Available online at www.sciencedirect.com
ScienceDirect
Information Processing Letters 105 (2008) 249–254
Information Processing Letters
www.elsevier.com/locate/ipl

# On the interconnection of message passing systems

A. Álvarez [a], S. Arévalo [b], V. Cholvi [c,*], A. Fernández [b], E. Jiménez [a]

[a] *Polytechnic University of Madrid, Spain*
[b] *Universidad Rey Juan Carlos, Spain*
[c] *Departamento de Lenguajes y Sistemas Informáticos, Universitat Jaume I, Campus de Riu Sec, 12071 Castellón, Spain*

Received 13 March 2007; received in revised form 18 July 2007; accepted 5 September 2007

Available online 19 September 2007

Communicated by A.A. Bertossi

**Abstract**

One of the most important abstractions for designing distributed programs is the *broadcast* facility. In this paper, we study the interconnection of distributed message passing systems. We have shown that totally ordered systems cannot be properly interconnected in any form. However, we have provided a simple protocol to properly interconnect FIFO ordered systems.


*Keywords:* Distributed systems; Interconnection networks; Formal methods; Design of algorithms

## 1. Introduction

One of the most important abstractions for designing distributed programs is the *broadcast* facility, with which a process sends a message to all the processes in the system. Such a facility provides one-to-all communication, and can be seen equivalent to sending point-to-point messages from the sender to the rest of processes. However, by itself, the above definition of broadcast does not impose any ordering restriction. This may lead to problems in its use since, for instance, messages may not be necessarily received in the order they were sent, or different processes may receive them in different order.

Then, typically, the above broadcast semantics is completed with restrictions on the order messages are delivered. The most popular ordering requirements imposed to broadcast primitives are the FIFO, the totally, and the causal orderings [1–8]. The first one requires that all messages sent by the same process are received in the order they were sent. The second one requires that all messages are received in the same order, irrespective of the sender. The third one enforces the receiving order of messages that are causally related [9].

In this paper, we study the interconnection of distributed message passing systems. By this, we mean the addition, to several existing message passing systems with a given ordering requirement, of a simple interconnection system to obtain a single system with the same ordering requirements as the original ones. There are mainly two reasons for interconnecting message passing 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.

* Corresponding author.
*E-mail address:* vcholvi@uji.es (V. Cholvi).

0020-0190/$ – see front matter 
doi:10.1016/j.ipl.2007.09.006

They can keep using their protocols at their local level.

- Second, depending on the network topology, it could be more efficient to implement several systems and interconnect them than having a single large system. An example of this would be a system that has to be implemented on two local area networks connected with a low-speed point-to-point link. 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 broadcast. 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, only one message crosses the link for each broadcast.

Some work has already been done regarding the interconnection of message passing systems [10–12]. However, all of these papers have just focussed on causally ordered systems.

Here, we extend these previous works to the case of systems that are either totally or FIFO ordered. We show that whereas totally ordered systems cannot be interconnected in the model of interconnection we consider, in the case of FIFO ordered systems interconnection is always possible. In this latter situation, we give a simple protocol to interconnect fully general heterogeneous systems, in contrast with the existing interconnecting protocols for causally ordered systems which are especially designed for concrete architectures as a mean of improving their performance. At this point, we note that since any causal system is also FIFO, and taking into account that causal ordered systems have been interconnected in the past [10–12], this implies that some particular FIFO systems (i.e., FIFO systems that are also causal) have been previously interconnected. In turn, here we show that any FIFO system, regardless if it is causal or not, can be interconnected. Finally, we would like to remark that our aim here is not centered on having a very efficient protocol, but on proving that in fact it is possible to interconnect FIFO systems.

The rest of the paper is organized as follows. In Section 2, we introduce our framework for the interconnection of message passing systems. In Section 3, we show the impossibility of interconnecting totally ordered systems. In Section 4, we study the interconnection of FIFO systems, and show how to interconnect them. Finally, in Section 5, we present some concluding remarks.

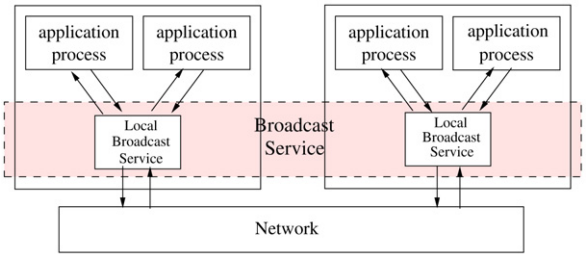


Fig. 1. System architecture.

## 2. Model and definitions

From a physical point of view, we consider distributed systems formed by a set of *nodes* connected by a *communication network*. The logical system we consider is formed by *processes* (executed in the nodes of the system) which interact by exchanging *messages* among them (using the communication network). The interface between the processes and the network has two types of events [1]: by using $bc\text{-}send_i(m)$, process $i$ broadcasts the message $m$ to all processes of the system. Similarly, by using $bc\text{-}recv_i(m, j)$, process $i$ receives the message $m$ from process $j$. Fig. 1 illustrates the above mentioned system architecture.

The basic broadcast service specification for $n$ processes consists of sequences of $bc\text{-}send_i$ and $bc\text{-}recv_i$ events, $0 \leqslant i \leqslant n-1$. In these sequences, each $bc\text{-}recv_i(m, j)$ event is mapped to an earlier $bc\text{-}send_j(m)$ event, every message received was previously sent, and every message that is sent is received once and only once at each process. For simplicity, we also assume that any given message is sent at most once. This assumption does not introduce any new restriction, since it can be forced by associating a (bounded) timestamp with every send operation [13].

Following, we define *totally ordered* and *FIFO ordered* systems, according to the ordering requirements of the broadcast services they implement.

**Definition 1.** We say that a system is *totally ordered* if for all messages $m_1$ and $m_2$ and all processes $p_i$ and $p_j$, if $m_1$ is received at $p_i$ before $m_2$, then $m_2$ is not received at $p_j$ before $m_1$.

**Definition 2.** We say that a system is *FIFO ordered* if for all messages $m_1$ and $m_2$ and all processes $p_i$ and $p_j$, if $p_i$ sends $m_1$ before it sends $m_2$, then $m_2$ is not received at $p_j$ before $m_1$.

We consider systems in which each message sent must be eventually received in every process of the sys-

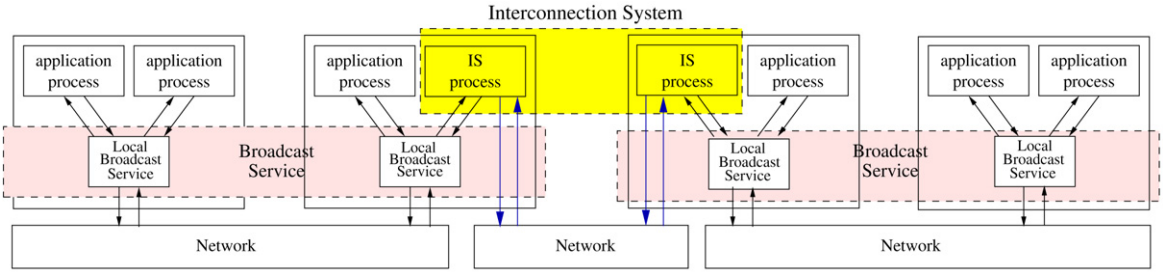


Fig. 2. Interconnection system.

tem. This is a very natural property (usually known as *Liveness*) which is preserved by every system that we have found in the literature. In our terminology it means that for each $bc\text{-}send_i(m)$ event, a $bc\text{-}recv_j(m,i)$ event will eventually occur for every process $j$ in the system.

Now, we define what we understand by *properly interconnecting* several totally/FIFO systems. Roughly speaking, it consists of interconnecting these systems (without modifying any of them) by using an *interconnection system* (denoted *IS*), so that the resulting system behaves as a single one and is also totally/FIFO ordered. Such an interconnection system is formed by a set of *interconnecting system processes* (denoted *IS processes*) that execute some distributed algorithm or protocol. Each of these processes is an application process of some of the original systems, and hence receives all the messages broadcast in that system and can itself broadcast new messages. In particular, the only way a value broadcasted by an application process in some system can be received by an application process in another system is if the interconnecting process of the latter system broadcasts it. The interconnecting processes can communicate among themselves via message passing. However, they cannot interfere with the protocol at their local original message passing system implementing the broadcast. Fig. 2 presents an example of an *IS* interconnecting two systems.

## 3. Impossibility of interconnecting totally ordered systems

In this section we show that totally ordered systems cannot be properly interconnected. To do so, we first show that in a totally ordered system, a message sent by a process $p_i$ cannot be locally delivered (i.e., to $p_i$ itself) in less time than the delay needed to transfer the message to any other process. Then, we use this result to show, by contradiction, the impossibility result.

Let us first consider a totally ordered system $S$, and assume that local computations take negligible (zero) time. Assume also that the delays of transferring messages between processes are in the range $[d', d]$, being $d'$ and $d$ two nonnegative constants such that $d' \leqslant d$. Let $t_{local}$ be the smallest delay since a process sends a message until the message is locally delivered. The following result gives a lower bound for $t_{local}$.

**Lemma 1.** *For any totally ordered system $S$ with at least two processes, we have that $t_{local} \geqslant d$.*

**Proof.** Assume the claim is not true and that messages are delivered locally in less than $d$ time (i.e., $t_{local} < d$). Consider a run in which some process $p_i$ of the system sends a message $m_1$ at time 0, and that some other process $p_j$ sends another message $m_2$ at the same time. Assume that the delay of both messages is $d$. Hence, $m_1$ is delivered to $p_i$ before receiving $m_2$, and $m_2$ is delivered to $p_j$ before receiving $m_1$. Since both messages must be eventually delivered, $m_1$ and $m_2$ will be received in different order in $p_i$ and in $p_j$. Consequently, by definition, the system is not totally ordered, reaching a contradiction. □

Now, by using the previous lemma, we obtain the following result.

**Theorem 1.** *Totally ordered systems cannot be properly interconnected.*

**Proof.** Consider a protocol that implements a totally ordered system in a single node (i.e., a single machine) but with, maybe, multiple processes. In this system, when a process executes *bc-send* operation, the protocol immediately delivers the messages to all processes of the node (by copying them into the processes receiving queues). This system has negligible time $t_{local}$, but since $d$ is also negligible, the previous lemma holds.

Let us now assume the existence of a protocol that properly interconnects several totally ordered systems. By definition, the resulting system must be totally ordered. However, if we use the above protocol to implement the systems to interconnect, one in each node, and

the network connecting the nodes has message delay $d > 0$, Lemma 1 is violated, since $t_{local} = 0$. Therefore, we reach a contradiction. □

## 4. Interconnection of FIFO ordered systems

In this section we show that, contrary to what happens with totally ordered systems, FIFO ordered systems can always be properly interconnected. First, we consider the case when there are only two systems. Later, we will consider the case of several systems.

Let us denote each of the FIFO ordered systems as $S^k$ (with $k \in \{0, 1\}$). The interconnecting protocol consists of two processes, denoted $isp^k$ (with $k \in \{0, 1\}$), that are part of each of the two systems. These interconnecting processes are only in charge of the interconnecting protocol. It is worthwhile to remark that each $isp^k$ is part of the system $S^k$ and, for that reason, can use the communication system implemented in $S^k$. Note also that the introduction of those processes does not require any modification of the original systems.

We consider that the set of processes of the resulting system $S^T$ includes all the processes in $S^0$ and $S^1$ combined, with the exception of $isp^0$ and $isp^1$, which are only used to interconnect $S^0$ and $S^1$.

Each $isp^k$ process executes two concurrent atomic tasks, $Propagate^k_{out}$ and $Propagate^k_{in}$ (atomicity is needed in order to avoid race conditions). $Propagate^k_{out}$ transfers messages issued in $S^k$ to $S^{\bar{k}}$ (we use $\bar{k}$ to denote $1 - k$), and $Propagate^k_{in}$ forwards within $S^k$ the messages transferred by $Propagate^{\bar{k}}_{out}$. Fig. 3 shows the implementation of the $Propagate^k_{in}$ and $Propagate^k_{out}$ tasks.

It must be noted that the link between $isp^0$ and $isp^1$ needs to be FIFO ordered. However, 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. A scheme of how the interconnecting protocol works is shown in Fig. 4.

The following theorem shows that the system $S^T$, obtained by connecting any two FIFO ordered systems $S^0$

| | |
|---|---|
| $Propagate^k_{out}(m)$ :: task which is | $Propagate^k_{in}(m)$ :: task which is |
| activated immediately after | activated immediately after |
| $bc\text{-}recv_{isp^k}(m, i)$ is executed | message $m$ is received from $isp^{\bar{k}}$ |
| begin | begin |
| if $m$ was not received from $isp^{\bar{k}}$ then | $bc\text{-}send_{isp^k}(m)$ |
| transfer $m$ to $isp^{\bar{k}}$ | end |
| end | |

Fig. 3. The interconnecting protocol in $isp^k$. Task $Propagate^k_{out}(m)$ is activated immediately after message $m$ is received in $isp^k$. As a result, it transfers such a message to $isp^{\bar{k}}$, but only if it was not received from $isp^{\bar{k}}$. This condition prevents messages going back and forth between $isp^k$ and $isp^{\bar{k}}$. On its turn, task $Propagate^k_{in}(m)$ is activated whenever the message $m$ is received from the process $isp^{\bar{k}}$. As a result, it issues a $bc\text{-}send_{isp^k}(m)$ operation, thus propagating the message $m$ to all processes within $S^k$.

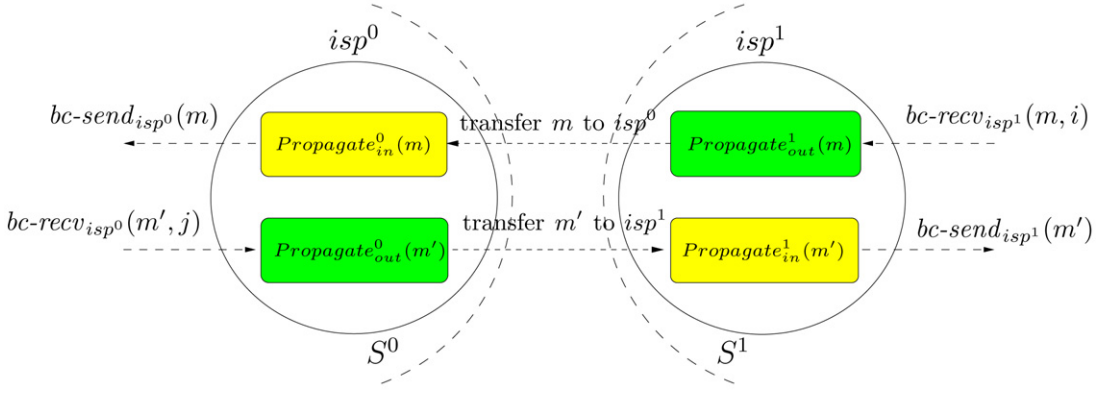


Fig. 4. Scheme of the interconnecting protocol.

and $S^1$ by using the above mentioned interconnecting protocol, is also FIFO ordered.

**Theorem 2.** *Any two FIFO ordered systems can be properly interconnected by using the protocol in Fig.* 3.

**Proof.** By contradiction. Assume there are two messages, $m_1$ and $m_2$, sent in that order by, say, process $p_i$ in system $S^0$. Now, assume they are received by, say, process $p_j$ in system $S^1$ in reverse order.

Since $S^1$ is a FIFO ordered system, $m_2$ must have been sent by $isp^1$ before $m_1$. Therefore, since the two systems are connected by a FIFO ordered communication channel, we have that $m_2$ must have been sent by $isp^0$ before $m_1$. This implies that, since $S^0$ is a FIFO ordered system, $m_2$ must have been sent (by $p_i$) before $m_1$. Thus, we reach a contradiction. □

Now, in the following corollary, we show that the same interconnecting protocol can be used to properly interconnect any number of FIFO ordered systems.

**Corollary 1.** *Let $S^0, S^1, \ldots, S^{n-1}$ be n FIFO ordered systems. They can be properly interconnected by using the protocol in Fig.* 3.

**Proof.** We use induction on $n$ to show the result. Let $S^T$ denote the resulting system. For $n = 1$ the claim is clearly true, since $S^T = S^0$. For $n = 2$ it is immediate from Theorem 2. Now, assume that we can obtain a FIFO ordered system $S'$ by properly interconnecting the systems $S^0, S^1, \ldots, S^{n-2}$. Then, from Theorem 2, we can properly interconnect $S'$ and $S^{n-1}$ to obtain a FIFO ordered system $S^T$. □

*Performance*. As it has been pointed in the Introduction, the aim of the proposed interconnecting protocol is not centered on efficiency, but on the fact that it is possible to interconnect FIFO systems. Anyhow, here we compare the performance of a system obtained using our interconnecting protocol with the performance of a system that directly uses a broadcast protocol connecting all the processes. We assume that the same broadcast protocol is used in the global system of reference and in each of the subsystems interconnected with our interconnecting protocol.

First, observe that our interconnecting protocol should not affect the *response time* a process observes when issuing a broadcast operation, since its broadcast protocol is not affected by the interconnection.

Regarding the *network traffic*, we assume that the broadcast protocol used generates one message per receiving process for each broadcast operation. Then, in a global system with $n$ processes, each broadcast operation generates $n-1$ messages. With our interconnection protocol $n+1$ messages are generated for two subsystems, since we add two interconnecting processes, and one message will be sent from one process to the other. Generalizing these results for $m$ subsystems, the number of messages for the interconnected system becomes $n+m-1$. Clearly, as $m$ increases, this could generate bottleneck problems.

Finally, we consider the *latency*, which is the time until a broadcast value is visible in any other process. For simplicity, we will discard here local computation times at the interconnecting processes. Then, if we have $m$ subsystems, a system running the basic protocol has latency $l$, the delay of a message between two interconnecting processes is $d$, and we interconnect the systems in a star fashion, the worst case latency is $3l + 2d$.

## 5. Concluding remarks

In this paper, we have studied the interconnection of distributed message passing systems that are either totally ordered or FIFO ordered. We have shown that totally ordered systems cannot be properly interconnected in any form. However, we have provided a simple protocol to properly interconnect FIFO ordered systems.

## References

[1] H. Attiya, J. Welch, Distributed Computing Fundamentals, Simulations and Advanced Topics, McGraw Hill, 1998.
[2] N.A. Lynch, Distributed Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.
[3] R. van Renesse, K.P. Birman, S. Maffeis, Horus: a flexible group communication system, Commun. ACM 39 (4) (1996) 76–83.
[4] P.M. Melliar-Smith, L.E. Moser, V. Agrawala, Broadcast protocols for distributed systems, IEEE Trans. Parallel Distrib. Syst. 1 (1) (1990) 17–25.
[5] M. Raynal, A. Schiper, S. Toueg, The causal ordering abstraction and a simple way to implement it, Inf. Process. Lett. 39 (6) (1991) 343–350.
[6] A. Schiper, K. Birman, P. Stephenson, Lightweight causal and atomic group multicast, ACM Trans. Comput. Syst. 9 (3) (1991) 272–314.
[7] H. Garcia-Molina, A. Spauster, Ordered and reliable multicast communication, ACM Trans. Comput. Syst. 9 (3) (1991) 242–271.
[8] M.F. Kaashoek, A.S. Tanenbaum, S.F. Hummel, An efficient reliable broadcast protocol, SIGOPS Oper. Syst. Rev. 23 (4) (1989) 5–19.
[9] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Commun. ACM 21 (7) (1978) 558–565.
[10] R. Baldoni, R. Beraldi, R. Friedman, R. van Renesse, The hierarchical daisy architecture for causal delivery, Distributed Systems Engineering 6 (2) (1999) 71–81.

[11] L.E.T. Rodrigues, P. Verissimo, Causal separators for large-scale multicast communication, in: ICDCS '95: Proceedings of the 15th International Conference on Distributed Computing Systems, IEEE Computer Society, Washington, DC, USA, 1995, p. 83.

[12] N. Adly, M. Nagi, Maintaining causal order in large scale distributed systems using a logical hierarchy, in: Proc. IASTED Int. Conf. on Applied Informatics, 1995, pp. 214–219.

[13] S. Haldar, P.M.B. Vitányi, Bounded concurrent timestamp systems using vector clocks, J. ACM 49 (1) (2002) 101–126.