On the Interconnection of Causal Memory Systems*

Antonio Fernández
Universidad Rey Juan Carlos
28933 Móstoles, Spain
afernandez@acm.org

Ernesto Jiménez
Universidad Politécnica de Madrid
28031 Madrid, Spain
ernes@eui.upm.es

Vicent Cholvi
Universitat Jaume I
12071 Castellón, Spain
vcholvi@inf.uji.es

ABSTRACT

A large amount of work has been invested in devising algorithms to implement distributed shared memory (DSM) systems under different consistency models. However, to our knowledge, the possibility of interconnecting DSM systems with simple protocols and the consistency of the resulting system has never been studied. With this paper, we start a series of works on the properties of the interconnection of DSM systems, which tries to fill this void.

In this paper, we look at the interconnection of propagation-based causal DSM systems. We present extremely simple algorithms 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 generalized to interconnect any number of DSM propagation-based causal systems.

1. INTRODUCTION

Shared memory (reading and writing of shared variables) is a well-known mechanism for inter-process communication in concurrent programs. However, while the semantic of read and write operations in sequential programs is clear, the situation is different for 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* [1, 2, 4, 6, 12].

The consistency memory models proposed in the literature can be broadly classified into *strong* and *weak* memory models. The strong memory models are close in behavior to a centralized memory, which makes simple to write programs with them (the return value of each read operation is rather predictable). However, it is widely accepted that strong memory models do not scale well with the number of processes [3, 6]. On the other hand, weaker memory models can be more efficiently implemented, since they require less consistency overhead. This implies more possible return values for each read operation [2, 5, 8], which makes harder to write programs for these models. Hence, we are faced with a tradeoff between simplicity of programming and performance of the consistency model implementation.

The *causal* memory model has attracted the attention of a number of researchers because is considered to be powerful enough to allow easy programming (like strong memory models), but at the same time allows for inexpensive implementations (like weak memory models). As a consequence, a number of algorithms implementing the causal memory model have been proposed in the literature (see for instance [2, 9, 10]). Most algorithms 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

A huge amount of work has been invested in devising algorithms for implementing distributed shared memory systems under different memory models, as well as in studying the type of problems that can be solved with them. However, to our knowledge, there has not been any work on devising algorithms to interconnect these DSM systems. The present work attempts to be the first investigating the interconnection of DSM systems.

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

*This work is partially supported by the CICYT under grant TEL99-0582 and the Comunidad Autónoma de Madrid under grant CAM-07T/00112/1998.

PODC 2000 Portland Oregon

show that the resulting system is also causal. For instance, the proposed algorithms could be used to obtain a causal DSM system by combining systems implemented with the algorithm proposed by Ahamad et al. [2], and systems implemented with the algorithm proposed by Prakash et al. [9] (just to point out two of them).

We first study the connection of two propagation-based causal system. We assume the existence in each system of a special process, called *gate* process, which will be in charge of actually running the interconnection algorithm. Those gate processes are connected by a reliable FIFO channel, which will be used to exchange the data required by the interconnection. We present algorithms that can be run by the gate processes in order to connect both systems. Basically, these algorithms propagate the variable updates from one system to the other. For this reason, we denote them *bridge* algorithms. We then show that the system obtained by connecting two systems with the gate processes, running the proposed bridge algorithms, is causal.

Next, we show that the interconnection scheme for two systems can be generalized for a larger number of systems. Hence, we show that several propagation-implemented causal systems can be interconnected with our bridge algorithms to obtain a large causal system.

Note that the sequential memory model, which is maybe the most widely known, is in fact causal. Hence, these results also apply to it. Therefore, our results also show that two sequential systems can be interconnected via a FIFO channel maintaining the causality of the overall resulting system.

1.2 Interest of our work

An important question to pose is why interconnecting causal systems with new algorithms instead of using a known algorithm for the full system. There are several reasons for using this new approach. First, in this way, we can interconnect systems that are already running different causal algorithms without changing them. They can keep using different algorithms at their local level.

Second, it could be interesting to use different algorithms in different environments and our approach to combine the resulting systems. When choosing the causal algorithms to use in a system, it is basic to analyze the characteristics of the network on which the system will run. For example, important characteristics of a network are the latency or the availability of multicast support. If we want to have a causal system in which there are several networks with different characteristics involved, it may be convenient or more efficient to use the most appropriate algorithm in each network and use our methods to connect the systems obtained. 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. In this case, it would seem appropriate to use one of the causal algorithms previously proposed in each of the local area networks, and use our bridge algorithms via the link to connect the whole system.

The rest of the paper is organized as follows. In Section 2 we introduce the basic framework and provide a formal definition of causal system. In Section 3 we introduce the bridge algorithms we propose for interconnecting two causal systems. In Section 4 we show that the union of two causal systems with the bridge algorithms is causal. In Section 5 we show that our approach can be used to connect more than two causal systems. Finally, in Section 6 we present our concluding remarks.

2. PRELIMINARIES

In this section we provide a formal definition of causal distributed shared memory systems. A *distributed shared memory system* (*DSM system* or *system* for short) consists of a set of 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 i (within the system S^q) that v is stored in the variable x. To simplify the notation, we assume that a given value is written at most once in any given variable. This assumption does not introduce new restrictions, since it can be forced by associating a timestamp[1] with every write operation. We also assume that the initial values of the variables are set by using write operations.

A *system computation* α^q of a system S^q consists of the 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 $\xrightarrow{\alpha^q}$ the order in which the operations in α^q happen. For operations of the same process i, $\xrightarrow{\alpha^q}$ also defines the order in which these operations have been executed by i. We now introduce the serial computation concept.

DEFINITION 1 (SERIAL COMPUTATION). *A computation* α^q *is* serial *if* $\forall op = r_i^q(x)v\ (\exists op' = w_j^q(x)v : op' \xrightarrow{\alpha^q} op$ *and* $\nexists op'' = w_k^q(x)u : op' \xrightarrow{\alpha^q} op'' \xrightarrow{\alpha^q} op)$.

In order to capture "causality" (in the sense of [7]), we need to define the causal order[2].

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* $op \xrightarrow{\alpha^q} op'$.
2. $op = w_i^q(x)v$ *and* $op' = r_j^q(x)v$
3. $\exists op'' : op \prec^{\alpha^q} op'' \prec^{\alpha^q} op'$

[1] Note that there are logical implementations of clocks that provide finite values[11].

[2] The causal order is actually a preorder, since the antisymmetric relation does not necessarily hold.

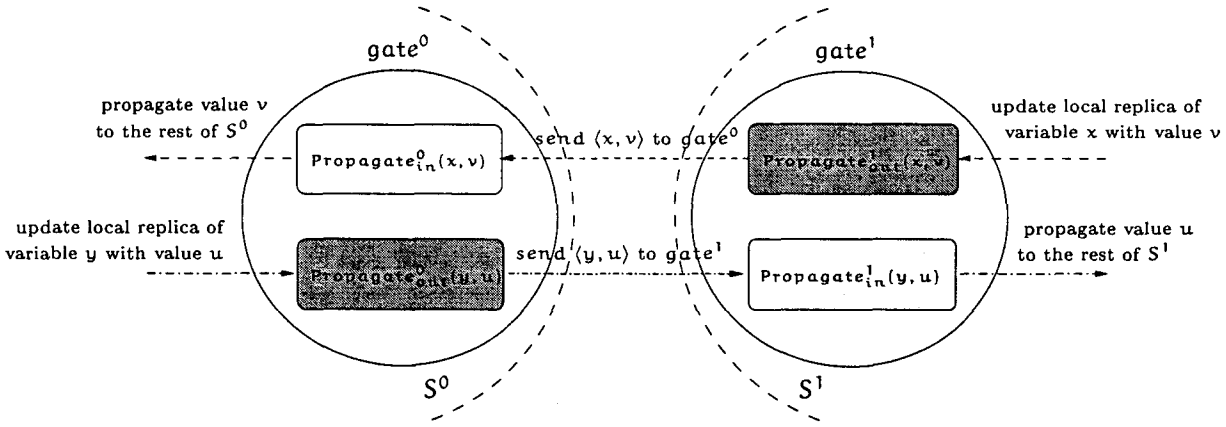

Figure 1: Task scheme of the bridge algorithms.

From this last definition, we also derive the *non-transitive causal order* as a restriction of the causal order if the transitive closure (i.e., the third condition) is not applied. We use $op \prec_n^{\alpha^q} op'$ to denote that op precedes op′ in the non-transitive causal order. Note that, if $op \prec_n^{\alpha^q} op'$, then op has been executed before op′. Hence, if $op \prec^{\alpha^q} op'$ then op has been executed before op′.

By using the causal order and the serial computation concept, we now define both causal view and causal computation as follows. Let α^q be a computation of system S^q, and the causal order $\prec^{\alpha^q}$ be as previously defined.

DEFINITION 3. *We say that* β_i^q *is a* causal view *of* α_i^q *if it is a permutation of* α_i^q*, and each prefix of* β_i^q *is serial and preserves the causal order* $\prec^{\alpha^q}$.

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

Finally, we provide the causal system definition.

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

Note that the definition of causal system only imposes a restriction on the possible computations observed. It does not impose any restriction on how to enforce these computations. Hence, it does not restrict the algorithms used to implement the causal system.

As we said, in this paper we will consider only causal systems implemented with *replication* and *propagation*. We assume that each process of a causal system has a replica of each of the variables of the shared memory. The replicas of a given process are managed by a *local causal module* of the causal algorithm, which is in charge of updating them. Every time a process issues a write operation, this is propagated to the causal module of the other processes of the system. Each causal module chooses the appropriate instant at which actually update the local replica. When a process issues a read operation, it obtains the current value of the local replica of the variable to be read.

It is important to note that the causal module of a process is completely free to choose the instant at which a local replica of a variable is updated, as long as the resulting computation is causal. Note that the causality of a computation strongly depends on the values obtained by the read operations. Then, the causal module of a process that seldom reads has a significant freedom on the order and time the local replicas are updated. For instance, if two write operations on variables x and y are issued, and the write on x causally precedes the write on y, the causal module of some process can choose to update first the local replica of y and then the local replica of x, as long as the local read operations issued between the updates do not violate the causality (e.g., do not show the change of order). However, we want to note that we do not know of any causal algorithm that uses this possibility of playing with the updating order, possibly due to the added complexity and bookkeeping involved. In this work we consider both kinds of algorithms.

3. THE BRIDGE ALGORITHMS FOR INTERCONNECTING CAUSAL SYSTEMS

In this section, we introduce algorithms for interconnecting two propagation-based causal systems S^0 and S^1, which we will show ensure that the resulting system is also causal. We will refer to these algorithms as *bridge algorithms* since, as we said, all they do is basically propagating the variable updates from one system to the other. We present two bridge algorithms, so that each system will choose which one to use depending on which causal algorithm it is running. We consider two classes of causal algorithms, depending on whether they guarantee the following property.

PROPERTY 1. *In any computation* α^k *of system* S^k *(where* $k \in \{0, 1\}$*), if processes* i *and* j *issue the write operations* $w_i^k(x)v$ *and* $w_j^k(y)u$*, and* $w_i^k(x)v \prec^{\alpha^k} w_j^k(y)u$*, then every process of the system* S^k *will update its local replica of* x *with value* v *before updating its local replica of* y *with*

$Propagate^k_{out}(x,v)$ :: task which is activated immediately after the local replica of variable x in $gate^k$ is updated with value v.
begin
if $\langle x,v\rangle$ was not received from $gate^{\overline{k}}$ then
$r^k_{gate^k}(x)v$
send $\langle x,v\rangle$ to $gate^{\overline{k}}$
end
	$Propagate^k_{in}(x,v)$ :: atomic task which is activated on the reception of a pair $\langle x,v\rangle$ from $gate^{\overline{k}}$.
	begin
	$w^k_{gate^k}(x)v$
	end

Figure 2: The bridge algorithm for systems satisfying Property 1.

value u.

We will assume that for each system S^k (where $k \in \{0,1\}$) there is a "gate" process, denoted $gate^k$. Such a process does not perform any user operation and is in charge of executing the bridge algorithm of the corresponding system. It is worthwhile to remark that $gate^k$ is part of the system S^k; for that reason, $gate^k$ has a local replica of each variable of the shared memory, and those replicas are updated following the causal algorithm implemented in S^k.

Each bridge algorithm contains two concurrent atomic tasks, $Propagate^k_{out}$ and $Propagate^k_{in}$. Whereas $Propagate^k_{out}$ deals with transferring write operations issued in S^k to the system $S^{\overline{k}}$ (we use $\overline{k}$ to denote $1-k$), $Propagate^k_{in}$ deals with applying within S^k the write operations transferred from the system $S^{\overline{k}}$ by $Propagate^{\overline{k}}_{out}$. To work properly, $Propagate^k_{out}$ has to guarantee that two causally ordered write operations are transferred to $S^{\overline{k}}$ following the causal order, using a reliable FIFO ordered communication channel. Similarly, $Propagate^k_{in}$ must apply the write operations transferred from S^k in exactly the same order they are received. An scheme of how the bridge algorithms work is shown in Fig. 1

As it has been said, our algorithm requires a reliable FIFO ordered communication channel. Note, however, that nothing has been said about how to implement it. In a practical case, this channel could be implemented in a number of ways, either by using shared memory or by using message passing.

We will first consider a system implemented with a causal algorithm that satisfies Property 1. As we said, all the causal algorithms we have found fall within this class. Fig. 2 shows an implementation of the $Propagate^k_{in}$ and $Propagate^k_{out}$ tasks for this case. Task $Propagate^k_{out}$ is activated with parameters x and v immediately after the local replica of variable x in $gate^k$ is updated with value v. As a result, it sends the pair $\langle x,v\rangle$ to the $gate^{\overline{k}}$ process, but only if such a pair was not received from $gate^{\overline{k}}$. This condition prevents that pair from going back and forth between $gate^k$ and $gate^{\overline{k}}$. It is important that the update of the variable and the task $Propagate^k_{out}$ are executed as an atomic action.

On its turn, task $Propagate^k_{in}$ is activated with parameters x and v whenever the pair $\langle x,v\rangle$ is received from the process $gate^{\overline{k}}$. As a result, it performs a causal write operation, thus causally propagating the value v to all the replicas of variable x within S^k. Note that, in order to avoid race conditions, we require task $Propagate^k_{in}$ to be atomic as well.

Let us consider now the more general case in which Property 1 is not necessarily satisfied by the causal algorithm of the system S^k. In this case, we need a second implementation of task $Propagate^k_{out}$, which is only slightly different from that of Fig. 2. In this implementation, on top of the process shown in Fig. 2, immediately before the local replica of variable x in $gate^k$ is updated with a new value v, $Propagate^k_{out}$ issues a read operation on x, $r^k_{gate^k}(x)s$, where the previous value s of x is read. This implementation enforces that two causally ordered write operations are propagated by $Propagate^k_{out}$ following the causal order. It is important that the read operation described, the update of the variable, and the process shown in Fig. 2 are executed together in an atomic fashion.

The following lemma presents the fundamental property satisfied by both bridge algorithms.

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

PROOF. The claim follows if the causal algorithm used by S^k satisfies Property 1 and the algorithm of Fig. 2 is used, since in $gate^k$ the local replica of x is updated with v before the local replica of y is updated with u, and $Propagate^k_{out}$ sends the pairs in the same order the updates are applied.

Now, we show by contradiction that, if we use the second implementation of $Propagate^k_{out}$, then the local replicas of x and y of $gate^k$ are also updated in that order. This will prove the property, since $Propagate^k_{out}$ sends the pairs in the same order the updates are applied. Let us assume, 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 $gate^k$, and since the system is causal, the resulting computation $\alpha^k_{gate^k}$ must have a causal view.

From the description above of the second implementation of $Propagate^k_{out}$ and our assumption, $gate^k$ has issued the following operations, in this order: $r^k_{gate^k}(y)t$, $r^k_{gate^k}(y)u$,

$r^k_{gate^k}(x)s$, and $r^k_{gate^k}(x)v$, where t and s are the previous values of y and x, respectively. Hence, from the first condition of the definition of $\prec^{\alpha^k}$, $r^k_{gate^k}(y)u \prec^{\alpha^k} r^k_{gate^k}(x)s \prec^{\alpha^k} r^k_{gate^k}(x)v$. We also have that $w^k_i(x)v \prec^{\alpha^k} w^k_j(y)u$ from the statement of the lemma. Finally, from the second condition of the definition of $\prec^{\alpha^k}$, $w^k_j(y)u \prec^{\alpha^k} r^k_{gate^k}(y)u$. Since any causal view $\beta^k_{gate^k}$ of $\alpha^k_{gate^k}$ must preserve the order $\prec^{\alpha^k}$, the above operations on x must appear in $\beta^k_{gate^k}$ in the order $w^k_i(x)v \overset{\beta^k_{gate^k}}{\rightarrow} r^k_{gate^k}(x)s \overset{\beta^k_{gate^k}}{\rightarrow} r^k_{gate^k}(x)v$. Let us consider now the operation $w^k_l(x)s$ that writes s in x. There are three possible cases, either there is no such operation in $\beta^k_{gate^k}$, $w^k_i(x)v \overset{\beta^k_{gate^k}}{\rightarrow} w^k_l(x)s$, or $w^k_l(x)s \overset{\beta^k_{gate^k}}{\rightarrow} w^k_i(x)v$. In either case, the seriality of $\beta^k_{gate^k}$ is violated and α^k can not be causal, which is a contradiction. Hence, the local replica of x must be updated before that of y. □

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 the bridge algorithms, is causal. We consider that the set of processes of S^T includes all the processes in S^0 and S^1 except $gate^0$ and $gate^1$ (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 gate processes running bridge algorithms. Similarly, α^k (where $k \in \{0,1\}$) 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 $w^{\overline{k}}_i(x)v$ in α^T issued by some processes i in $S^{\overline{k}}$ appears in α^k as the write operation $w^k_{gate^k}(x)v$ issued by the process $gate^k$ in S^k. This is so because *every* write operation issued by $gate^k$ in α^k is, from the bridge algorithms, just the propagation of a write operation issued by a process of $S^{\overline{k}}$.

DEFINITION 6. *Let* op *and* op′ *be two operations in* α^T *such that* op $\prec^{\alpha^T}$ 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* $op^i \prec^{\alpha^T}_n op^{i+1}$ *for* $1 \leq i < m$.

Note that at least one causal sequence always exists between op and op′ if op $\prec^{\alpha^T}$ op′. A causal sequence Seq between op and op′ can be divided in 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 $subSeq^k_i$ to express that all the operations of the ith subsequence belong to system S^k.

We use $first(subSeq_i)$ and $last(subSeq_i)$ to denote the first and last operation of the subsequence $subSeq_i$, respectively. Note that, in two consecutive subsequences $subSeq^k_i$ and $subSeq^{\overline{k}}_{i+1}$ of a given sequence, $last(subSeq^k_i) = w^k_j(x)v$ and $first(subSeq^{\overline{k}}_{i+1}) = r^{\overline{k}}_l(x)v$, i.e. the first operation of the later subsequence reads the value written by the last operation of the former subsequence.

LEMMA 2. *Let* op *and* op′ *be two operations in* α^T *such that* op $\prec^{\alpha^T}$ op′. *If there is a causal sequence between* op *and* op′ *with one single subsequence* $subSeq^k_1$, *then* op $\prec^{\alpha^k}$ op′.

PROOF. Let us assume, by the way of contradiction, that the claim does not hold. Then there must be two operations op^i and op^{i+1} of $subSeq^k_1$ such that $op^i \prec^{\alpha^T}_n op^{i+1}$, but does not hold $op^i \prec^{\alpha^k}_n op^{i+1}$. However, if $op^i \prec^{\alpha^T}_n op^{i+1}$ we have two cases:
Case 1: $op^i \overset{\alpha^T}{\rightarrow} op^{i+1}$ and both operations are issued by the same process. Since the operations issued by this process of S^k appear in the same order in α^T and α^k, then $op^i \overset{\alpha^k}{\rightarrow} op^{i+1}$. Hence $op^i \prec^{\alpha^k}_n op^{i+1}$, from the first condition of the non-transitive causal order definition, and we reach a contradiction.
Case 2: $op^i = w^k_j(x)v$ and $op^{i+1} = r^k_l(x)v$ (where j and l are two processes in S^k). But then $op^i \prec^{\alpha^k}_n op^{i+1}$, from the second condition of the non-transitive causal order definition, and we reach a contradiction. □

LEMMA 3. *Let* op *and* op′ *be two operations in* α^T *issued by system* S^k *such that* op $\prec^{\alpha^T}$ op′. *Then* op $\prec^{\alpha^k}$ 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 $subSeq^k_1$. Hence, from Lemma 2, op = $first(subSeq^k_1) \prec^{\alpha^k}$ op′ = $last(subSeq^k_1)$.

Assume the claim is true for sequences with i subsequences. We show it also holds if Seq has i + 2 subsequences. By induction hypothesis, we have that op = $first(subSeq^k_1) \prec^{\alpha^k} last(subSeq^k_i)$. Note that $last(subSeq^k_i) = w^k_j(x)v$ is propagated to system $S^{\overline{k}}$ by process $gate^k$. Before doing so, $gate^k$ issues the operation $r^k_{gate^k}(x)v$ (see $Propagate^k_{out}$ of Fig. 2). Later on, $gate^k$ propagates $last(subSeq^{\overline{k}}_{i+1}) = w^{\overline{k}}_l(y)u$ as $w^k_{gate^k}(y)u$ (see $Propagate^k_{in}$ of Fig. 2). From the definition of causal order, $w^k_j(x)v \prec^{\alpha^k} r^k_{gate^k}(x)v \prec^{\alpha^k} w^k_{gate^k}(y)u$. From Lemma 2 we have that $first(subSeq^k_{i+2}) = r^k_s(y)u \prec^{\alpha^k} op' = last(subSeq^k_{i+2})$. Also, $w^k_{gate^k}(y)u \prec^{\alpha^k} first(subSeq^k_{i+2}) = r^k_s(y)u$. Hence, by transitivity, op = $first(subSeq^k_1) \prec^{\alpha^k} op' = last(subSeq^k_{i+2})$. □

To set up some notation, given a write operation op issued in $S^{\overline{k}}$, we denote by prop(op) the write operation issued by $gate^k$ as a result of propagating op to S^k as defined by the bridge algorithms.

LEMMA 4. *Let* op *and* op′ *be two write operations in* α^T *issued by system* $S^{\overline{k}}$. *If* op $\prec^{\alpha^T}$ op′, *then* prop(op) $\prec^{\alpha^k}$ prop(op′).

PROOF. From Lemma 3, op $\prec^{\alpha^{\overline{k}}}$ op′. Then, the result follows from Lemma 1, the fact that the channel connecting $gate^{\overline{k}}$ to $gate^k$ is reliable and FIFO, and the implementation of task $Propagate^k_{in}$ (see Fig. 2). □

LEMMA 5. *Let* op *and* op′ *be two operations in* α^T *issued respectively by systems* $S^{\overline{k}}$ *and* S^k, *such that* op = $w_i^{\overline{k}}(x)v \prec^{\alpha^T}$ op′. *Then* prop(op) $\prec^{\alpha^k}$ op′.

PROOF. Let Seq be a causal sequence between op and op′. Let us assume $last(subSeq_1^{\overline{k}}) = w_j^{\overline{k}}(y)u$ and $first(subSeq_2^k) = r_i^k(y)u$. From Lemma 4, prop(op) $\prec^{\alpha^k}$ $prop(last(subSeq_1^{\overline{k}})) = prop(w_j^{\overline{k}}(y)u) = w^k_{gate^k}(y)u$. From Lemma 3 we have that $first(subSeq_2^k) = r_i^k(y)u \prec^{\alpha^k}$ op′. From the definition of causal order $w^k_{gate^k}(y)u \prec^{\alpha^k} r_i^k(y)u$. Hence, from transitivity, prop(op) $\prec^{\alpha^k}$ op′. □

LEMMA 6. *Let* op *and* op′ *be two operations in* α^T *issued respectively by systems* S^k *and* $S^{\overline{k}}$, *such that* op $\prec^{\alpha^T}$ op′ = $w_i^{\overline{k}}(x)v$. *Then* op $\prec^{\alpha^k}$ prop(op′).

PROOF. Let Seq be a causal sequence between op and op′ with n subsequences. Let us assume $last(subSeq^k_{n-1}) = w_j^k(y)u$ and $first(subSeq_n^{\overline{k}}) = r_i^{\overline{k}}(y)u$. From Lemma 3, op $\prec^{\alpha^k}$ $last(subSeq^k_{n-1}) = w_j^k(y)u$. From the implementation of task $Propagate^k_{out}$ (see Fig. 2) the value u is read from y by $gate^k$ before propagating it. Hence, from the definition of causal order, $w_j^k(y)u \prec^{\alpha^k} r^k_{gate^k}(y)u$. Since $r_i^{\overline{k}}(y)u$ has to be executed after the propagation of $w_j^k(y)u$, so has to be op′. Then, prop(op′) = $w^k_{gate^k}(x)v$ is executed after $r^k_{gate^k}(y)u$, and $r^k_{gate^k}(y)u \prec^{\alpha^k}$ prop(op′) = $w^k_{gate^k}(x)v$. Hence, from transitivity, op $\prec^{\alpha^k}$ prop(op′). □

Since S^k is a causal system, α^k has to be causal. Therefore, any sequence α_i^k (see Definition 3) has at least one causal view β_i^k. Like in α^k, every write operation in β_i^k (see also Definition 3) of the process $gate^k$ is the propagation of a write operation issued by a process of $S^{\overline{k}}$. Let us denote by orig(op) the original write operation propagated as operation op in β_i^k by process $gate^k$. Now, from β_i^k, we will derive a sequence γ_i^k which we will show is a causal view of $\alpha^T_{i(k)}$, defined as the sequence obtained by removing from α^T all read operations except those from the process i in system S^k.

DEFINITION 7. γ_i^k *is the sequence obtained by replacing in* β_i^k *every write operation* op *from* $gate^k$ *by the write operation* orig(op).

Note that every prefix of γ_i^k can also be obtained from the corresponding prefix of β_i^k with same length by replacing every write operation op from $gate^k$ by the write operation orig(op).

LEMMA 7. γ_i^k *is a permutation of the operations in* $\alpha^T_{i(k)}$.

PROOF. $\alpha^T_{i(k)}$ 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 $gate^k$ of all the write operations in α^T of processes in system $S^{\overline{k}}$. Then, the difference in their respective sets of operations is that, for each operation op issued by $gate^k$ in α_i^k, $\alpha^T_{i(k)}$ 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. Finally, γ_i^k is obtained from β_i^k by replacing each op issued by $gate^k$ by orig(op). Hence, the set of operations in γ_i^k is the same as that of $\alpha^T_{i(k)}$. □

LEMMA 8. *Each prefix of* γ_i^k *preserves the causal order* $\prec^{\alpha^T}$.

PROOF. The proof uses contradiction. We show that if some prefix of γ_i^k does not preserve the order $\prec^{\alpha^T}$ then some prefix of β_i^k does not preserve the order $\prec^{\alpha^k}$. But since β_i^k is a causal view of α_i^k, by definition of causal view each prefix of β_i^k must preserve the causal order $\prec^{\alpha^k}$, and we reach a contradiction.

Then, by way of contradiction, let us assume there is a prefix of γ_i^k, denoted $prefix(\gamma_i^k)$, which does not preserve the order $\prec^{\alpha^T}$. Let $prefix(\beta_i^k)$ denote the corresponding prefix of β_i^k. Hence, there must be at least two operations op and op′ such that op $\prec^{\alpha^T}$ op′ but op′ precedes op in $prefix(\gamma_i^k)$. 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 $\prec^{\alpha^k}$ op′. Now note that since op′ precedes op in $prefix(\gamma_i^k)$, op′ also precedes op in $prefix(\beta_i^k)$, by definition of γ_i^k. Then, $prefix(\beta_i^k)$, does not preserve the order $\prec^{\alpha^k}$. However, this is not possible since, by definition, β_i^k is a causal view. Hence, we reach a contradiction.

Case 2: op and op′ have been issued by processes of $S^{\overline{k}}$. Since both operations are in γ_i^k, 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 $gate^k$. From Lemma 4, we have that prop(op) $\prec^{\alpha^k}$ 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^k. Then $prop(op')$ precedes $prop(op)$ in $prefix(\beta_i^k)$. However, this is not possible since, by definition, β_i^k is a causal view of computation α_i^k. Hence, we reach a contradiction.

Case 3: op has been issued by a process of $S^{\overline{k}}$ and op' has been issued by a process of S^k. Note that op must be a write operation, since γ_i^k only contains read operations from process i of system S^k. Operation op is propagated from $S^{\overline{k}}$ to S^k as described by the bridge algorithms, and the write operation performed by op appears in S^k as an operation $prop(op)$ issued by process $gate^k$. From Lemma 5, $prop(op) \prec^{\alpha^k} op'$.

Observe now that, by definition, operation $prop(op)$ in β_i^k is replaced by op to obtain γ_i^k. Then op' must precede $prop(op)$ in $prefix(\beta_i^k)$. However, this is not possible since, by definition, β_i^k is a causal view of computation α_i^k. Hence, we reach a contradiction.

Case 4: op has been issued by a process of S^k and op' has been issued by a process of $S^{\overline{k}}$. Note that op' must be a write operation, since γ_i^k only contains read operations from process i of system S^k. Operation op' is propagated from $S^{\overline{k}}$ to S^k as described by the bridge algorithms, and the write operation performed by op' appears in S^k as an operation $prop(op')$ issued by process $gate^k$. From Lemma 6, $op \prec^{\alpha^k} prop(op')$.

Observe now that, by definition, operation $prop(op')$ in β_i^k is replaced by op' to obtain γ_i^k. Then $prop(op')$ must precede op in $prefix(\beta_i^k)$. However, this is not possible since, by definition, β_i^k is a causal view of computation α_i^k. Hence, we reach a contradiction. □

LEMMA 9. *Each prefix of γ_i^k is serial.*

PROOF. Let us assume, by way of contradiction, that some prefix $prefix(\gamma_i^k)$ of γ_i^k is not serial. Let $prefix(\beta_i^k)$ be the corresponding prefix of β_i^k. By definition of serial computation, we must have one of the following two situations in $prefix(\gamma_i^k)$: (a) there is an operation $op = r(x)u$ and there is no $op' = w^k(x)u$ such that $op' \xrightarrow{\gamma_i^k} op$, or (b) there are such operations but there is also an operation $op'' = w(x)v$ such that $op' \xrightarrow{\gamma_i^k} op'' \xrightarrow{\gamma_i^k} op$. Note that, by definition of γ_i^k, the operations of $prefix(\gamma_i^k)$ and those of $prefix(\beta_i^k)$ read and write the same values in the same variables in the same order. Then, whichever of the two situations above occurs in $prefix(\gamma_i^k)$, also occurs in $prefix(\beta_i^k)$, and $prefix(\beta_i^k)$ cannot be serial. However, $prefix(\beta_i^k)$ is serial because β_i^k is the causal view of α_i^k, and all its prefixes are serial. Hence, we reach a contradiction and $prefix(\gamma_i^k)$ has to be serial. □

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

PROOF. To prove that S^T is causal, we need to show that each computation α^T is causal. To do so for a given computation α^T it is enough to show that, for each process i and each system S^k ($k \in \{0,1\}$, $i \neq gate^k$), there is a causal view (namely γ_i^k) of $\alpha_{i(k)}^T$.

Let α^T be a computation of S^T and let $\alpha_{i(k)}^T$ be obtained from α^T, for some $k \in \{0,1\}$ and some process i of system S^k, $i \neq gate^k$. From Lemma 7, γ_i^k, as defined in Definition 7, is a permutation of the operations in $\alpha_{i(k)}^T$. Also, from Lemma 8, each prefix of γ_i^k preserves the causal order $\prec^{\alpha^T}$. Finally, from Lemma 9, each prefix of γ_i^k is serial. Hence, from Definition 3, γ_i^k is a causal view of $\alpha_{i(k)}^T$.

Since this holds for each process $i \neq gate^k$ 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. □

Note that S^T is a propagation–based system, since S^0 and S^1 are also propagation–based systems and from the implementation of the bridge algorithms.

5. GENERALIZATION TO SEVERAL SYSTEMS

The following corollary shows that the bridge algorithms can be used to interconnect any number of systems to obtain a large causal system.

COROLLARY 1. *Let $S^0, S^1, ..., S^{n-1}$ be n propagation–based causal systems. They can be interconnected with the bridge algorithms to obtain a propagation–based system S^T causal.*

PROOF. We use induction on n to show the result. For $n = 1$ the claim is clearly true, since $S^T = S^0$. For $n = 2$ it is immediate from Theorem 1. Now, assume that we can obtain a propagation-based causal system S' by interconnecting the systems $S^0, S^1, ..., S^{n-2}$. Then, from Theorem 1, we can interconnect S' and S^{n-1} to obtain the propagation–based causal system S^T. □

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented algorithms for interconnecting causal systems. Such systems may have different implementations, as far as they are propagation–based. We have shown that those algorithms guarantee that the resulting system will remain causal. To the best of our knowledge, this has been the first work on devising algorithms to interconnect memory systems.

Several areas for future work are suggested by this work. The most important of them is studying the interconnection of systems with other consistency models and exploring the models of the resulting system. Currently, we are engaged in such a study.

7. REFERENCES

[1] S. Adve. *Designing Memory Consistency Models for Shared-Memory Multiprocessors.* PhD thesis, University of Wisconsin-Madison, 1993.

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

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

[4] V. Cholvi. Specification of the behavior of memory operations in distributed systems. *Parallel Processing Letters*, 8(4):589–598, December 1998.

[5] J. Goodman. Cache consistency and sequential consistency. Technical Report 61, IEEE Scalable Coherence Interface Working Group, March 1989.

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

[7] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1991.

[8] R. Lipton and J. Sandberg. PRAM: A scalable shared memory. Technical Report CS-TR-180-88, Princeton University, Department of Computer Science, September 1988.

[9] R. Prakash, M. Raynal, and M. Singhal. An adaptive causal ordering algorithm suited to mobile computing environments. *Journal of Parallel and Distributed Computing*, 41:190–204, 1997.

[10] M. Raynal and M. Ahamad. Exploiting write semantics in implementing partially replicated causal objects. In *Proceedings of the 6th EUROMICRO Conference on Parallel and Distributed Computing*, pages 157–163, Feb 1998.

[11] A. Singh. Bounded timestamps in process networks. *Parallel Processing Letters*, 6(2):259–264, 1996.

[12] H. Sinha. *Mermera: Non-Coherent Distributed Shared Memory for Parallel Computing.* PhD thesis, Computer Science Department, Boston University, April 1993.