Parallel Interconnection of Broadcast Systems with Multiple FIFO Channels

R. de Juan-Marín[(1)], V. Cholvi[(2)], E. Jiménez[(3)], F. D. Muñoz-Escoí[(1)]

[(1)]Inst. Tecnológico de Informática
Univ. Politécnica de Valencia
46022 Valencia (Spain)

[(2)]Depto. de Lenguajes y Sistemas Informáticos
Universitat Jaume I, Campus de Riu Sec
12071 Castellón (Spain)

[(3)]Escuela Universitaria de Informática
Univ. Politécnica de Madrid, Ctra. Valencia, km 7
28031 Madrid (Spain)

rjuan@iti.upv.es, vcholvi@uji.es, ernes@eui.upm.es, fmunyoz@iti.upv.es

Technical Report ITI-SIDI-2009/005

Parallel Interconnection of Broadcast Systems with Multiple FIFO Channels

R. de Juan-Marín[(1)], V. Cholvi[(2)], E. Jiménez[(3)], F. D. Muñoz-Escoí[(1)]

[(1)]Inst. Tecnológico de Informática
Univ. Politécnica de Valencia
46022 Valencia (Spain)

[(2)]Depto. de Lenguajes y Sistemas Informáticos
Universitat Jaume I, Campus de Riu Sec
12071 Castellón (Spain)

[(3)]Escuela Universitaria de Informática
Univ. Politécnica de Madrid, Ctra. Valencia, km 7
28031 Madrid (Spain)

Technical Report ITI-SIDI-2009/005

e-mail: rjuan@iti.upv.es, vcholvi@uji.es, ernes@eui.upm.es, fmunyoz@iti.upv.es

August 6, 2009

Abstract

This paper proposes new protocols for the interconnection of FIFO- and causal-ordered broadcast systems, thus increasing their scalability. They use several interconnection links between systems, which avoids bottleneck problems due to the network traffic, since messages are not forced to go throughout a single link but instead through the several links we establish. General architectures to interconnect FIFO- and causal-ordered systems are proposed. Failure management is also discussed and a performance analysis is given, detailing the benefits introduced by these interconnection approaches that are able to easily increase the resulting interconnection bandwidth.

1 Introduction

There have been multiple papers [1, 4, 10, 13, 7, 2] that had devoted their attention to the interconnection of message broadcast systems. Some of them [1, 4, 10, 13, 7] were focused on causal-ordered systems, thus reducing both the size of the vector clocks [15] being used in the broadcast protocols and the amount of needed messages (since smaller groups were used). Most of them have relied on either FIFO interconnection links [1, 10, 7] or on causal broadcast among the interconnection servers [4].

The aim of such solutions is to enhance the scalability of the resulting broadcast mechanisms. Such scalability might be needed in different current distributed applications, like P2P applications or the data centres being used to implement *cloud computing* systems.

Other scalability efforts have been focused on other aspects of causal communication, introducing some principles that have guided the design of the interconnection solutions. One example is the usage of causal separators [16] that divide the global system into causal zones (i.e., subgroups) and reduce the size of the vector clocks needed for guaranteeing causal delivery. Another example is the solution described in [11], that also interconnects previously existing systems and ensures causal delivery, but without requiring that all messages were broadcast; i.e., point-to-point communication among different systems is also considered. To this end, such global system also relies on a set of causal servers, each one from a different local system, and using vector clocks to ensure causal delivery in such set of servers, whilst system-local communication does only rely on linear logical clocks or on physical synchronisation.

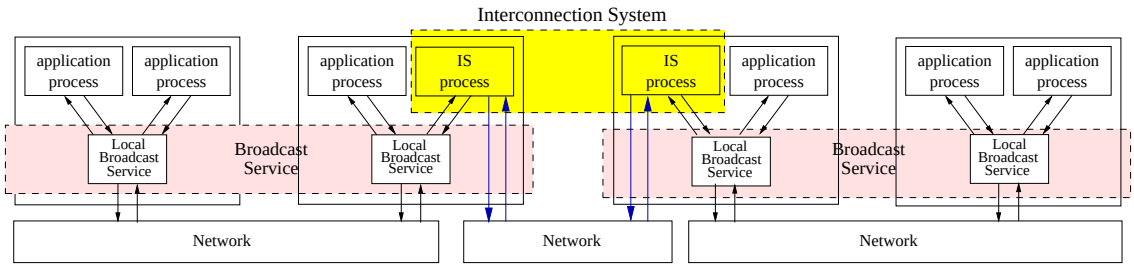

Figure 1: Interconnection System.

Similar efforts can be found in order to interconnect FIFO-ordered systems [10, 2], although in such case the interconnection is almost trivial, since it only depends on local information from the sender node.

However none of such papers has proposed any technique for increasing the usable bandwidth of such interconnecting protocols, implementing some technique for using simultaneously several interconnecting channels able to transmit multiple messages in parallel. Note that in most cases, each broadcast system is deployed over a very fast LAN, whilst the interconnecting links are far slower. In the common case, we might assume that such solution could be provided by the network layer, using multiple paths between each pair of interconnected servers, and selecting an appropriate path per message in order to avoid congestion. But this cannot be assumed in all scenarios. For instance, the set of data centres in a cloud computing environment might use dedicated inter-centre channels; i.e., there will be a single path between each pair of centres. Thus, we do not obtain any bandwidth improvement trying to set up multiple logical paths in such scenario. So, in some cases, a transport or application-level parallelisation of these interconnections might enhance the overall system performance. This paper scans this alternative, providing interesting results.

The rest of the paper is organised as follows. In Section 2, we introduce our framework for the interconnection of message-passing systems. In Section 3, we show how to interconnect FIFO-ordered systems by using several interconnection links between systems. In Section 4, we introduce the architecture with which interconnect FIFO-ordered systems. Sections 5 and 6 repeat the same for causal-ordered systems, whilst Section 7 describes how process failures can be managed. Finally, Section 8 provides a performance analysis and in Section 9, we present some concluding remarks.

2 Model

In this paper, we use a model similar to the one in [2]. From a physical point of view, we consider distributed systems made up of a set of *nodes* connected by a *communication network*. The logical system we consider consists of *processes* (executed in the nodes of the system) which interact by exchanging *messages* with one another (using the communication network). The interface between the processes and the network has two types of events [3]: by using *bc-send*$_i(m)$, process i broadcasts the message m to all processes of the system. Similarly, by using *bc-recv*$_i(m)$, process i receives the message m.

The basic broadcast service specification for n processes consists of sequences of *bc-send*$_i$ and *bc-recv*$_i$ events, $0 \leq i \leq n-1$. In these sequences, each *bc-recv*$_i(m)$ event is mapped to an earlier *bc-send*$_j(m)$ event, every message received was previously sent, and every message that is sent is received once and only once in each process. For the sake of simplicity, we also assume that any given message is sent once, at the most. This assumption does not introduce any new restriction, since it can be forced by associating a (bounded) timestamp with every send operation [9].

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

Definition 1. *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.*

The definition of *causally ordered* systems requires us to firstly introduce the *happens-before* (denoted

with $\rightarrow$) relation between messages. The important property of the happens-before relation is that it completely characterizes the causality relations between messages.

Given a sequence of *bc-send*$_i$ and *bc-recv*$_i$ events, $0 \leq i \leq n-1$, message m_1 is said to *happen-before* message m_2 if either:

1. The *bc-recv*$_i$ event for m_1 happens before the *bc-send*$_i$ event for m_2.
2. m_1 and m_2 are sent by the same process and m_1 is sent before m_2.

Now, we define a causally ordered system as follows.

Definition 2. *We say that a system is* causally ordered *if for all messages m_1 and m_2 and every process p_i, if m_1 happens-before m_2, then m_2 is not received at p_i before m_1 is.*

We consider systems in which each message sent must eventually be received in every process of the system. 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-send*$_i(m)$ event, a *bc-recv*$_j(m)$ event will eventually occur for every process j in the system.

Now, we define what we understand by *properly interconnecting* several equally ordered systems. Roughly speaking, this consists in 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 preserves the same ordering. Such an interconnection system is made up of a set of *interconnecting system processes* (denoted *IS processes*) that execute some distributed algorithm or protocol. Each of these processes receives all the messages broadcast in its system and can itself broadcast new messages received from the interconnection link, but it cannot generate and broadcast new messages on its own. More specifically, a value broadcast by an application process in some system can only be received by an application process in another system 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 in their original local message-passing system in any way. Figure 1 presents an example of an *IS* interconnecting two systems with the above-mentioned architecture and two *IS processes*.

3 Interconnection of FIFO-Ordered Systems

By using the model introduced in the previous section, [2] provided a simple protocol to properly interconnect FIFO-ordered systems. However, the aim of such a protocol was not focused on having a very efficient protocol, but on proving that it is in fact possible to interconnect FIFO systems. Therefore, to interconnect any pair of systems, the protocol used two *IS processes*. Clearly, this could generate bottleneck problems, since all messages must pass throughout the single link formed by this pair of *IS processes*. Thus, this raised the question as to whether it is possible or not to use several *IS processes* per interconnected system. In this section, we provide an interconnecting protocol for FIFO-ordered systems that uses several *IS processes* in each system.

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 several processes, denoted *isp*$_v^k$ (with $k \in \{0,1\}$ and v denoting the *IS process* within system S^k), that are part of each of the two systems. Note that the number of *IS processes* may be different in S^0 and in S^1.

These interconnecting processes are only in charge of the interconnecting protocol. It is worthwhile remarking that each *isp*$_v^k$ 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 in the resulting system S^T includes all the processes in S^0 and S^1, with the exception of the *IS processes*, which are only used to interconnect S^0 and S^1.

Each *isp*$_v^k$ process executes two concurrent atomic tasks, namely $Propagate\_out(isp_v^k, m)$ and $Propagate\_in(isp_v^k, m)$ (atomicity is needed in order to avoid race conditions).

- $Propagate\_out(isp_v^k, m)$ transfers the message m issued by a process in $set_w(isp_v^k)$ to $S^{\overline{k}}$ (we use $\overline{k}$ to denote $1-k$). Each process in system S^k (except for the *IS processes*) must be included in one transfer set (associated with only one *IS process*). Furthermore, the transfer of messages from processes in $set_w(isp_v^k)$ is performed to a single *IS process* in $S^{\overline{k}}$, denoted $link_w(isp_v^k)$. However, an *IS process* may transfer messages to many *IS processes* and receive transfers from many of them, but they are not necessarily the same.

  Both $set_w(isp_v^k)$ and $link_w(isp_v^k)$ are set up prior to running the protocol.

- $Propagate\_in(isp_v^k, m)$ forwards the messages received from $S^{\overline{k}}$ to within S^k. Note that when isp_v^k receives a transfer, it performs the broadcast to the whole set of processes in system S^k, regardless of the transfer sets these processes belong to.

$Propagate\_out(isp_v^k, m)$ :: task which is activated once *bc-recv*$_{isp_v^k}(m)$ is executed
begin
  if m was sent by a process in $set_w(isp_v^k)$
  then transfer m to $link_w(isp_v^k)$
end
	$Propagate\_in(isp_v^k, m)$ :: task which is activated immediately after message m is received from $S^{\overline{k}}$
	begin
	  bc-send$_{isp_v^k}(m)$
	end

Figure 2: The interconnecting protocol in isp_v^k

Fig. 2 shows the implementation of the $Propagate\_out(isp_v^k, m)$ and $Propagate\_in(isp_v^k, m)$ tasks.

It must be noted that the link between pairs of *IS processes*, one in each system, needs to be FIFO-ordered. However, nothing has been said about how to implement this. In a practical case, this channel could be implemented in a number of ways, either by using shared memory or by using message passing. Figure 3 shows an illustrative example of how transfer links are established between two interconnected FIFO systems. Each *IS process* isp_v^k is in charge of transferring the messages issued by processes in $set_w(isp_v^k)$ to system $S^{\overline{k}}$. There are three *IS processes* in system S^0 and two *IS processes* in system S^1. Both isp_1^0 and isp_2^0 transfer messages to isp_1^1, and isp_3^0 transfers messages to isp_2^1. In turn, isp_1^1 transfers messages to isp_1^0 and isp_2^1 transfers messages both to isp_2^0 and isp_3^0.

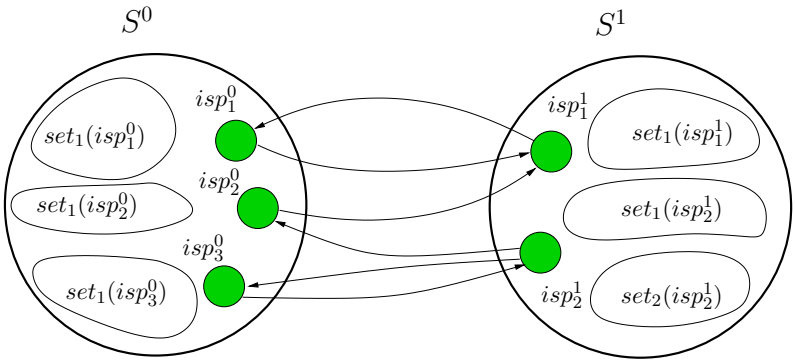

Figure 3: Example of the interconnecting protocol for two systems.

The following theorem shows that the system S^T, obtained by connecting any two FIFO-ordered systems S^0 and S^1 by using the above-mentioned interconnecting protocol, is also FIFO ordered.

Theorem 1. *Any two FIFO-ordered systems can be properly interconnected by using the protocol in Fig. 2.*

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 the reverse order.

Since S^1 is a FIFO-ordered system, m_2 must have been sent by some *IS process* in S^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 transferred by some *IS process* in S^0 before m_1. This implies that, since S^0 is a FIFO-ordered system system, m_2 must have been sent (by p_i) before m_1. Thus, we reach a contradiction. □

Note that the same interconnecting protocol can be used to properly interconnect any number of FIFO-ordered systems. This can be easily shown by induction on the number of systems. 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 1. Now, assume that we can obtain a FIFO-ordered system S' by properly interconnecting the systems $S^0, S^1, ..., S^{n-2}$. Then, from Theorem 1, we can properly interconnect S' and S^{n-1} to obtain a FIFO-ordered system S^T.

Similarly to what happened with the interconnection protocol proposed in [2], 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. The latency (i.e., the time until a broadcast value is visible in any other process) is also the same.

However and contrary to the interconnection protocol proposed in [2], we can now avoid bottleneck problems due to the *network traffic*, since messages are not forced to go through a single link but through the several links we establish.

4 An Architecture to Interconnect FIFO-Ordered Systems

In this section, we describe a general architecture to interconnect FIFO-ordered systems. Such an architecture can be built following these steps:

Step 1: For each process p in system S^k, choose an *IS process* in system S^k. Call such a process *isp*(p).

Step 2: For each *isp*(p), set up a series of paths to some *IS processes*, denoted *paths*(p). A *path* is formed by a series of subsequent FIFO-ordered links that connect a pair of *IS processes*. Such paths should have only one *IS process* per system they interconnect. Note that different paths (either from the same *IS process* or not) may share some of their links.

Step 3: Transfer the messages issued by process p (to other systems) by using *isp*(p) through *paths*(p).

Step 4: When an *IS process* receives a transfer, it broadcasts that message to every process within its own system.

The correctness proof of the above-mentioned architecture is very similar to the proof of Theorem 1 (only S^0 and S^1 must be changed by two arbitrary pairs of systems, say S^k and $S^{k'}$), and we omit it here.

Note that the protocol proposed in the previous section fits into the proposed architecture. However, other interconnection protocols that adhere to the proposed architecture could be implemented. Fig. 4 shows an illustrative example with four systems and three different ways of interconnecting them. In the example, we show the case where three *IS processes* in system S^0 (denoted isp_1^0, isp_2^0 and isp_3^0) are respectively used to transfer the messages issued by processes p, q and r in S^0 (i.e., *isp*$(p) = isp_1^0$, *isp*$(q) = isp_2^0$ and *isp*$(r) = isp_3^0$). As can be seen, isp_1^0 sets up three links directly to isp_1^1, isp_1^2 and isp_2^3 (in red). Moreover, isp_2^0 establishes a link to isp_1^2; then, this one establishes another link to isp_2^1, and this one to isp_2^3 (in black). Finally, isp_3^0 establishes a link to isp_1^3; then, this one establishes two links, one to isp_1^2 and another one to isp_2^1 (in blue).

5 Interconnection of Causal-Ordered Systems

Contrary to what happens with totally ordered systems, that can not be interconnected in any way [2][1], and similar to what happens with FIFO ordered systems, causally ordered systems can always be prop-

[1] This result does not contradict the *FIFO forwarding* theorem of [10] that states that total order systems can be interconnected with a FIFO total order interconnecting protocol, since such a protocol needs to be intrusive; i.e., it needs to modify the regular behaviour of the local total order protocol in each system, and such degree of intrusiveness is not allowed in our system model assumptions.

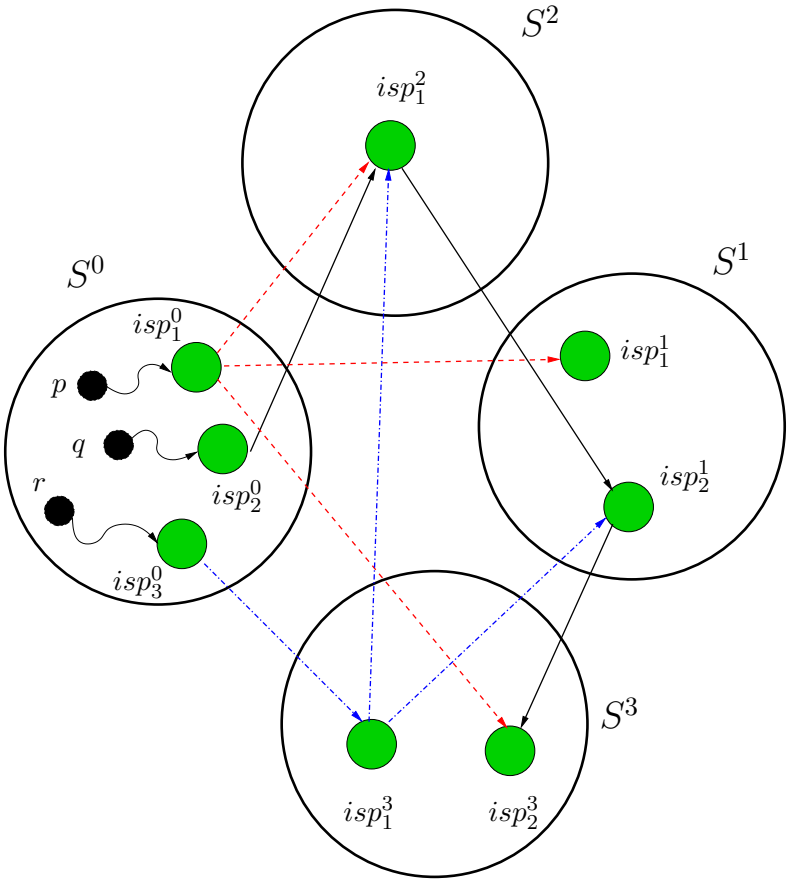

Figure 4: Example of the architecture to interconnect FIFO-ordered systems.

erly interconnected. As in the case of FIFO ordered systems, in order to avoid bottleneck problems it would be interesting to design an interconnecting protocol that uses several *IS processes* at each system. Unfortunately, the next theorem shows than in causally ordered systems this is not possible.

Theorem 2. *Any two causally ordered systems cannot be properly interconnected if there is more than one IS process at each system, and such IS processes are not coordinated.*

Proof. By contradiction. Let us now assume the existence of a protocol that properly interconnects two causally ordered systems S^0 and S^1 such that there are two IS processes isp_1^0 and isp_2^0 in S^0 (not coordinated in any form) and two IS process isp_1^1 and isp_2^1 in S^1 (not coordinated in any form).

Assume that a process i in S^0 issues message m. Let isp_1^0 the *IS process* that will transfer such message to isp_1^1. Now, consider a process j in S^0 that receives message m and after it, issues message m' that will be transferred, by means of isp_2^0, to isp_2^1.

It could happen that isp_2^0 transfers message m' to isp_2^1 before isp_1^0 transfers message m to isp_1^1. If isp_2^1 sends message m' before isp_1^1 sends message m, it could happen that some process in S^1 receives m' before m. This breaks causality and we reach a contradiction. □

As a consequence of this theorem, if we want to interconnect causally ordered systems we are forced to use only one *IS process* per system (there are multiple samples of such protocols [10, 4, 13, 11, 2]), or to coordinate in some way such *IS processes*. Let us explore this second alternative.

As shown in the proof of Theorem 2, when multiple interconnection links are used, we need to guarantee that causally related messages are delivered in the destination system in the appropriate order. Concurrent messages do not introduce any problem, they can be delivered without any constraint. At a glance, the

resulting interconnecting protocol should take care of ensuring an appropriate delivery order for causally related messages.

Most causally ordered interconnection protocols based on a single *IS process* per system simply relied on a single FIFO link in order to implement the interconnection [10, 4, 2] of two causal systems. If multiple *IS processes* are used, with multiple interconnection links, we might ensure that all such links deliver all forwarded messages in a global FIFO order (i.e., the messages are delivered in the receiver system in the same order they were sent from the sender system). This trivially ensures that the semantics of Theorem 2 are maintained, and also complies with the *FIFO forwarding* theorem of [10].

In order to comply with this requirement, the interconnection protocol presented in Section 3 is taken as a basis and is extended in the following way:

1. In each system S^k, one of its *IS processes* is selected as a sequencer with a deterministic criterion; e.g., that with the lowest node identifier. Let us name isp^k_{seq} such *IS process*. It maintains the number of broadcast messages, in a local variable seq_num, and it will assign a sequence number to each message broadcast by such system. This follows the same principle described in [5] in order to implement the Isis ABCAST protocol (with causal total order guarantees) on top of its CBCAST one (with reliable causal delivery). But there is a big difference in our approach. We do not want to extend the underlying causal broadcast protocol being used in the local system. Instead of this, we will tag with such sequence numbers the messages being forwarded by the *IS processes*; i.e., such sequence numbers are internally maintained in the interconnecting protocol, and they are completely unknown in the causal broadcast protocols being used in each interconnected system.

2. When a message m sent by any process p^k_i of the local system is delivered in its associated *IS process* (i.e., $isp(p^k_i)$), such $isp(p^k_i)$ waits until isp^k_{seq} sends to $isp(p^k_i)$ the appropriate sequence number for m; i.e., $sn(m)$. Once $sn(m)$ is known, $isp(p^k_i)$ sends $\langle m, sn(m)\rangle$ through its interconnection link to $S^{\overline{k}}$.

3. In the system $S^{\overline{k}}$ that plays the receiver role for m, all *IS processes* also need to maintain a local variable $received$ that accumulates the amount of received messages from S^k. To this end, $received$ was initialised to zero, and it is increased each time a message being sent by any $isp^{\overline{k}}_x$ is delivered.

4. Once $\langle m, sn(m)\rangle$ is received by its associated $isp^{\overline{k}}_j$, such process will causally broadcast m in $S^{\overline{k}}$ as soon as $sn(m) = received + 1$ holds in $isp^{\overline{k}}_j$. This ensures FIFO global delivery of all messages forwarded through all interconnection links between S^k and $S^{\overline{k}}$, but they can be propagated in a parallel way, so the bandwidth of such system interconnection can be greatly enhanced.

The resulting interconnecting protocol is summarised in Figure 5, as a set of four atomic concurrent tasks.

$Sequence\_out(isp^k_{seq}, m)$ :: task which is activated once *bc-recv*$_{isp^k_{seq}}(m)$ is executed begin $sn(m) = ++seq\_num$ send $\langle id(m), sn(m)\rangle$ to $isp(sender(m))$ end	$Receive(isp^k_v, m)$ :: task activated once m is received from any isp^k_j begin $received$++ end
$Propagate\_out(isp^k_v, m)$ :: task which is activated once *bc-recv*$_{isp^k_v}(m)$ is executed begin if m was sent by a process in *set*$_w(isp^k_v)$ then wait for receiving $\langle id(m), sn(m)\rangle$ transfer $\langle m, sn(m)\rangle$ to *link*$_w(isp^k_v)$ end	$Propagate\_in(isp^k_v, \langle m, sn(m)\rangle)$ :: task activated once message $\langle m, sn(m)\rangle$ is received from $S^{\overline{k}}$ begin wait until $sn(m) =$*received*$+1$ *bc-send*$_{isp^k_v}(m)$ end

Figure 5: The interconnecting protocol in S^k

The following theorem shows that the system S^T, obtained by connecting any two causal-ordered systems S^0 and S^1 by using this interconnecting protocol is also causal-ordered.

Theorem 3. *Any two causal-ordered systems can be properly interconnected by using the protocol in Fig. 5.*

Proof. By contradiction. Assume there are two messages m_1 and m_2 sent in system S^0 and verifying that $m_1 \rightarrow m_2$. Now, assume they are received by, say, process p_j in system S^1 in the order $m_2 < m_1$.

Due to task $Sequence\_out$ in S^0, it is guaranteed by the interconnecting protocol that if $m_1 \rightarrow m_2$ then $sn(m_1) < sn(m_2)$. Due to tasks $Receive$ and $Propagate\_in$ in S^1, no process in S^1 will be able to deliver m_2 before m_1 since $Propagate\_in$ compels that the isp_v^1 process that receives m_2 does not broadcast such message in S^1 until it has delivered m_1 (due to their sequence numbers order, and the management of variable $received$ in both tasks). If so happens, $m_1 \rightarrow m_2$ also holds in S^1 and no process can break such causal order, since it is assumed that S^1 is a causal broadcast system. As a result, all S^1 processes deliver m_1 before delivering m_2 and this raises a contradiction with the assumption given in the previous paragraph, proving thus the theorem. □

6 An Architecture to Interconnect Causal-Ordered Systems

The interconnecting protocol presented in Sect. 5 is able to interconnect two causal systems. Such interconnection mechanism could be easily extended to multiple causal systems. Thus, if there is a set of causal systems $\{S^o, S^1, ..., S^{n-1}\}$ that have been interconnected in order to achieve a global causal-order system S^T, then we can interconnect another system S^n by setting one or several interconnecting links between itself and one of the systems that belong to S^T. Note that in case of setting multiple links, such links can not be set with different S_i, S_j systems of S^T since this will define cycles in the resulting global system. When a cycle exists, there will be at least two different paths for connecting two different nodes (i.e., causal-ordered subsystems) in such global system. If two paths are available in order to interconnect two different systems S^n and S^j, Theorem 2 arises again, and the implicit coordination between the *IS processes* chosen in the sending system (e.g., S^n) disappears, since each message travels along a different path and the sequence numbers of two causally related messages in S^n is not maintained in the receiver system S^j. Note that in order to preserve such dependencies in the sequence numbers all messages should be forwarded along the same path.

These constraints can be formalised in the following Lemma and Theorem.

Lemma 1. *Given a set of N causal systems $S^T = \{S^o, S^1, S^2, ..., S^{n-1}\}$ interconnected in pairs with the interconnecting protocol shown in Fig. 5, S^T is properly interconnected if it does not contain any link cycle.*

Proof. By contradiction. Let us assume that S^T is a causal-ordered global system and that there exists a cycle in S^T, and that at least two of its systems S^i and S^j belong to such cycle. If so happens, there are at least two different FIFO paths $path_1$ and $path_2$ for interconnecting S^i and S^j. At least one of such paths has a length greater than one link. Note that otherwise both paths would have been the same (a single link connecting directly S^i and S^j).

Let us assume that processes in S^i have sent two different messages m_1 and m_2 causally related in the following way $m_1 \rightarrow m_2$. So, m_1 is forwarded to S^j before m_2, but using a different path. For instance, m_1 was forwarded along $path_1$ whilst m_2 was along $path_2$. Since both paths are FIFO ordered, but they are not coordinated in order to ensure a global causal order (recall that such coordination was ensured for a single link, but not for different paths of multiple links that have traversed through different systems), it is possible that m_2 be delivered in S^j before m_1 is delivered. This breaks the causal order, and contradicts the initial assumption of S^T being a causal global system. Thus, this proves the lemma.

□

Theorem 4. *n causal-ordered systems $S^0, S^1, ..., S^{n-1}$, can be pair-wise interconnected with our causal IS protocol to obtain a system S^T that is also causal-ordered.*

Proof. We use induction on n to show the result. For $n = 1$ the claim is trivially true. Then, if we have a causal-ordered system S' by interconnecting systems $S^0, S^1, ..., S^{n-2}$, then we can interconnect S' and S^{n-1} in a pair-wise manner following both Theorem 3 and Lemma 1, and the resulting system S^T is causal-ordered, proving this theorem. □

7 Fault Tolerance

The interconnecting protocols previously outlined can easily tolerate failures. Note that most recoverable applications (e.g., replicated databases [12]) demand *uniform* [8], *stable* [5] or *safe* [6] delivery for broadcast messages. This means that a message is not delivered until the group communication system can ensure that it has been received by all message target processes[2]. Moreover, such message can not be "*garbage recycled*" in the sender process until such stable delivery is ensured. In [4], such uniform delivery is also taken as the key principle in order to achieve fault tolerance.

The rules to follow are these:

- An *IS process* isp_v^i does not report the uniform delivery of a message m broadcast in its system S^i until it gets a uniform confirmation from all other *IS processes* isp_w^j to which it previously forwarded m.
- A message m that has been forwarded to a system S^j is reported as uniform in such system S^j following the regular protocol being used in S^j. This means that the receiving *IS process* isp_w^j knows about such message uniformity at that time, and needs to report such issue to its sender isp_v^i once this step is completed.
- If such receiving isp_w^j fails, it will be replaced by another process in S^j that will play such isp role. Two different scenarios arise:
  - The old isp_w^j was able to broadcast all messages received from isp_v^i. If so happens, the new isp_w^j will be able to report such messages as uniform, using the regular protocols of S^j. No problem arises in this case.
  - The old isp_w^j failed before being able to broadcast all messages received from isp_v^i. If so happens, system S^j does not know anything about such messages and the new isp_w^j will be unable to report any of such messages as uniformly delivered. If so happens, isp_v^i will forward again such unreported messages to (the new) isp_w^j process, once a given time-out is exhausted. Moreover, in case of interconnecting causal systems, the other *IS processes* in S^j will be blocked waiting for the delivery of those missed messages, and they will be able to tell such new isp_w^j which were the $sn(m)$ of such messages. So, isp_w^j will be able to ask isp_v^i for the messages associated to those $sn(m)$.

  - If one of the forwarding isp_v^i nodes crashes, a new isp_v^i process will be created. If there were some forwarded messages not yet reported as uniform, such new isp_v^i process knows which were such messages and it forwards them again and resets their time-outs. So, such messages are appropriately managed.

These rules avoid any message loss, and uniform delivery ensures that all broadcast messages are eventually delivered to all their destination processes. So, node failures are easily overcome.

8 Performance Analysis for Causal-Ordered Systems

The usage of FIFO interconnecting links in order to implement interconnection protocols for two causal and/or FIFO broadcast systems have been previously proposed in several papers [10, 13, 2]. Note also that

[2] In modern systems, this constraint is relaxed: the message can be delivered as soon as it is received and complies with the intended order semantics. Later on a uniform/stable/safe notification is delivered to the receiver process, indicating that such message delivery is already uniform/stable/safe.

the daisy architecture described in [4] also becomes a single FIFO link when only two causal systems need to be interconnected.

However, none of such papers explored the alternative of using more than one *IS process* per system. In most cases, the intra-system links will be far more efficient than the inter-system ones. Imagine, for instance that each system is deployed in a given laboratory or enterprise site, using a fast LAN (e.g., SCI has a bandwidth of 20 Gbps, and there are also 10Gb Ethernet LANs nowadays, with delays far below one ms in both cases), whilst inter-system links might have the regular bandwidth and delays of a WAN (less than 100 Mbps of bandwidth and more than 50 ms of delay, in most cases). As a result, the interconnecting protocols and links could be easily overloaded using a negligible workload in the systems being interconnected, since the latter are two orders of magnitude faster than the former. So, the parallelisation of such interconnections is able to multiply the resulting bandwidth without increasing the transmission delays. Such benefit is directly applicable to the protocol described in Sect. 3. It does not need any further analysis, since the interconnecting protocol does not demand any synchronisation among the *IS processes* of the systems being interconnected.

Let us concentrate in the analysis of the causal interconnection protocol described in Section 5, assuming that only two causal-ordered systems need to be interconnected. To this end, the following parameters are needed:

- *n_isps*: Number of *IS processes* in the sender system; i.e., number of interconnection links being used. Since there is a single sequencer process needed for synchronising all *IS processes* of such sender system, and such synchronisation requires a single additional message in some cases (when the *IS process* that forwards the message is not the sequencer), we need this parameter in order to set the probability of requiring such extra message.
- *ibw*: Intra-system bandwidth (in Mbps).
- *id*: Intra-system message transmission delay (in seconds).
- *sr*: The average sending rate at one broadcast system; i.e., the number of messages sent per time unit (in seconds).
- *ms*: Message size, in Mb (megabits). So, the product $sr \times ms$ is an expression that provides the required bandwidth (in Mbps). So, the following should be ensured:

$$sr < \frac{ibw}{ms} \tag{1}$$

- *ebw*: Inter-system link bandwidth (in Mbps). This parameter sets an important constraint in the global system, since the interconnection will usually be the bottleneck of such system. Such constraint is:

$$sr < \frac{ebw \times n\_isps}{ms} \tag{2}$$

- *ed*: Inter-system message transmission delay (in seconds).

Using such parameters, sections 8.1 and 8.2 evaluate the optimal number of interconnecting links and compare the performance of our parallelised interconnection with the daisy architecture proposed in [4], respectively.

8.1 Optimal Number of Links

In order to find out which is the optimal number of interconnecting links, let us explore the time needed for broadcasting a message in the whole system and how such time depends on the number of links. Thus, the implementation of a global causal-ordered broadcast only needs a system-local reliable broadcast protocol complemented with tagging all messages with vector clocks and considering such clocks in the delivery

step. Such kind of protocol [5] can be implemented using a single round of messages; i.e., if there are n nodes in a system, only $n-1$ messages are needed.

Once each message is delivered in its associated *IS process*, such process needs to wait for the sequence number that should tag such message. This sequence number is sent by the sequencer process using a point-to-point message. Note, however, that the sequencer is also an *IS process*. So, this additional message is only needed with a probability of $1 - \frac{1}{n\_isps}$. Moreover, such message is smaller than all other messages considered in the next expressions, requiring thus a smaller transmission time. However, we have not considered such issue in those expressions.

Later, the message is forwarded through the interconnection link and re-broadcast in the receiving system. This implies, again, a single round of messages. So, if no additional workload is considered, the minimal time needed for receiving a message broadcast by system S_i in a node of system S_j ($trans\_time$) is:

$$trans\_time = \left(3 - \frac{1}{n\_isps}\right) \times \left(id + \frac{ms}{ibw}\right) + \left(ed + \frac{ms}{ebw}\right) \tag{3}$$

Thus, with a negligible workload, the optimal number of *IS processes* per system is 1, since this eliminates the need of a synchronisation (intra-system) message (i.e., that carrying the sequence number), as it can be seen in expression (3).

However, when workload is considered, we could use a queueing model [14] in order to represent such system, but as it is already mentioned in [14, Chapter 5], in a queueing network we only need to identify its bottleneck centre in order to set upper bounds to the global system throughput. In our system, and with the assumptions given above, such bottleneck centre is the interconnecting channel. In the best case, the bandwidth of the interconnecting links could be the same as the internal bandwidth of each interconnected system; i.e., constraints (1) and (2) generate the same thresholds when n_isps is 1. If so happens, the optimal number of sender *IS processes* will be one, as seen above in (3). Otherwise, we need to set as many interconnecting links as given by expression (4):

$$n\_isps = \left\lceil \frac{ibw}{ebw} \right\rceil \tag{4}$$

Note that both id and ed are modeled as "*delay centres*" [14]; i.e., a link can be shared by multiple messages being forwarded along it, and we do not need a queue in order to model such delay. As a result, they do not appear in expression (4), where only queueing servers need to be considered. Indeed, such queueing centres are modelling the bandwidth of each kind of link. They have a service demand of $\frac{ms}{ibw}$ seconds in the intra-system communications and $\frac{ms}{ebw}$ seconds in the inter-system links. This explains how expression (4) is derived: its target is to balance the serving time of both kinds of servers, and this can be achieved increasing the number of interconnecting links.

8.2 Comparison with Other Solutions

To our knowledge, no other paper has proposed a parallelised interconnection of causal-ordered broadcast systems. However, the technique described in [4] can be considered as a close approach. Despite proposing a single interconnecting server for each existing system, it recommends that systems were split into multiple subsystems when they have grown excessively. So, this is an indirect way of introducing multiple interconnecting servers in each original system. Moreover, this provides the advantage of reducing the size of the vector clocks being used in each subsystem for their local broadcasts. In order to implement the global interconnection, the daisy architecture builds an upper-layer causal-ordered system composed by the interconnecting servers of all broadcast systems. Each time a message is broadcast in one of the systems, its interconnecting server re-broadcasts such message to all other interconnecting servers, who broadcast again such message to all nodes in their respective systems. So, such global broadcast consists of three different causal broadcast interactions.

Let us compare the daisy architecture with our solution described in Sect. 5. To this end, let us assume a global system where there are initially two causal-ordered broadcast systems S^0 and S^1, with $3n$ nodes each one. The intra-system bandwidth and link delays are ibw and id, respectively, whilst the inter-system

bandwidth and link delays are ebw and ed, respectively. We also assume that $ibw > ebw$ and $id < ed$. Due to the size of such systems, each of them has been divided into three sets of n nodes per set using our approach, or into three separate new systems (S^{00}, S^{01}, S^{02}, and S^{10}, S^{11}, S^{12}) using the daisy architecture, again with n nodes per system.

In such scenario, the broadcast of a message m sent in S^0 with our solution implies:

- $3n - 1$ messages in order to broadcast such message into S^0.
- One additional message (in the worst case) in order to assign a sequence number to such message and notify it to the associated *IS process*.
- One inter-system message through the interconnecting link.
- $3n - 1$ messages in order to broadcast m into S^1.

Globally, this has required $6n - 1$ messages (that might be $6n - 2$ if the isp being used in S^0 is also the sequencer process) transmitted through intra-system links and one single message traversing inter-system (and slower) links. Additionally, it has required three hops in the best case, or four, in the worst one.

On the other hand, with the daisy architecture, such broadcast needs these messages:

- Let us assume that the sender of message m belongs to S^{00}. It requires $n - 1$ messages in order to broadcast m into such system.
- As a result, m can be broadcast in the system composed by all interconnecting servers of the six newly created broadcast systems. Five messages are needed to this end. Two of such messages forward m to other systems that initially belonged to S^0. So, they are fast messages. On the other hand, the other three need to use the assumed slow interconnecting links.
- Into each system, $n - 1$ messages are needed to locally re-broadcast m. Since there are five systems of this kind, $5n - 5$ messages are needed in this step.

At the end, this architecture needs the same global amount of point-to-point messages; i.e., $6n - 1$ messages. But in our approach, only one of such messages need to use the slow links, whilst in the daisy architecture, three messages have been forwarded through such links. This implies that with additional workload, such slow interconnecting links will be saturated sooner using the daisy approach. If the difference between the original intra-system and inter-system bandwidths is important, our solution guarantees better scalability than a daisy architecture.

On the other hand, the daisy architecture needs only three logical hops to broadcast a message between different systems, whilst our approach might need four logical hops in some cases. However, using our solution only one hop is needed into each of the initial systems S^0 and S^1 in order to locally broadcast a given message, whilst the daisy architecture introduces also three hops between processes located in different parts of such original systems.

9 Conclusions

In this paper, we have studied the interconnection of broadcast systems that are either FIFO or causally ordered. We have provided interconnection protocols that can use several interconnection links between systems, which avoid bottleneck problems due to the network traffic, since messages are not forced to go through a single link but throughout the several links we establish. Furthermore, we have proposed a general architecture with which to interconnect multiple broadcast systems. The usage of multiple interconnection links is specially convenient when scalability is a must and such interconnection links provide a limited bandwidth (compared to that of intra-system links).

References

[1] N. Adly and M. Nagi. Maintaining causal order in large scale distributed systems using a logical hierarchy. In *Proc. IASTED Int. Conf. on Applied Informatics*, pages 214–219, 1995.

[2] Ángel Álvarez, Sergio Arévalo, Vicent Cholvi, Ernesto Jiménez, and Antonio Fernández. On the interconnection of message passing systems. *Information Processing Letters*, 105(6):249–254, 2008.

[3] Hagit Attiya and Jennifer Welch. *Distributed Computing Fundamentals, Simulations and Advanced Topics*. McGraw Hill, 1998.

[4] Roberto Baldoni, Roberto Beraldi, Roy Friedman, and Robbert van Renesse. The hierarchical daisy architecture for causal delivery. *Distributed Systems Engineering*, 6(2):71–81, 1999.

[5] Kenneth P. Birman, André Schiper, and Pat Stephenson. Lightweight causal and atomic group multicast. *ACM Trans. Comput. Syst.*, 9(3):272–314, 1991.

[6] Gregory Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[7] Antonio Fernández, Ernesto Jiménez, and Vicent Cholvi. On the interconnection of causal memory systems. *J. Parallel Distrib. Comput.*, 64(4):498–506, 2004.

[8] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. ACM Press, 2nd edition, 1993.

[9] Sibsankar Haldar and Paul M. B. Vitányi. Bounded concurrent timestamp systems using vector clocks. *Journal of the ACM*, 49(1):101–126, 2002.

[10] Scott Johnson, Farnam Jahanian, and Jigney Shah. The inter-group router approach to scalable group composition. In *Intnl. Conf. on Distr. Comp. Syst. (ICDCS)*, pages 4–14, Austin, TX, USA, June 1999. IEEE-CS Press.

[11] Satoshi Kawanami, Tomoya Enokido, and Makoto Takizawa. A group communication protocol for scalable causal ordering. In *18th Intnl. Conf. on Adv. Inform. Netw. and Appl. (AINA)*, pages 296–302, Fukuoka, Japan, March 2004. IEEE-CS Press.

[12] Bettina Kemme, Alberto Bartoli, and Özalp Babaoglu. Online reconfiguration in replicated databases based on group communication. In *Intnl. Conf. on Dependable Systems and Networks (DSN)*, pages 117–130, Göteborg, Sweden, July 2001. IEEE-CS Press.

[13] Philippe Laumay, Eric Bruneton, Noel De Palma, and Sacha Krakowiak. Preserving causality in a scalable message-oriented middleware. In *IFIP/ACM Intnl. Conf. on Distr. Syst. Platforms (Middleware)*, pages 311–328, Heidelberg, Germany, November 2001.

[14] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984.

[15] Friedemann Mattern. Virtual time and global states of distributed systems. In M. Cosnard et al., editor, *Proc. Workshop on Parallel and Distributed Algorithms*, pages 215–226, North-Holland / Elsevier, 1989. (Reprinted in: Z. Yang, T.A. Marsland (Eds.), "Global States and Time in Distributed Systems", IEEE, 1994, pp. 123-133.).

[16] Luís Rodrigues and Paulo Veríssimo. Causal separators for large-scale multicast communication. In *Intnl. Conf. on Distr. Comp. Syst. (ICDCS)*, pages 83–91, Vancouver, Canada, May 1995. IEEE-CS Press.