Preserving Causality in a Scalable Message-Oriented Middleware

Philippe Laumay, Eric Bruneton, Noël De Palma, and Sacha Krakowiak

Laboratoire Sirac**,
INRIA Rhône-Alpes ; 655, Avenue de l'Europe ; 38334 Saint-Ismier Cedex ; France
Tel : +33 4 76 61 53 89 - Fax : +33 4 76 61 52 52
Philippe.Laumay@inrialpes.fr

Abstract. We present a solution to guarantee scalable causal ordering through matrix clocks in Message Oriented Middleware (MOM). This solution is based on a decomposition of the MOM in domains of causality, i.e. small groups of servers interconnected by router servers. We prove that, provided the domain interconnection graph has no cycles, global causal order on message delivery is guaranteed through purely local order (within domains). This allows the cost of matrix clocks maintenance to be kept linear, instead of quadratic, in the size of the application. We have implemented this algorithm in a MOM, and the performance measurements confirm the predictions.

1 Introduction

Message-oriented middleware (MOM) is growing in importance with the development of new applications made of loosely coupled autonomous components that communicate on large-scale networks. This class of applications (including stock exchange quotations, electronic commerce services, web-content provisioning) is characterized by asynchronous exchanges, heterogeneity, and requirements for scalability and evolution. MOM provides an infrastructure that transmits messages and events to the widely spread components of a service, gluing them together through logical coupling [1].

Asynchronous interaction over large-scale networks is a major source of non-determinism. Message ordering provides a way of reducing this non-determinism, as it allows the application designer to assert properties such as causal delivery of messages or atomic (uniform) broadcast. Industrial specifications and implementations of MOM are now integrating this aspect. For example, the CORBA Messaging reference specification [2] defines the ordering policy as part of the messaging Quality of Service.

However, usual message ordering mechanisms pose scalability problems, as an increase of the number of nodes and/or the distance between them degrades

** Sirac is a joint laboratory of Institut National Polytechnique de Grenoble, INRIA and Université Joseph Fourier.

R. Guerraoui (Ed.): Middleware 2001, LNCS 2218, pp. 311–328, 2001.

the performance of classical clock-synchronization algorithms. A common ordering mechanism uses logical time [3] to order events according to the causal order [4]. The causal precedence relation induces a partial order on the events of a distributed computation. It is a powerful concept, which helps to solve a variety of problems in distributed systems like algorithms design, concurrency measurement, tracking of dependent events and observation [5].

In many applications, causal order based on logical time is not enough to capture the dependencies needed by the application's logic. Vector clocks bring progress over logical (scalar) timestamps by inducing an order that exactly reflects causal precedence [6,7,8]. Matrix clocks [9,10] extend vector clocks by capturing a global property, namely "what A knows about what B knows about C". Such shared knowledge is needed in many instances involving close cooperation, such as replica update management and collaborative work. However, matrix clocks require $O(n^3)$ global state size for a n-node system, and change propagation still needs $O(n^2)$ message size [11]. This precludes the use of matrix clocks for large-scale systems (several hundreds or thousands of nodes).

A solution to achieve scalability on a MOM is to divide the node interconnection graph in several smaller interconnected groups. Inter-group communication is performed by special nodes called causal router servers, which link two or more groups and are responsible for transmitting messages while maintaining global causal consistency. This solution reduces the amount of necessary information that needs to be stored and exchanged to maintain causality. However, to be scalable, it requires that all computations should be kept local; an algorithm that requires the cooperation of all nodes does not scale well.

This paper proposes a solution based on the splitting of the MOM in domains of causality, which improves performance while reducing causality related costs. We have proved the following property : **iff there is no cycle in the domain interconnection graph[1], local causal ordering in each domain guarantees global causal ordering**. Thus, through a purely local (domain-wise) algorithm, the scalability of causal ordering through matrix clocks in a MOM is greatly improved. With a suitable choice of domains, the communication costs are now linear (instead of quadratic) with respect to the application size (number of nodes). We have implemented this algorithm in AAA[2], a MOM developed in our group. The results of the performance measurements confirm the predictions.

This paper is organized as follows. Related work on MOM scalability is surveyed in Section 2. Section 3 presents the AAA MOM and Section 4 introduces the domains of causality and presents the proof of the main result on domain-based causality. Section 5 describes the implementation of causality domains in the AAA MOM. Section refPerformance-evaluation presents performance results. We conclude in Section 7.

[1] For a precise characterization of this property, see Section 4.2

[2] The AAA MOM was developed in the Sirac laboratory in collaboration with the Bull-INRIA Dyade consortium, and is freely available on the Web with an implementation of the JMS interface. `http://www.objectweb.org/joram/`

2 Related Work

Many solutions have been proposed to lower the cost of causal ordering by reducing the amount of control information. A first group of solutions is based on vector clocks, which require causal broadcast and therefore do not scale well. These solutions include [12], in which nodes are grouped in hierarchically structured clusters, and [13] in which nodes are organized in a Daisy architecture.

A solution based on Hierarchical Matrix Timestamps (HMT) is proposed in [14]. The HMT stores information about other nodes in the same domain and summarizes information about other domains. But this technique is specially adapted to update propagation in replicated databases using a weak consistency algorithm[3] and is not suitable for causal communication.

An original solution for causal delivery is introduced in [15]. This solution does not use a logical clock and implements the causal history relation with lists of causally linked messages. The nodes interconnection graph is split in *subnets* separated by *vertex separators*. This approach allows cycles in the subnet interconnection graph and reduces the size of exchanged information. It does not reduce the amount of control information necessary within a subnet, which is detrimental to scalability.

A last set of solutions is based on the interprocess communication topology. In [16], processes are assumed to communicate mostly with processes of the same group, and the causal history can be omitted for messages exceptionally addressed to a process of another group. The same idea was developed in [17]. The algorithm proposed improves the FM class of algorithms[4] and brings up some solutions to reduce the clock size. One solution uses the communication path : a process only keeps the information about the set of processes with which it may communicate. But this algorithm does not ensure the global causal delivery of messages.

From this brief survey, we may conclude that both the message size (on the network) and control information size (on the nodes) are crucial as far as scalability is concerned and must be treated with the same importance. The next two sections present our solution, using the AAA MOM as a test bed.

3 The AAA Environment

The AAA (Agent Anytime Anywhere) MOM [18] is a fault-tolerant platform that combines asynchronous message communication with a programming model using distributed, persistent software entities called agents. Agents are autonomous reactive objects executing concurrently, and communicating through an event-reaction pattern [19]. Agents are persistent and their reaction is atomic, allowing

[3] Each replica autonomously updates local copies and periodically propagates the log of update operations to other replicas.

[4] The authors define the FM algorithms as all the causal ordering algorithms which use a logical clock (counter, vector or matrix) and mark each event (message) with a timestamp information.

recovery in case of node failure. The Message Bus (i.e. the MOM) guarantees the reliable, causal delivery of messages. The MOM is represented by a set of agent servers (or servers for short) organized in a bus architecture (see Fig. 1).

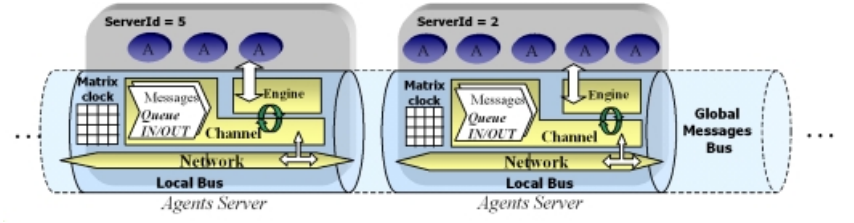

Fig. 1. Two interconnected Agent Servers details.

The combination of the agents and bus properties provides a solution to transient nodes or network failures, while causal ordering decreases the non-determinism of asynchronous exchanges. Each server is made up of three components, the *Engine*, the *Channel* and the *Network*, which are implemented in the Java language. The *Engine* guarantees the Agents' properties and the *Channel* ensures reliable message delivery and causal order. The causal order algorithm uses a matrix clock on each server, which has a size of n^2 for n servers. This causes two problems :

- Network overload, due to timestamp data exchange for clock updates. Even if only modifications of the matrix are sent instead of the full matrix (the *Updates* optimized algorithm described in Appendix A, which a similar approach can be found in [20].), the message size is $O(n^2)$ in the worst case.
- High disk I/O activity to maintain a persistent image of the matrix on each server in order to recover communication in case of failure.

The *Network* component is responsible to send the messages (basic communication layer). Our solution, presented in the next section, uses a decomposition approach to solve these two problems.

4 Domain-Based Causality

To solve the problem of the control information size for matrix clocks, we propose to replace single bus architecture by a virtual multi-bus (or Snow Flake) architecture [21]. Causality is only maintained on each individual bus, also called a *domain of causality*. The idea of splitting the set of servers is not new, but the published solutions either use vector clocks and broadcast [12,13], or use matrix clocks but only reduce the message timestamp size [14,15]. A quite similar solution based on group composition was used in [22], but the authors only prove that if the groups topology is a tree the end-to end ordering is respected, but

not the opposite. Moreover the ordering algorithm was not given, and no implementation exists. In our paper we prove that if the domains interconnection graph is cyclic then the global causality is not respected and vice-versa.

Our solution, combined with the *Updates* optimization, reduces both the message timestamp size and the control information size on servers. In addition, the modularity of the architecture allows it to be adapted to any physical or logical topology.

We have found that if causality is enforced in each domain, then it is automatically respected globally, provided **there is no cycle** in the domain interconnection graph. The proof of this property is presented in Section 4.3.

4.1 Domain of Causality

A domain of causality is a group of servers in which the causal order is respected. Adjacent domains are interconnected by a specific server, which has the function of router and is responsible of message transmission between domains while respecting causality. Such a server, which is in at least two domains, is called a *causal router-server*. In our system, the architecture is not imposed[5]. Domains may be freely organized in any acyclic graph and the logical architecture can be easily mapped on the real network topology[6] to improve the delivery algorithm. This technique reduces the size of the information exchanged on the network and furthermore decreases the size of the control information on the servers. Both factors improve the scalability of the MOM (a performance analysis and experimental results are given in section 6).

As an example (see Fig. 2) an 8-server MOM is logically split in four domains. Domain A includes {S1,S2,S3}, domain B includes {S4,S5}, domain C includes {S7,S8} and domain D and E includes respectively {S3, S6} and {S1,S4}.

Causal message ordering is enforced in each single domain. When a client (agent) connected e.g. to server 1 needs to communicate with a client connected to server 8, the message must be routed (like an IP packet on a network) using paths S1→S3 (domain A), S3→S6 (domain D), S6→S8 (domain C). Message routing is ensured by the system and is completely invisible to the clients, which are not aware of the interconnection topology. A server that needs to send a message to a server in another domain must send it to a causal-router-server (servers S1, S3, S4 and S6 in Fig. 2).

4.2 Domain-Based Computations

A distributed domain-based system is defined by a set $\mathbb{P} = \{p_1, ..., p_n\}$ of processes and a set $\mathbb{D} = \{d_1, ..., d_n\}$ of domains. The distribution of processes among domains is defined by a subset $\mathbb{R}$ of $\mathbb{P} \times \mathbb{D}$: process p is in domain d (noted $p \in d$)

[5] As opposed to the solution of [13], which imposes a daisy architecture and [12], which imposes a hierarchic architecture.

[6] In practice, most network are connected to form a spanning tree specifically to avoid cycles.

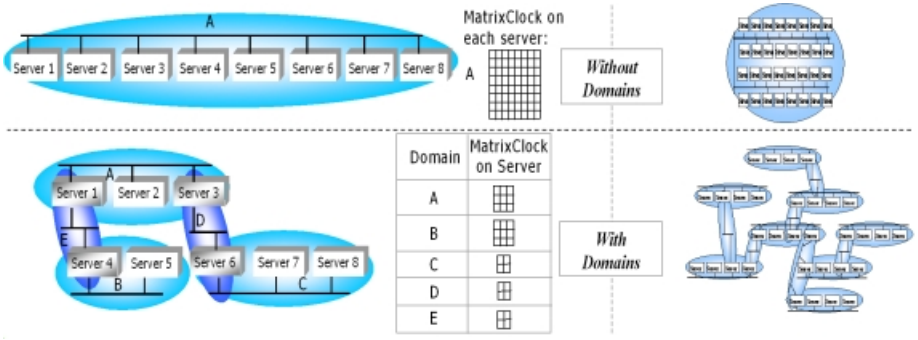

Fig. 2. Example of domains of causality

iff $(p, d) \in \mathbb{R}$. Processes communicate through messages. A computation is defined by a set of messages $\mathbb{M} = \{m_1, \ldots, m_q\}$. For each message m, there is a process $src(m)$, the sender, and a different process $dst(m)$, the receiver. The global history (or *trace*) of the computation is defined by the set of events associated with message send and receive. We assume that a process can only exchange messages with processes in the same domain (interdomain communication is examined later). Within a process p, we define a local order on messages sent or received by p : let m, m' be two such messages; then $m <_p m'$ means that (in the local time of process p) the sending or receiving of m precedes the sending or receiving of m'.

Causal precedence is usually defined between events; we extend it to messages as follows. Let m, m' be messages; m *causally precedes* m' (noted $m \prec m'$) iff one of the following three conditions holds :

- m and m' are sent by a process p, and m is sent before m' : $src(m) = p \wedge src(m') = p \wedge m <_p m'$.
- m is received by a process p, and p later send m' : $dst(m) = p \wedge src(m') = p \wedge m <_p m'$.
- there is a message n such that $m \prec n \wedge n \prec m'$.

We are only interested in *correct* traces, i.e. traces for which relation $\prec$ is a partial order $(m \neq m' \wedge m \prec m' \Rightarrow \neg(m' \prec m))$. A correct trace *respects causality* iff the order in which messages are received by each process p agrees with the causal order : $dst(m) = p \wedge dst(m') = p \wedge m \prec m' \Rightarrow m <_p m'$. A trace respects causality in domain d iff its restriction to d (i.e. its subset restricted to messages with source and destination in d) respects causality. As mentioned before, messages can only be exchanged between processes in the same domain. However, since a process may be included in several domains, indirect communication is possible, through a chain of messages, between processes in different domains. In order to formalize this indirect communication, we introduce the notions of path and chain.

A (process) *path* (of length c) from process p_1 to process p_c is a nonempty sequence $(p_1, ..., p_c)$ of processes such that any two consecutive processes in this

sequence are in the same domain : $\forall i < c, \exists d \in \mathbb{D}, p_i \in d \wedge p_{i+1} \in d$. We call p_1 the source of the path and p_c its destination. A *direct* path is one in which all processes are different (no loops); a *minimal* path is a direct path such that $i+1 < j \Rightarrow \neg(\exists d \mid p_i \in d \wedge p_j \in d)$ (the path does not "linger" in a domain). As a direct consequence, a minimal path of length > 2 has its origin and destination in different domains. A *cycle* is a direct path such that there is a domain including the source and the destination of the path, and there is no domain including all processes in the path. This generally corresponds to a cycle in the domain graph (two domains are connected in this graph if there is a process included in both domains) but not always (for example, if a domain is included in another one, a situation that does not occur in practice).

A *chain* (of length k) in a trace is a nonempty sequence $(m_1, ..., m_k)$ of messages such that each message (after the first one) is sent by the process that received the preceding message, after the receive : $\forall i < k, \exists p \in \mathbb{P}$, $dst(m_i) = p \wedge src(m_{i+1}) = p \wedge m_i <_p m_{i+1}$. We call $src(m_1)$ the source of the path and $dst(m_k)$ its destination. The path associated with a chain is $(src(m_1), src(m_2), ..., src(m_k), dst(m_k))$; this sequence is indeed a path since any two consecutive processes are in the same domain (since messages are local to a domain). A direct (resp. minimal) chain is a chain whose associated path is direct (resp. minimal).

Communication between processes in different domains is performed by means of "virtual messages". A *virtual message* between process p in domain d and process p' in domain d' is represented by a chain of (real) messages, with source p and destination p'. We can then define "virtual traces" in which all messages are virtual. A virtual trace may be formally defined as follows. Let T be a trace. We define T' as a *virtual trace* associated with T by defining a set $C = c_1, ..., c_k$ of minimal chains of T that do not "crossover" : if m_i and m_{i+1} are two consecutive messages of a chain $c \in C$, then no message of another chain $c' \in C$ can be sent by $p = dst(m_i)$ after m_i is received and before m_{i+1} is sent (see Fig. 3). Then T' is the trace derived[7] from T by considering each chain $(m_1, ..., m_k) \in C$ as a direct message from $src(m_1)$ to $dst(m_k)$. Several virtual traces may be derived from a (real) trace, including the real trace itself (by defining $C = \{(m_1), ..., (m_q)\}$. A correct virtual trace is one associated with a correct real trace.

4.3 The Main Theorem

Our main result gives the condition under which a virtual trace respects causality globally. It provides the base for an efficient, scalable causal message system based on domain decomposition.

Theorem 1. *The following two propositions are equivalent :*

P1 Any virtual trace associated with a correct trace that respects causality in each domain respects.

P2 The domain interconnection graph is acyclic.

[7] This derivation is straightforward, but its formal description is cumbersome.

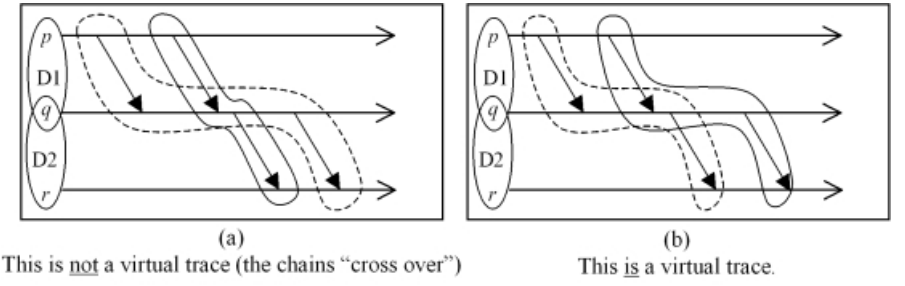

Fig. 3. Virtual traces and real traces

The proof of the theorem uses the following lemmas (see Appendix B for demonstrations).

Lemma 1 *In a correct trace, for any chain* $(m_1, ..., m_k)$ *whose source* p *and destination* q *are different, there exists a direct chain* $(n_1, ..., n_L)$ *with the same source and destination, such that* $m_1 \leq_p n_1$ *and* $n_L \leq_q m_k$.

Lemma 2 *If a correct trace does not respect causality, then there are two processes* p *and* q*, a message* n *from* p *to* q*, and a chain* $(m_1, ..., m_n)$ *from* p *to* q *such that* $n <_p m_1$ *and* $m_n <_q n$.

The proof of the main theorem is in two parts: we first prove $P1 \Rightarrow P2$ (by proving $\neg P2 \Rightarrow \neg P1$); then we prove $P2 \rightarrow P1$.

Part 1 : $\neg P2 \Rightarrow \neg P1$. If there is a cycle in the domain graph, there exists a correct virtual trace, associated with a correct real trace that respects causality in each domain, which does not respect causality globally.

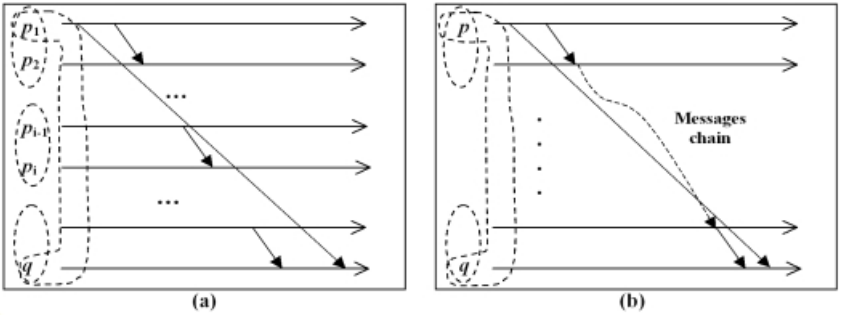

Fig. 4. Break of causality through a cycle of domains.

Since there is a cycle, there exists a direct path $(p, ..., p_i, ..., q)$ such that there is a domain including p and q, and there is no domain including all processes

of the path. Then consider the trace represented on Fig. 4 (a) (taking the real trace itself as a virtual trace).

This trace respects causality in each domain, because (a) no single domain includes all processes of the path; therefore, the restriction of the trace to any domain does not include all messages of the global trace; and (b) a trace derived from the original global trace by removing any single message respects causality. However, the trace does not respect global causality (consider the situation between p and q). This completes Part 1 of the proof.

Part 2 : $P2 \Rightarrow P1$. Lemma 2 may be reformulated as follows : if in a correct trace T there does not exist two processes p and q, a message n from p to q, and a chain $(m_1, ..., m_n)$ from p to q such that $n <_p m_1$ and $m_n <_q n$, then T respects causality. Therefore, in order to prove $P2 \Rightarrow P1$, we shall prove $P2 \Rightarrow P(x)$, where $P(x)$ is the following property : for any virtual trace associated with a correct trace that respects causality in each domain, there does not exist two processes p and q, a virtual message n from p to q represented by a minimal chain of length x, and a chain $(m_1, ..., m_n)$ of virtual messages from p to q such that $n <_p m_1$ and $m_n <_q n$. The proof is by induction on x.

Proof of $P(1)$ **:** By contradiction. Consider a correct virtual trace associated with a correct trace that respects causality in each domain. Assume there exists, for such a trace, two processes p and q, a virtual message n' from p to q represented by a chain of length 1 (the real message n), and a chain $(m'_1, ..., m'_n)$ of virtual messages from p to q such that $n' \leq_p m'_1$ and $m'_n \leq_q n'$ (this is the situation represented on Fig. 4 (b)).

Let $(m_1, ..., m_k)$ be the real chain representing the virtual chain $(m'_1, ..., m'_n)$. By Lemma 1, there is a direct chain $(n_1, ..., n_h)$ such that $m_1 \leq_p n_1$ and $n_h \leq_q m_k$. This chain cannot have length 1, otherwise causality would be violated in the domain including p and q (such a domain exists since there is a real message from p to q). Let then $(p, r, ..., q)$ be the (direct) path associated with this chain. No domain includes all processes of this path, because causality would be violated in such a domain. But p and q are in the same domain, hence $(p, r, ..., q)$ is a cycle, which contradicts $P2$. Therefore $P(1)$ is true.

Induction step : Assuming $P(y)$ holds for any $y < x$, we shall prove $P(x)$. The proof is by contradiction. Consider a correct virtual trace associated with a correct trace that respects causality in each domain. Assume there exists, for such a trace, two processes p and q, a virtual message n' from p to q associated with a minimal chain $(n_1, ..., n_x)$ of length $x > 1$, and a virtual chain $(m'_1, ..., m'_n)$ from p to q such that $n' < pm'_1$ and $m'_n <_q n'$ (this is similar to Fig. 4 (b), replacing the message from p to q by a virtual message).

Let $(p, a_1, ..., a_{x-1}, q)$ be the (minimal) path associated with chain $(n_1, .., n_x)$. Let $(m_1, ..., m_u)$ be the real chain corresponding to the virtual message chain $(m'_1, ..., m'_n)$. By Lemma 1, there exists a direct chain $(L_1, ..., L_v)$ such that $m1 \leq_p L_1$ and $L_v \leq_q m_u$. Let $(p, b_1, ..., b_{v-1}, q)$ be the corresponding direct path (note that $v > 1$, otherwise $(n_1, ..., n_x)$ would not be minimal). Now consider the path $(a_{x-1}, ..., a_1, p, b_1, ..., b_{v-1}, q)$. There are two possible cases.

Case 1 : all processes of path $(a_{x-1}, ..., a_1, p, b_1, ..., b_{v-1}, q)$ are different. In that case, there is a domain including a_{x-1} and q, and there is no domain including p and q (because the minimal path $(p, a_1, ..., a_{x-1}, q)$ has length > 2), and hence no domain includes all processes of the path. Therefore the path is a cycle, which contradicts $P2$.

Case 2 : not all processes of path $(a_{x-1}, ..., a_1, p, b_1, ..., b_{v-1}, q)$ are different. Since the ai are all different and distinct from p and q, and so are the b_j, there must be i and j such that $a_i = b_j$. By definition of a virtual trace, it is not possible for message L_{j+1} to be sent after n_i is received and before n_{i+1} is sent. Therefore, only the two cases represented on Fig. 5 may occur. In case (a), the virtual trace associated with the (non crossing) minimal chains $\{(n_1, ..., n_i), (L_1), ...\}$ does not satisfy $P(i)$, with $i < x$. In case (b), the virtual trace associated with the (non crossing) minimal chains $\{(n_{i+1}, ..., n_x), ...(L_v)\}$ does not satisfy $P(x - i)$, with $x - i < x$. Therefore, in both cases (a) and (b), there is a virtual trace that does not satisfy $P(y)$, $y < x$, which contradicts the induction hypothesis.

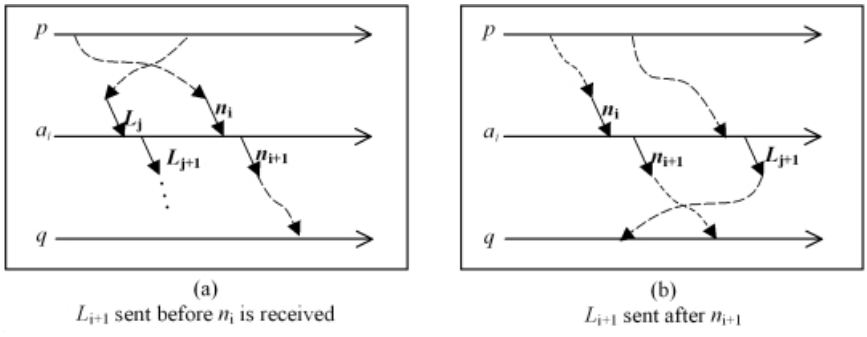

Fig. 5. "Cross over" of virtual traces.

In both cases 1 and 2, we find a contradiction. Therefore $P(x)$ is true, and so is $P(n)$ for any $n = 1$. This completes part 2 of the proof, $P2 \Rightarrow P1$.

5 Implementation

Recall that the MOM is logically split in a set of interconnected domains; causality is enforced within each domain and there is no cycle in the domain interconnection graph. This transformation of the MOM must be transparent for the clients; i.e. agent names must remain unchanged at the application level. Two problems need to be solved: the management of multiple matrix clocks (in the case of causal-router-servers, which belong to more than one domain), and message routing.

Server Modification

To solve the first problem, we have created on each server a local bus of domain (structure called *Message Consumer*), one for each domain that includes

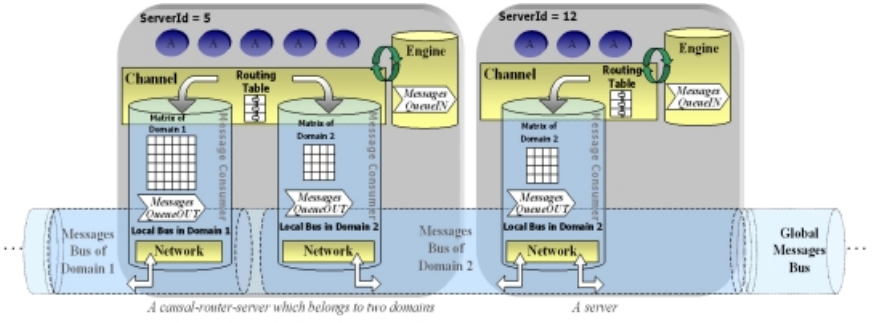

Fig. 6. Two examples of the new agent server structure.

the server. To solve the problem of message routing we have followed the classical network protocol approach, using a routing table. These new structures are represented in Fig. 6.

An agent server now has as many *MessageConsumer* as domain, which own a message queue and the matrix clock associated to its domain. With this implementation, a server can belong to an arbitrary number of domains, and any server can be a causal-router-server. The routing table gives, for each destination server, the identifier of the server to which the message should be sent: the destination server, within a domain, and a router server otherwise. The routing table is built statically at boot time. During server initialization, the servers and the routing tables are constructed, based on a shortest path algorithm.

The *Channel* ensures the transmission of messages and guarantees reliability and causal ordering (see section 3). The *Channel* put messages into the proper *MessageConsumer* using the routing table, then piggybacks messages with a matrix timestamp corresponding to the domain to which the message is sent. At reception, the recipient *Channel* checks the message timestamp and redirects the message to either the local queue (*QueueIN*) or the proper *MessageConsumer* according to its destination domain (see Fig. 7).

6 Performance Evaluation

6.1 Protocol Description

The simplest performance indicator for the AAA MOM is the turn-around time of a message between servers, which is the sum of two terms: the first one related to transfer itself (serialization-deserialization, transfer time, agent saving), the second one related to causal ordering (checking, updating and saving the matrix clock). The first term is nearly constant under our experimental conditions. Therefore the results are a good indicator of the efficiency of the causal ordering algorithm.

For the experiments, we have created an agent on each agent server, which sends back received messages (ping-pong). Messages are sent by a main agent on

Sender Message_Consumer

```
evt = Get_Agent_Sent_Event()
// an agent send an event
domainDestServer= RoutingTable[evt.dest]
messCons = MessageConsumer(domainDestServer)
// get the message consumer of
// the destination domain server
stamp = messCons.matrixclock(domainDestServer)
// get the stamp of the message
// = the updated matrix of the domain
msg = evt + stamp
messCons.network.Send(msg) ⟶
// the message is saved into
// MessageQueueOut and sent
Recv(ACK) ⟵
Remove(evt)
// from the MessageQueueOUT
```

Receiver Message_Consumer

```
⟶ msg = messCons.Recv
// receive a message from a MessageConsumer
// message is saved into LocalQueue
⟵ Send(ACK)
Check(messCons.matrixclock)
messCons.matrixclock.update(msg.stamp)
IF (evt.dest == this.server)
 | Push(evt, QueueIN)
 | // push event to message queue QueueIN
 | // the Agent destination will react to it
ELSE
 | systemDest= RoutingTable[evt.dest]
 | messCons = MessageConsumer(systemDest)
 | // get the message consumer of
 | // the destination domain
 | stamp = messCons.matrixclock(systemDest)
 | // get the stamp of the message
 | // = the updated matrix of the domain
 | msg = evt + stamp
 | messCons.network.Send(msg)
 | // the message is transferred into
 | // MessageQueueOut and sent to the
 | // next router-server or server
FI
```

Fig. 7. Channel Modification

server 0, which computes the round-trip average time for 100 sends. We did three series of tests: unicast on the local server, unicast on a remote server, broadcast on all servers. We did a series of experiments on a single host, but the number of servers was limited to 50 (because a single host cannot support more than 50 Java Virtual Machines). We then set up a network of ten hosts in order to increase the number of servers. We used PC Bi-Pentium II 450MHz with 512 Mo and a 9 Go SCSI hard drive, connected by a 100Mbit/s Ethernet adapter, running Linux kernel 2.0 or 2.2 (depending on machines). We only present the main results (full results are in [23]).

6.2 Experiments and Results

The initial measurements (with no causality domains) clearly show the quadratic increase of the message ordering cost with the number of servers, for both single host and multiple hosts experiments. Fig. 8 and Fig. 9 show typical results.

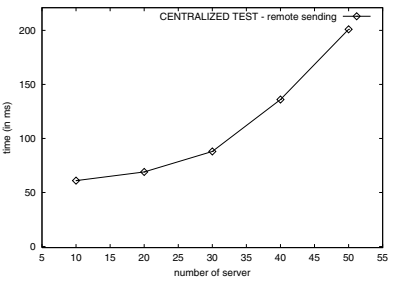

Fig. 8. Remote unicast without *domains of causality.*

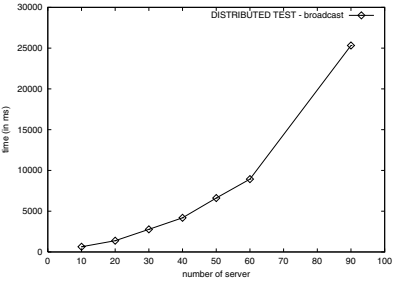

Fig. 9. Broadcast without *domains of causality.*

For the experiments with causality domains, we used a bus-like domain organization (Fig. 10). Other possible organizations are daisy and tree.

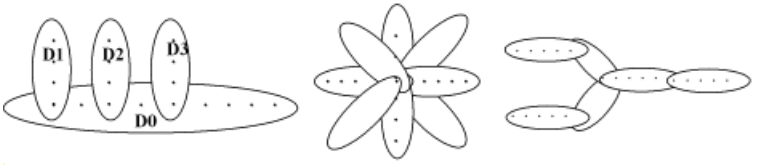

Fig. 10. A Bus, a Daisy and a Hierarchical division of domains.

With the bus-like organization, Fig. 11 shows a linear increase of the communication time (up to 150 servers, the limit of our experiment). To explain the linear dependency, consider a tree of domains of depth d, where each domain has k sub-domains exactly and s servers ($2 \leq k \leq s-1$), then the total number of servers is $n = 1 + (s-1)(k(d+1)-1)/(k-1) \approx sk^d$ and the maximum cost of sending a message is $C \approx (2d+1)s^2$ (the cost of sending a message in a domain of s servers is supposed to be s^2).

The linear cost results from our splitting in $\sqrt{n}$ domains of $\sqrt{n}$ servers with a fixed depth $d = 1$ (bus) which causes a cost of $C \approx K \times n$. For a general tree in which s and k are fixed (and $d > 1$), it would be possible to obtain logarithmic cost, because $C \approx 2ds^2 \approx 2s^2(\ln(n) - \ln(s))/\ln(k) \leq 2s^2 \ln(n)/\ln(k)$, so $C \approx K' \times \ln(n)$. However, $K' > K$ (in particular if we take into account the cost of message routing, proportional to d), therefore, a tree may be less efficient than a bus in some cases.

Fig. 12 clearly shows the performance gain brought by the use of causality domains.

All these results slightly depend on our peculiar test bed but nevertheless we believe that they can be considered sufficiently general for MOMs using a MatrixClock-based causal order with persistency.

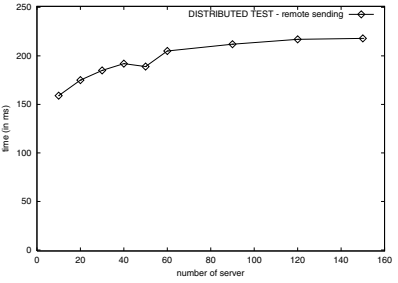

Fig. 11. Remote unicast with *domains of causality.*

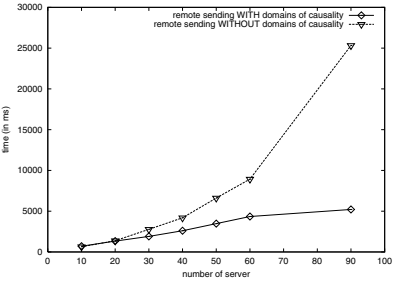

Fig. 12. Comparison of the cost with and without *domains of causality.*

7 Conclusion and Future Work

We have presented a solution based on *domains of causality* to reduce the cost of causal ordering in a scalable Message-Oriented Middleware based on matrix clocks. A domain of causality is a group of servers in which causal message delivery is enforced. The domains are interconnected by specific servers called causal router-servers which transmit messages between domains while guaranteeing the respect of causality. We have proved that the causality is globally respected in the entire network iff there is no cycle in the domains interconnection graph. Our solution has been successfully implemented on the AAA MOM and generates linear increase of the causal ordering costs with the number of servers, instead of quadratic increase with the classical causal ordering algorithm.

The modularity of this solution has many benefits. It is well adapted to a mobile environment (a group of mobile phones is represented by a domain and a station by a causal-router-server) and to LANs interconnection. However, the division of the MOM in domains needs to be done carefully and the new problem is to find an optimal splitting. Two directions may be followed: first, the splitting can be made according to the network architecture, secondly it can be made according to the application's topology. This latter solution exploits the description of applications (e.g. with an Architecture Description Language [24]) to obtain the application graph connectivity and to determine an optimal split of the communication architecture.

Our future work will further investigate the problem of optimal splitting of a MOM into domains, taking into account application needs and communication costs.

Acknowledgements. The authors gratefully acknowledge Luc Bellissard and André Freyssinet for their valuable and very useful advices on this work. We would also like to thank Jacques Mossière and the anonymous reviewers for their precious comments on earlier versions of this paper.

References

1. G. Banavar, T. Chandra, R. Strom, and D. Sturman. A case for message oriented middleware. In *Lecture Notes in Computer Science*, volume 1693, pages 1–18. Distributed Computing 13th International Symposium, September 1999. ISBN 3-540-66531-5.
2. Object Management Group. *CORBA Messaging*, May 1998. White Paper, http://www.omg.org.
3. L. Lamport. Time, clocks, and the ordering of events in a distributed system. In *Communications of the ACM*, volume 21, pages 558–565, 1978.
4. K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. In *ACM Transactions on Computer Systems*, volume 9, pages 272–314, August 1991.
5. M. Raynal, A. Schiper, and S. Toueg. The causal ordering and a simple way to implement it. In *Information Proceeding Letters*, volume 39, pages 343–350, 1991.
6. O. Babaoglu and K. Marzullo. *Consistent Global States of Distributed Systems: Fundamental Consepts and Mechanisms*, chapter 3. S. Mullenderr, addison-wesley edition, 1993.
7. V. Hadzilacos and S. Toueg. Fault-tolerant broadcast and related problems. In S. Mullender, editor, *Distributed Systems*, pages 55–96. Addison-Wesley, 1993.
8. F. Mattern. Virtual time and global states of distributed systems. In *the International Workshop on Parallel and Distributed Algorithms*, pages 215–226, 1989.
9. M.J. Fischer and A. Michael. Sacrifying serializability to attain high availability of data in an unreliable network. In *ACM Symposium on Principles of Database Systems*, pages 70–75, March 1982.
10. G.T.J. Wuu and A.J. Bernstein. Efficient solutions to the replicated log and dictionary problems. In *3nd ACM Symposium on PODC*, volume 99, pages 233–242, 1984.
11. M. Raynal and M. Singhal. Logical time: A way to capture causality in distributed systems. In *IEEE Computer*, pages 49–56, 1995.
12. N. Adly, Nagi M., and J. Bacon. A hierarchical asynchronous replication protocol for large-scale systems. In *IEEE Workshop on Parallel and Distributed Systems*, pages 152–157, October 1993.
13. R. Baldoni, R. Freidman, and R. Van Renesse. The hierarchical daisy architecture for causal delivery. In *IEEE International Conference on Distributed Systems*, pages 570–577, May 1997.
14. T. Johnson and K. Jeong. Hierarchical matrix timestamps for scalable update propagation. submitted to the 10th Workshop on Distributed Algorithms, June 1996.
15. L. Rodrigues and P. Veríssimo. Causal separators and topological timestamping: an approach to support causal multicast in large scale systems. In *15th International Conference on Distributed Systems*, May 1995.
16. Rodrigues L. and P. Veríssimo. *Topology-Aware Algorithms for Large-Scale Communication*, volume 1752, pages 127–156. Springer Verlag LNCS, 2000.
17. S. Meldal, S. Sankar, and J. Vera. Exploiting locality in maintaining potential causality. In *the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 231–239, 1991.
18. L. Bellissard, N. De Palma, A. Freyssinet, M. Herrmann, and S. Lacourte. *An Agent Platform for Reliable Asynchronous Distributed Programming.* Symposium on Reliable Distributed Systems, October 1999.

19. G. Agha, P. Wegner, and A. Yonezawa. *Research Directions in Concurrent Object-Oriented Systems*. MIT Press, Cambridge, 1993.
20. A.D. Kshemkalyani and M. Singhal. An efficient implementation of vector clocks. In *Information Processing Letters*, volume 43, pages 47–52, August 1992.
21. A. Homer and D. Sussman. *Programmation MTS et MSMQ avec Visual Basic et ASP*, chapter 5. Eyrolles, Mars 1999.
22. S. Johnson, F. Jahanian, and J. Shah. The inter-group router approach to scalable group composition. In *19th ICDCS*, pages 4–14, June 1999.
23. Philippe Laumay. Déploiement d'un bus à messages sur un réseau à grande échelle. Rapport de DEA, University Joseph Fourier, June 2000.
24. L. Bellissard, S. Ben Atallah, F. Boyer, and M. Riveill. Distributed application configuration. In *International Conference on Distributed Computing Systems*, pages 579–585. IEEE Computer Society, May 1996.

Appendix

A The *Updates* Algorithm

State // an int value.
Mat[][] // an 2-dimension table representing the Matrix clock.
Mat[x, y].value // the value of the clock for [x,y], i.e. the real clock.
Mat[x, y].state // the last modified state of the (x,y) matrix clock value.
Mat[x, y].node // source node of last modification.
Node[x].state // value of the x channel state during the last sending.
Updates // timestamp sent; = a set of [line,column,value]
// (i.e. Mat[line, column].value).

Initially :

$$\left|\begin{array}{l} State = 0; \\ \forall i, j Mat[i, j].state = 0; \\ \forall i, j Node[i, j].state = 0; \end{array}\right.$$

Sending from S_i to S_j :

$$\left|\begin{array}{l} Node[j].state = State; \\ Mat[i, j].value+ = 1; \\ Mat[i, j].state = State; \\ Mat[i, j].node = j; \\ State+ = 1; \\ Updates = \left\{ (k, l, Mat[k, l].value) \left| \begin{array}{l} Mat[k, l].state > Node[j].state \\ Mat[k, l].node \neq j \end{array} \right. \right\} \end{array}\right.$$

Receiving on S_i from S_j :

$$\left| \forall (k, l, v) \in Updates, Mat[k, l].value < v \rightarrow \left| \begin{array}{l} Mat[k, l].value = v \\ Mat[k, l].state = State \end{array} \right. \right.$$

B Proofs of Lemma 1 and Lemma 2

Lemma 1. In a correct trace, for any chain $(m_1, ..., m_k)$ whose source p and destination q are different, there exists a direct chain $(n_1, ..., n_L)$ with the same source and destination, such that $m_1 \leq_p n_1$ and $n_L \leq_q m_k$.

Proof : by induction on k. The property is trivially true for $k = 1$. Assume it holds for chains of length $k < K$, and consider a chain $(m_1, ..., m_K)$ of length K. If this is a direct chain, the proof is done. If not, then by definition the path $(p_1, ..., p_{K+1})$ is not direct, i.e. there exists $i < j$ such that $p_i = p_j$. Consider the chain $(n_1, ..., n_L)$ defined as follows[8] :

a) $(m_j, ..., m_K)$ if $i = 1 \wedge j < K + 1$.
b) $(m_1, ..., m_{i-1})$ if $i = 1 \wedge j = K + 1$.
c) $(m_1, ..., m_{i-1}, m_j, ..., m_K)$ if $i > 1 \wedge j < K + 1$.

This chain has the same source and destination as $(m_1, ..., m_n)$, and has length $< K$. By the induction hypothesis, there exists a direct chain $(n'_1, ..., n'_h)$ with the same source and destination, and such that $n_1 \leq_p n'_1$ and $n'_h \leq_q n_L$.

In addition, $m_1 \leq_p n_1$ and $n_L \leq_q m_k$. The first inequality trivially holds for case b) and c). It also holds in case a), for $m_j <_p m_1 \Rightarrow m_{j-1} <_p m_1 \Rightarrow m_{j-1} \prec m_1$ which is impossible because the trace is correct and we already have $m_1 \prec m_{j-1}$. Likewise, the second inequality also holds in the three cases. Hence, $m_1 \leq_p n'_1$ and $n'_h \leq_q m_k$. The property therefore holds for chains of length K, which concludes the induction step, and the proof.

Lemma 1' (needed for the proof of Lemma 2). If $n \prec m$, then either there is a chain $(n, ..., m)$, or there is a chain $(l, ..., m)$, where l is a message sent after n by the sender of n.

Proof : by recursive descent, using a case analysis at each step. If $n \prec m$, there are 3 possible cases :

- m and n are sent by the same process p, and m is sent after n. Then there is a chain of the form $(l, ..., m)$, where l is a message sent after n (take e.g. the chain (m)).
- m is sent by a process that previously received n. Then there is a chain of the form $(n, ..., m)$ (take e.g. the chain (n, m)).
- There exists a message k such that $n \prec k \wedge k \prec m$. By recursion (using the analysis of the 2 previous cases), one finds four possible cases, and in each case there exists either a chain $(n, ..., m)$ or a chain $(l, ..., m)$, where l is a message sent after n by the sender of n. The recursion terminates since all chains are finite.

Lemma 2. If a correct trace does not respect causality, then there are two processes p and q, a message n from p to q, and a chain $(m_1, ..., m_n)$ from p to q such that $n <_p m_1$ and $m_n <_q n$.

Proof : if a correct trace does not respect causality, then some process q received message m before message n, and $n \prec m$. By Lemma 1', there exists

[8] The case $i = 1 \wedge j = K + 1$ does not occur, because $p \neq q$.

either a chain $(n, ..., m)$ or a chain $(l, ..., m)$, where l is a message sent after n by the sender of n.

In the first case, since $dst(n) \neq src(m)$, chain $(n, ..., m)$ has the form $(n, l, ..., m)$. Then, by considering chain $(l, ..., m)$, $l \prec m \wedge m \prec l$ (since $m <_q n$ and $n <_q l$), which contradicts the assumption that the trace is correct (Fig. 13 (a)).

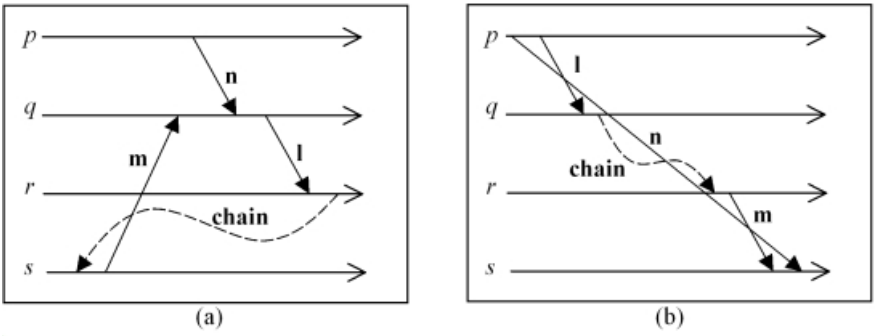

Fig. 13. Break of correctness (a) or causality (b).

In the second case, there is a message n from $p = src(n)$ to q, and a chain $(l, ..., m)$ from p to q such that $n <_p l$ and $m <_q n$ (Fig. 13 (b)). This concludes the proof.