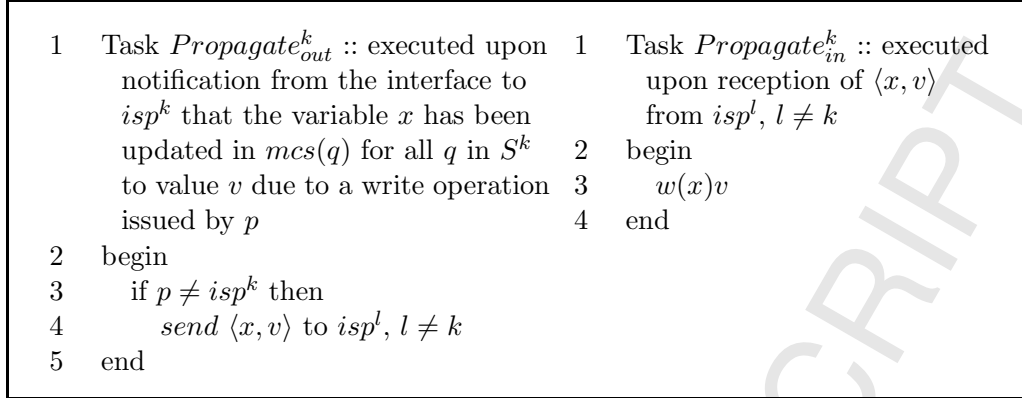
14

1	Task $Propagate^k_{out}$ :: executed upon notification from the interface to isp^k that the variable x has been updated in $mcs(q)$ for all q in S^k to value v due to a write operation issued by p	1	Task $Propagate^k_{in}$ :: executed upon reception of $\langle x, v \rangle$ from isp^l, $l \neq k$
2	begin	2	begin
3	if $p \neq isp^k$ then	3	$w(x)v$
4	send $\langle x, v \rangle$ to isp^l, $l \neq k$	4	end
5	end		

Fig. 4. The causal *IS-protocol* for each isp^k, $k \in \{0,1\}$.

Theorem 5 *The system S^T obtained by connecting N globally ordered causal systems $S^0, \cdots, S^{N-1}$ using the causal IS-protocol in Fig. 4 is causal.*

PROOF. See Appendix A.2. □

6 Interconnection of Cache Systems

In this section, we study the interconnection of cache systems [9]. Roughly speaking, this memory model forces independent variables considered in isolation to be sequential. That is, data operations on any individual variable must "appear" to have been executed atomically in an order that is consistent with the order seen in individual processes. Formally, a cache system is defined as follows:

Definition 11 (Cache System) *A system S is cache if for every execution α and every variable x there is a legal view β_x of α_x preserving $\prec$ (where α_x denotes the subset of operations obtained by removing from execution α all the operations on variables other than x).*

We show that, unlike the previous models, the interconnection of cache systems is always possible, independently of how they are implemented. In Figure 5, we present an *IS-protocol* that can be used to connect cache systems of any type. It consists of only one task $Propagate^k$. Note that each *IS-process* maintains a copy of the latest value propagated from the other system in $last(x)$ for each variable x. That copy must be initialized with a special value (namely, $NoData$). Note also that initially one of the *IS-processes* (for instance isp^0) must send a message with $\langle x, NoData \rangle$ to the other for each variable x to start the interconnection. Theorem 6 states formally the guarantees provided by the above mentioned protocol.

15

```
1    Task Propagate^k(x) :: executed upon reception of ⟨x, v⟩
       from isp^l, l ≠ k
2    begin
3        if v ≠ NoData then
4            w(x)v
5            last(x) := v
6        r(x)u
7        if u = last(x) then
8            u := NoData
9        send ⟨x, u⟩ to isp^l, l ≠ k
10   end
```

Fig. 5. The cache *IS-protocol* for each isp^k, $k \in \{0, 1\}$.

Theorem 6 *The system S^T obtained by connecting N cache systems (regardless of whether they are FIFO/globally ordered or not) $S^0, \cdots, S^{N-1}$, using the cache IS-protocol in Fig. 5 is cache.*

PROOF. See Appendix A.3. □

Observe that the proposed interconnection algorithm is usually highly inefficient in terms of the network traffic it causes, given that, for each memory object, there is a continuous exchange of messages between the *IS-processes*. Since the main target of this work is to identify whether consistency models can be interconnected, we are not very concerned about efficiency. However, we note that the algorithm can be easily optimized to decrease the network traffic. For instance, instead of handling each variable independently, the algorithm could work at a larger scale, dealing with several (or even all the) variables simultaneously. This optimization can be done by modifying task $Propagate^k()$ so that it is executed only upon reception of a given set of $\langle x, v \rangle$ pairs (for different variables), and executing the same code for each one of them. Additionally, instead of transferring continuously messages between the *IS-processes*, such a transfer could be performed only when a given time interval has passed or when the variable x is updated at the sending system, whichever happens first. This optimizations can be done by adding some simple pieces of code that implement this wait condition. Note, however, that in some cases, these changes could increase the latency. The correctness proofs of these alternative interconnecting protocols are in essence the same as the one for the algorithm shown in Figure 5.

16

7 Performance

In this paper, we have approached the interconnection problem from a theoretical point of view, trying to decide whether it is possible to interconnect given consistency models. We have not been concerned above with the efficiency of the interconnection algorithms we have proposed and the performance of the resulting interconnected system. However, we do here a brief and simple performance evaluation of algorithms and systems. We compare the performance of a system obtained using our *IS-protocols* with the performance of a system that uses a *MCS-protocol* connecting all the processes directly. 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*.

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 (in particular, cannot be blocked) by the interconnection. Since the three models that we study are fast, the response time of the algorithms that implement them (e.g., [15] for pRAM, [16,21] for causal, or [6] for cache) only depends on local computation at a node. This does not change with the interconnection.

Second, let us look at the *latency* of a DSM system, which is the largest time until a value that is written becomes visible in any other process. In a single DSM system this time depends on the *MCS-protocol* used. For instance, if we discard the time for local computations, for the pRAM and causal algorithms [15, 16, 21] the latency depends on the time to complete a broadcast in the system. Let us denote this time by $T_B(n)$ in a system with n *MCS-processes*, which we safely assume is at most linear on n. The only algorithm for cache consistency of which we are aware [6] has instead latency $L_{cache}(n) = \Theta(nT_B(n))$[4].

The three proposed *IS-protocols* propagate as soon as they can any new value they are aware of by sending a message. In the case of pRAM and causal consistencies this is done immediately, while in the case of cache consistency the *IS-process* may need to wait for a message from the other *IS-process*. Then, using the algorithms referred above, the interconnection of two systems S^0 and S^1 with n_0 and n_1 *MCS-processes* respectively ($n = n_0 + n_1$), has latency $T_B(n_0) + T_B(n_1) + d$ (*IS-processes* are assumed to use existing *MCS-processes*) in the case of pRAM and causal, where d is the delay of a point-to-point communication. In the case of cache this latency is $L_{cache}(n_0) + L_{cache}(n_1) + d$. In the three cases, since $T_B(n)$ is at most linear, the latency of the interconnected system is larger than the latency in the original system. However, since the

---

[4] The algorithm of [6] uses a token-passing scheme in which the n *MCS-processes* broadcast messages in a cyclic fashion.

broadcast delay cannot be smaller than d, the latency increases at most by a constant factor of 3. If this is generalized to N systems the increase factor depends on the topology. In the worst case, systems are connected as a line, and the worst factor becomes $2N - 1$. In the best case systems are connected as a star, and the increase factor is again bounded by a small constant 5.

Regarding the *network traffic*, the pRAM and causal protocols [15, 16, 21] broadcast a message for each write operation. Let $M_B(n)$ be the number of messages that a broadcast requires, which again is at most linear on n. Since the interconnection of S^0 and S^1 implies sending one point-to-point message for each variable update, the traffic in the interconnected system is $M_B(n_0) + M_B(n_1) + 1$ messages per write operation. This again implies a small increase of networks traffic, bounded by a factor of 3 if $M_B(n)$ is constant, and bounded by an additive constant term if $M_B(n)$ is linear. Generalizing for N systems the worst case is a factor of $2N - 1$, when $M_B(n)$ is constant and systems are connected as a line, and the best is an additive term of $O(N)$, when $M_B(n)$ is linear and systems are connected in a star.

The evaluation of network traffic for the cache consistency model cannot be based on messages per write operation, since both the basic *MCS-protocol* [6] and the *IS-protocol* of Figure 5 send messages continuously, even if variables are not written. The difference is that the *IS-protocol* sends one message per variable in the memory. If both protocols transmit at similar intervals, the single system sends $M_B(n)$ messages every interval, while the system after the interconnection of S^0 and S^1 sends $M_B(n_0) + M_B(n_1) + V$ messages, where V is the number of variables. Clearly, this latter value can be very large if V is large. However, as mentioned in Section 6, some optimizations can reduce the traffic. For instance, if all the changes in the memory are sent by the *IS-process* in one single message, the traffic becomes $M_B(n_0) + M_B(n_1) + 1$ messages. Then, the increase of traffic is similar to the one observed for the pRAM and causal consistencies.

8 Conclusions

In this paper, the interconnection of distributed shared memory systems has been studied. We have classified the consistency models in two groups, depending on whether they are *fast* or not. In the case of non-fast consistency models, we have shown that they cannot be interconnected in any way. In contrast, in the case of fast consistency models we have provided protocols with which to interconnect some of them. Whereas in some cases it is possible to interconnect fast consistency models without any restriction, other fast consistency models need some additional constraints. In this last situation, we gave sufficient conditions and the corresponding protocols to do so. Fig-

Memory Model	Globally Ordered Systems	FIFO Ordered Systems	General Systems
Non-fast Models	No	No	No
Causal	Yes	No	No
pRAM	Yes	Yes	No
Cache	Yes	Yes	Yes

Fig. 6. Possibilities of interconnection under the different types of systems in this work.

ure 6 summarizes these results. At this point, we note that whereas we have shown that cache systems can be interconnected in a more general fashion than pRAM and causal systems, the protocol that we have used for such a task is, in general, less efficient than the protocols used for interconnecting pRAM and causal systems.

References

[1] E. Jiménez, A. Fernández, V. Cholvi, Decoupled interconnection of distributed memory models, in: OPODIS, 2003, pp. 235–246.

[2] R. C. Steinke, G. J. Nutt, A unified theory of shared memory consistency, J. ACM 51 (5) (2004) 800–849.

[3] V. Cholvi, J. Bernabéu, Relationships between memory models, Inf. Process. Lett. 90 (2) (2004) 53–58.

[4] L. Higham, L. Jackson, J. Kawash, Specifying memory consistency of write buffer multiprocessors, ACM Trans. Comput. Syst. 25 (1) (2007) 1.

[5] S. Haldar, K. Vidyasankar, On specification of read/write shared variables, J. ACM 54 (6) (2007) 31.

[6] E. Jiménez, A. Fernández, V. Cholvi, A parametrized algorithm that implements sequential, causal, and cache memory consistencies, Journal of Systems and Software 81 (1) (2008) 120–131.

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

[8] L. Lamport, How to make a multiprocessor computer that correctly executes multiprocess programs, IEEE Transactions on Computers 28 (9) (1979) 690–691.

[9] J. Goodman, Cache consistency and sequential consistency, Tech. Rep. 61, IEEE Scalable Coherence Interface Working Group (March 1989).

[10] M. Ahamad, R. Bazzi, R. John, P. Kohli, G. Neiger, The power of processor consistency, in: Proceedings of the 5th ACM Symposium on Parallel Algorithms and Architectures, 1993, pp. 251–260.

[11] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, J. Hennessy, Memory consistency and event ordering in scalable shared-memory multiprocessors, in: Proceedings of the 17th Annual International Symposium on Computer Architecture, ACM, 1990, pp. 15–26.

[12] P. Keleher, Distributed shared memory using lazy consistency, Ph.D. thesis, Rice University (1994).

[13] B. Bershad, M. Zekauskas, W. Sawdon, The Midway distributed shared memory system, in: COMPCON, 1993.

[14] L. Iftode, J. P. Singh, K. Li, Scope consistency: a bridge between release consistency and entry consistency, in: SPAA '96: Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures, ACM Press, New York, NY, USA, 1996, pp. 277–287.

[15] R. Lipton, J. Sandberg, PRAM: A scalable shared memory, Tech. Rep. CS-TR-180-88, Princeton University, Department of Computer Science (September 1988).

[16] M. Ahamad, G. Neiger, J. Burns, P. Kohli, P. Hutto, Causal memory: Definitions, implementation and programming, Distributed Computing 9 (1) (1995) 37–49.

[17] A. Fernández, E. Jiménez, V. Cholvi, On the interconnection of causal memory systems, J. Parallel Distrib. Comput. 64 (4) (2004) 498–506.

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

[19] L. Lamport, On interprocess communication: Parts I and II, Distributed Computing 1 (2) (1986) 77–101.

[20] H. Attiya, R. Friedman, Limitations of fast consistency conditions for distributed shared memories, Information Processing Letters 57 (1996) 243–248.

[21] M. Ahamad, M. Raynal, G. Thia-Kime, An adaptive protocol for implementing causally consistent distributed services, in: ICDCS, 1998, pp. 86–93.

A Appendix

A.1 Correctness Proof of Theorem 3

Let p be some process in S^k, $k \in \{0, 1\}$ and β_p^k be a legal view of execution α_p^k preserving $\prec_q$ for all q in S^k, as in Definition 5. From Definition 9, such a legal view must exist by the fact that S^k is a pRAM system. We denote by $orig(op)$ the original write operation propagated as operation op in α_p^k by process isp^k. Similarly, given a write operation op issued in S^l, $l \neq k$, we denote by $prop(op)$ the write operation issued by isp^k as a result of propagating op to S^k as defined by the IS-protocol. We define β_p^T as the sequence obtained by replacing in β_p^k every write operation op from isp^k by the write operation $orig(op)$.

Lemma 1 β_p^T is formed by all operations of α_p^T.

PROOF. First of all, note that the difference between α_p^k and α_p^T is that, for each operation op issued by isp^k in α_p^k, α_p^T contains the original operation $orig(op)$. Since β_p^k is a sequence formed by all operations of α_p^k, and β_p^T is obtained by replacing in β_p^k every write operation op from isp^k by the write operation $orig(op)$, then the set of operations in β_p^T is the same as that of α_p^T. □

The following Lemmas show that β_p^T preserves the order in which the operations are issued in any process of S^T.

Lemma 2 Let $op = w_q^k(x)v$ and $op' = w_q^k(y)u$ be two operations of α^T issued by the same process q of S^k. If $op \prec_q op'$ on α^k, then $Propagate_{out}^k$ will send to S^l, $l \neq k$, $\langle x, v \rangle$ before $\langle y, u \rangle$.

PROOF. Directly since, as the system S^k is FIFO ordered, isp^k receives the message in S^k with the value v of variable x from process q before the message with the value u of variable y also from process q, and then $Propagate_{out}^k$ sends the pair $\langle x, v \rangle$ to isp^l before it sends $\langle y, u \rangle$. □

Lemma 3 Let op and op' be two write operations of α^T issued by the same process q of S^l, where $l \neq k$. If $op \prec_q op'$ on α^l, then $prop(op) \rightarrow prop(op')$ in β_p^k, for all p.

PROOF. We know that β_p^k is a legal view that preserves the q's process order $\prec_q$ on α^k, for all q. Then, the result follows from Lemma 2, from the

fact that the channel connecting isp^l to isp^k is reliable and FIFO, and from the implementation of task $Propagate_{in}^k$ (see Fig. 3). □

Lemma 4 β_p^T *preserves* $\prec_q$ *for all* q.

PROOF. By way of contradiction, let us assume that β_p^T does not preserve the order among operations issued by a process q of S^T. Hence, there must be at least two operations op and op' of α_p^T issued by q such that $op \prec_q op'$ but op' precedes op in β_p^T. Let us consider two possible cases.

- *Case 1: q is in* S^k. Since op' precedes op in β_p^T, op' also precedes op in β_p^k, by definition of β_p^T. Then, β_p^k does not preserve q's process order $\prec_q$. However, this is not possible since, by definition, β_p^k is a legal view preserving $\prec_q$, for all q. Hence, we reach a contradiction.
- *Case 2: q is in* S^l, $l \neq k$. Since both operations are in β_p^T, which only contains read operations from process p 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 3, we have that $prop(op) \rightarrow prop(op')$ in β_p^k. Observe now that, by definition, operation $prop(op)$ in β_p^k is replaced by op and operation $prop(op')$ is replaced by op' to obtain β_p^T. Then op precedes op' in β_p^T and we reach a contradiction.

□

Lemma 5 β_p^T *is legal.*

PROOF. By definition, β_p^k is legal. Also by definition, β_p^T is obtained by replacing in β_p^k every write operation op from isp^k by the write operation $orig(op)$. Therefore, β_p^T is legal. □

Theorem 7 (Corresponds to Theorem 3) *The system* S^T *obtained by connecting* N *FIFO ordered pRAM systems* $S^0, \cdots, S^{N-1}$ *using the pRAM IS-protocol in Fig. 3 is pRAM.*

PROOF. Let $N = 2$. From Lemma 1, β_p^T is formed by all operations of α_p^T. Also, from Lemma 4, β_p^T preserves $\prec_q$ for all q. Finally, from Lemma 5, β_p^T is legal. Then, β_p^T is a legal view of α_p^T preserving $\prec_q$ for all q. Hence, S^T is a pRAM system. The extension to more than 2 systems follows from Observation 1. □

22

A.2   Correctness Proof of Theorem 5

Let p be some process in system S^k, $k \in \{0, 1\}$, and let $mcs(p)$ be its *MCS-process*. Recall that α_p^k (resp. α_p^T) is the set obtained by removing from α^k (resp. α^T) all read operations except those from process p. We define β_p^k as a sequence with the same operations as α_p^k that preserves the order in which all operations of α_p^k are issued by process p, and the order in which every write operation is applied in $mcs(p)$. Formally,

Definition 12 *Let β_p^k be a sequence of the operations in α_p^k. Let op and op′ be in α_p^k. Then $op \rightarrow op'$ in β_p^k, if any of the following happen:*

(1) op and op′ are operations from the same process p of S^k and $op \prec_p op'$.
(2) $op = w_q^k(x)u$, $op' = w_s^k(y)v$, and in $mcs(p)$ the local copy of x is updated with u before updating y with v.
(3) $op = w_q^k(x)u$, $op' = r_p^k(y)v$, and in $mcs(p)$ the local copy of x is updated with u before p issues op′.

Note that, as in α_p^k, every write operation of process isp^k in β_p^k is the propagation of a write operation issued by a process of S^l, $l \neq k$. We define β_p^T as the sequence obtained by replacing in β_p^k every write operation op from isp^k by the write operation $orig(op)$.

Definition 13 (Non-Transitive Execution Order) *Let op and op′ be two operations in an execution α. Then op precedes op′ in the non-transitive execution order ($op \prec_{nt} op'$) on α if any of the following holds:*

(1) op and op′ are operations from the same process p and $op \prec_p op'$ on α.
(2) $op = w(x)v$ and $op' = r(x)v$.

Definition 14 ($\prec$–Related Sequence) *Let op and op′ be two operations in an execution α such that $op \prec op'$ on α. A $\prec$–related sequence between op and op′ is a sequence of operations $op^1, op^2, \ldots, op^m$ belonging to α such that $op^1 = op$, $op^m = op'$, and $op^i \prec_{nt} op^{i+1}$ on α, for $1 \leq i < m$.*

Note that at least one $\prec$–related sequence always exists between op and op' if $op \prec op'$ on α.

When considering the composed system S^T, a $\prec$–related sequence Seq between operations op and op' of execution α^T can be divided into n subsequences $subSeq_1, subSeq_2, \ldots, subSeq_n$, such that all the operations in subsequence $subSeq_i$ belong to the same system S^k and the operations in consecutive subsequences belong to different systems. We use $subSeq_i^k$ to express that all the operations of the ith subsequence belong to system S^k.

23

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

Lemma 6 *Let op and op' be two operations in α_p^T issued in system S^k such that $op \prec op'$ in α^T. If there is a $\prec$–related sequence between op and op' with one single subsequence $subSeq_1^k$, then $op \rightarrow op'$ in β_p^k.*

PROOF. Let us assume, by way of contradiction, that the claim does not hold. Then, $op \prec op'$ on α^k, and $op' \rightarrow op$ in β_p^k. This is only possible if there are at least two "consecutive" operations op^i and op^{i+n} in $subSeq_1^k$ and belonging to α_t^k such that $op^{i+n} \rightarrow op^i$ in β_p^k. We say op^{i+n} and op^i are two consecutive operations in $subSeq_1^k$ if they are in α_t^k, $t \neq p$, and between them there is no other operation belonging to α_p^k (i.e., every operation op^{i+l}, $1 \leq l < n$, is a read operation issued by a process other than p). Note that if $n > 1$ then these two consecutive operations op^i and op^{i+n} can only be write operations. We have three cases:

- *Case 1: $op^i = w^k(x)v$ and $op^{i+n} = w^k(y)u$.* From the definition of the $\prec$–related sequence, $op^i \prec op^{i+n}$ on α^k. As the system S^k is Globally Ordered, if $op^i \prec op^{i+n}$ on α^k, then op^i must be applied in all processes of S^k (and, of course, in p) before op^{i+n}. Therefore, from the second condition of Definition 12, $op^i \rightarrow op^{i+n}$ in β_p^k, and we reach a contradiction.
- *Case 2: $op^i = w^k(x)v$ and $op^{i+1} = r_p^k(x)v$.* From the definition of the $\prec$–related sequence, $op^i \prec_{nt} op^{i+1}$ on α^k. Obviously, the write operation $w^k(x)v$ must be applied before issuing $r_p^k(x)v$, since, otherwise, op^{i+1} could not obtain the value v in x. Therefore, from the third condition of Definition 12, $op^i \rightarrow op^{i+1}$ in β_p^k, and we reach a contradiction.
- *Case 3: op^i and op^{i+1} are issued by the same process p.* From the definition of the $\prec$–related sequence, $op^i \prec_{nt} op^{i+1}$ on α^k and, from case 1 of $\prec_{nt}$, $op^i \prec_p op^{i+1}$. Then, from the first condition of Definition 12, $op^i \rightarrow op^{i+1}$ in β_p^k, and we reach a contradiction.

□

Lemma 7 *Let op and op' be two operations in α_p^T.*

(1) *If they are issued by system S^k and $op \prec op'$ on α^T, then $op \rightarrow op'$ in β_p^k.*
(2) *If they are issued by system S^k and $op = w^k(x)v$ and $op' = w^k(y)u$ and $op \prec op'$ on α^T, then $Propagate_{out}^k$ will send the pairs $\langle x, v \rangle$ and $\langle y, u \rangle$ to S^l in this order.*
(3) *If they are issued by system S^l, $l \neq k$, and are write operations and $op \prec op'$ on α^T, then $prop(op) \rightarrow prop(op')$ in β_p^k.*

24

(4) *If they are issued by systems S^l and S^k respectively and $op = w^l(x)v \prec op'$ on α^T, then $prop(op) \to op'$ in β_p^k.*

(5) *If they are issued by systems S^k and S^l respectively and $op \prec op' = w^l(x)v$ on α^T, then $op \to prop(op')$ in β_p^k.*

PROOF.

Proof of Part 1: Let Seq be a $\prec$–related 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 6, $op = first(subSeq_1^k) \to op' = last(subSeq_1^k)$ in β_p^k. Assume the claim is true for sequences with i subsequences. We show it also holds if Seq has $i+2$ subsequences. By the induction hypothesis, we have that $op = first(subSeq_1^k) \to last(subSeq_i^k)$ in β_p^k. Note that $last(subSeq_i^k) = w_t^k(x)v$ is propagated to system S^l, $l \neq k$, by process isp^k after, in all processes of S^k, the local copy of x is updated with the value v. Later on, isp^l propagates the pair (y, u) from $last(subSeq_{i+1}^l) = w_q^l(y)u$ as $w_{isp^k}^k(y)u$ (see Fig. 4). Then, $w_t^k(x)v$ is applied by all processes in S^k (and, of course, by p) before $w_{isp^k}^k(y)u$ and therefore, from the second condition of β_p^k in Definition 12, $w_t^k(x)v \to w_{isp^k}^k(y)u$ in β_p^k. From the second condition of $\prec$–related order, $w_{isp}^k(y)u \prec_{nt} first(subSeq_{i+2}^k) = r_s^k(y)u$ on α^k, and then, $w_{isp}^k(y)u \prec op' = last(subSeq_{i+2}^k)$ on α^k. Then, from Lemma 6, $w_{isp^k}^k(y)u \to op' = last(subSeq_{i+2}^k)$ in β_p^k. Hence, by transitivity, $op = first(subSeq_1^k) \to op' = last(subSeq_{i+2}^k)$ in β_p^k.

Proof of Part 2: If there is a $\prec$–related sequence between op and op' with a single subsequence, then $op \prec op'$ on α^k, and it follows (since the system is globally ordered) that op is applied in all processes of S^k before op'. Otherwise, the proof of Part 1 shows the same fact when the $\prec$–related sequences between op and op' have more than one subsequence. Thus, since the task $Propagate_{out}^k$ of our IS–protocol (see Fig. 4) propagates operations in the order they are locally applied, it will send the pair $\langle x, v \rangle$ of op to S^l before the pair $\langle y, u \rangle$ of op'.

Proof of Part 3: From Part 1, $op \to op'$ in β_q^l, $l \neq k$. Then, the result follows from Part 2, from the fact that the channel connecting isp^l to isp^k is reliable and FIFO, and from the implementation of task $Propagate_{in}^k$ (see Fig. 4). The process isp^k issues $prop(op)$ and $prop(op')$ in S^k and, thus, from the first condition of execution order, $prop(op) \prec prop(op')$ on α^k. Hence, from Lemma 6, $prop(op) \to prop(op')$ in β_p^k.

Proof of Part 4: Let Seq be a $\prec$–related sequence with n subsequences between op and op'. Let us assume $last(subSeq_{n-1}^l) = w_q^l(y)u$ and $first(subSeq_n^k) = r_s^k(y)u$. From Part 3, $prop(op) \to prop(last(subSeq_{n-1}^l)) = prop(w_q^l(y)u) = w_{isp^k}^k(y)u$ in β_p^k. From second

25

condition of $\prec$–related order, $w_{isp}^k(y)u \prec_{nt} first(subSeq_n^k) = r_s^k(y)u$ on α^k, and then, $w_{isp}^k(y)u \prec op' = last(subSeq_n^k)$ on α^k. As we know, because the system S^k is Globally Ordered and from Definition 12 of β_p^k, $w_{isp^k}^k(y)u \rightarrow op' = last(subSeq_n^k)$ in β_p^k. Hence, by transitivity, $op = prop(op) \rightarrow op' = last(subSeq_n^k)$ in β_p^k.

Proof of Part 5: Similar to the proof of Part 4.

$\square$

Lemma 8 β_p^T *preserves* $\prec$.

PROOF. In this proof we show that if there are two operations op and op' in α_p^T such that $op \prec op'$ on α^T, then $op \rightarrow op'$ in β_p^T.

Let us make a case analysis:

- *Case op and op' are issued by processes in S^k:* From Part 1 of Lemma 7, if $op \prec op'$ on α^T, then $op \rightarrow op'$ in β_p^k. Then, by definition of β_p^T, we have that $op \rightarrow op'$ in β_p^T.
- *Case op and op' are issued by processes in S^l, where $l \neq k$:* Since both operations are in α_p^T, which only contains read operations from process p of system S^k, both operations must be write operations. Then, let op and op' be propagated as $prop(op)$ and $prop(op')$ operations.

  From Part 3 of Lemma 7, we have that if $op \prec op'$ on α^T, then $prop(op) \rightarrow prop(op')$ in β_p^k. Hence, replacing $prop(op)$ and $prop(op')$ by op and op', respectively, we have that, by definition of β_p^T, $op \rightarrow op'$ in β_p^T.
- *Case op is issued by some process in S^l and op' is issued by some process in S^k, where $l \neq k$:* op must be a write operation, since α_p^T only contains read operations from process p of system S^k. Such an operation will be propagated from S^l to S^k as described by the IS–protocol and it will appear in S^k as a (write) operation $prop(op)$ issued by process isp^k.

  From Part 4 of Lemma 7, if $op \prec op'$ on α^T, then $prop(op) \rightarrow op'$ in β_p^k. Hence, replacing $prop(op)$ by op, we have that, by definition of β_p^T, $op \rightarrow op'$ in β_p^T.
- *Case op is issued by some process in S^k and op' is issued by some process in S^l, where $l \neq k$:* op' must be a write operation, since α_p^T only contains read operations from process p of system S^k. Such an operation will be propagated from S^l to S^k as described by the IS–protocol and it will appear in S^k as a (write) operation $prop(op')$ issued by process isp^k.

  From Part 5 of Lemma 7, if $op \prec op'$ on α^T, then $op \rightarrow prop(op')$ in β_p^k. Hence, replacing $prop(op')$ by op', we have that, by definition of β_p^T, $op \rightarrow op'$ in β_p^T.

$\square$

Lemma 9 β_p^T *is legal.*

PROOF. If process p issues some read operation $op = r_p^k(x)u$ is because, when this operation is invoked, it has the value u in its local copy of x. Then, the latest write operation applied on x in p is $op' = w^k(x)u$. Hence, from the third condition of Definition 12, op' must be the previous nearest write operation on x in β_p^k. Therefore, from Definition 5, β_p^k is legal. Note that, by definition of β_p^T, if we replace in β_p^k every write operation op from isp^k by the write operation $orig(op)$, we obtain β_p^T. Then, β_p^T is legal. □

Theorem 8 (Corresponds to Theorem 5) *The system S^T obtained by connecting N globally ordered causal systems $S^0, \cdots, S^{N-1}$ using the causal IS-protocol in Fig. 4 is causal.*

PROOF. Let $N = 2$. From Lemma 1, β_p^T is formed by all operations of α_p^T. Also, from Lemma 8, β_p^T preserves $\prec$. Finally, from Lemma 9, β_p^T is legal. Then, β_p^T is a legal view of α_p^T preserving $\prec$. Hence, S^T is a causal system. The extension to more than 2 systems follows from Observation 1. □

A.3 Correctness Proof of Theorem 6

Let β_x^k be a legal view of $\alpha(x)^k$ preserving $\prec$ on $\alpha(x)^k$, as described in Definition 5. Such a legal view must exist by the fact that S^k is a cache system. We define op_i as the ith write operation propagated by process isp from one system to the other (regardless of the system in which it is issued). We use $op(x)_i^k$ to indicate that op_i is issued by some process in S^k on variable x. We use $prop^l(op(x)_i^k)$ to denote the write operation issued by the task $Propagate_i^l$ as a result of the propagation of $op(x)_i^k$.

We define $\beta_{x,i}^k$ as the subsequence of operations of β_x^k issued by processes of S^k from $op(x)_i^k$ (or $prop^k(op(x)_i^l)$) until $op(x)_{i+1}^k$ (or $prop^k(op(x)_{i+1}^l)$) without including them.

We define $\beta_{x,i}^T$ as the sequence formed by all operations issued by processes of S^T between the ith and $ith + 1$ propagation of write operations on variable x so that operations belonging to $\alpha(x)^k$ follow the order they have in β_x^k, and operations belonging to $\alpha(x)^l$ follow the order they have in β_x^l. Formally, $\beta_{x,i}^T$ can be obtained as follows: $op(x)_i \cdot head(x)_i^k \cdot head(x)_i^l \cdot tail(x)_i^k \cdot tail(x)_i^l$, where $head(x)_i^k$ denotes the subsequence of $\beta_{x,i}^k$ that includes all read operations from the beginning of $\beta_{x,i}^k$ until the first write operation in $\beta_{x,i}^k$ (not included), and $tail(x)_i^k$ is the subsequence of $\beta_{x,i}^k$ that includes all the operations in $\beta_{x,i}^k$ that are not in $head(x)_i^k$.

27

ACCEPTED MANUSCRIPT

Finally, we define β_x^T as the sequence obtained by concatenating the sequences of $\beta_{x,i}^T$ such that $\beta_{x,i}^T$ goes before $\beta_{x,i+1}^T, \forall i$. In what follows, we will prove that it is a legal view of $\alpha(x)^T$.

Lemma 10 β_x^T *is a sequence formed by all operations of* $\alpha(x)^T$.

PROOF. $\alpha(x)^T$ is, by definition, the set of all operations in $\alpha(x)^k$ and $\alpha(x)^l$ issued by all processes of S^k and S^l other than isp^k and isp^l (i.e., by all processes of S^T).

We know that β_x^k and β_x^l are sequences of all operations of $\alpha(x)^k$ and $\alpha(x)^l$, respectively, because they are legal views. Then, since β_x^T is formed as the sequence of operations of S^T obtained by concatenating the sequences of legal views β_x^k and β_x^l, it is a sequence of all operations of $\alpha(x)^T$. $\square$

Lemma 11 β_x^T *preserves* $\prec$.

PROOF.

We show that if there are two operations op and op' in $\alpha(x)^T$ such that $op \prec op'$ on $\alpha(x)^T$, then $op \rightarrow op'$ in β_x^T. We have two possible cases.

- *Case 1: op* and *op'* have been issued by processes of a same system. Let us suppose that op and op' are issued by processes of S^k. Note that, by definition, S^k is a cache system. Then, from Definition 11, there must be a legal view β_x^k preserving the execution order $\prec$ on $\alpha(x)^k$ and hence, from Definition 4, if $op \prec op'$ on $\alpha(x)^k$ then $op \rightarrow op'$ in β_x^k. It is easy to check from the definition of β_x^T that operations of α^T and issued by processes of S^k appear in β_x^T and in β_x^k in the same order. Hence, $op \rightarrow op'$ in β_x^T.
- *Case 2: op* and *op'* been issued by processes of different systems. Let us suppose that op is issued by some process of S^k, and op' is issued by some process of S^l. We know, from Case 1, that β_x^T preserves $\prec$ on $\alpha(x)^k$, and also preserves $\prec$ on $\alpha(x)^l$. Then, β_x^T will preserve $\prec$ on $\alpha(x)^T$ if it also preserves $\prec$ between any two operations from different systems. Then, by definition of β_x^T, it is enough to show that the second condition of the Execution Order is preserved between two operations op and op' from different systems such that $op = w^k(x)u$ and $op = r^l(x)u$ in $\beta_{x,i}^T$. We can see, by definition, that op must be the ith write operation propagated from S^k to S^l (that is, $op(x)_i^k$), and op' is a read operation in $head(x)_i^l$. Then, by definition, $op \rightarrow op'$ in $\beta_{x,i}^T$, and, hence, $op \rightarrow op'$ in β_x^T.

$\square$

Lemma 12 β_x^T *is legal.*

28

PROOF. Let $op = r(x)u$ be a read operation of β_x^T. From Definition 5, β_x^T is legal if $op' = w(x)u$ is the nearest previous write operation to op in β_x^T. We know, by definition, that β_x^k is the same sequence as β_x^T but replacing each write operation op from isp^k by $prop(op)$. We have two possible cases.

- *Case 1:* $op = r^k(x)u$ and $op' = w^k(x)u$ are operations in $\alpha(x)^T$ issued by processes of S^k. By definition, as β_x^k is a legal view of execution $\alpha(x)^k$ preserving $\prec$ on $\alpha(x)^k$, $op' = w^k(x)u$ is the nearest previous write operation to $op = r^k(x)u$ in β_x^k. Then, by definition of β_x^T, op' is also the nearest previous write operation to op in β_x^T. Therefore, β_x^T is legal.
- *Case 2:* $op = r^k(x)u$ and $op' = w^l(x)u$ are operations in $\alpha(x)^T$ issued by systems S^k and S^l respectively. Let $op' = w^l(x)u$ be the write operation $op(x)_i^l$. Then, its corresponding write operation in S^k is $prop^k(op(x)_i^l) = w_{isp^k}^k(x)u$. By definition, as β_x^k is a legal view of $\alpha(x)^k$ preserving $\prec$ on $\alpha(x)^k$, $prop^k(op(x)_i^l)$ is the nearest previous write operation to op in β_x^k. Then, by definition of β_x^T, $prop^k(op(x)_i^l)$ is replaced by $op' = op(x)_i^l$ to obtain β_x^T, and $op' = op(x)_i^l$ is also the nearest previous write operation to op in β_x^T. Therefore, β_x^T is legal.

□

Theorem 9 (Corresponds to Theorem 6) *The system S^T obtained by connecting N cache systems (regardless of whether they are FIFO/globally ordered or not) $S^0, \cdots, S^{N-1}$, using the cache IS-protocol in Fig. 5 is cache.*

PROOF. Let $N = 2$. From Lemma 10, β_x^T is formed by all operations of $\alpha(x)^T$. From Lemma 11, β_x^T preserves $\prec$. Finally, from Lemma 12, β_x^T is legal. Then, β_x^T is a legal view of $\alpha(x)^T$ preserving $\prec$. Hence, S^T is a cache system. The extension to more than 2 systems follows from Observation 1. □