A Multicast Mechanism with Ordering on Overlapping Groups

Xiaohua Jia and Shirley Y. So
Department. of Computer Science,
The University of Queensland, Australia 4072

Abstract

The paper proposes a new mechanism for group communication which guarantees the total ordering of message delivery, especially the ordering on the members common to overlapping groups. The mechanism is based on a method which propagates messages to all destinations along trees. Our approach has the following advantages over the propagation tree mechanism: (1) Greater parallelism, messages can be sent to destinations by using broadcast networks. (2) Less latency time, it takes less time to have a message reach all the destinations. (3) More flexible to dynamic membership changes, each change of process membership does not involve a restructuring of propagation trees.

1. Introduction

Multicast communication is used to send messages to a group of processes. There are two important features of multicast communication: atomicity and ordering. The atomicity is to guarantee that a multicast message is either received by all operational processes in the multicast group or by none of them. The ordering is to guarantee that messages are delivered to destination processes in the appropriate order so that all processes see the messages (events) in the same relative order. This paper concentrates on the issues of ordering. The ordering of multicast messages can be classified into the following categories:

(1) Single source single group (*SSSG*) ordering. See Fig. 1(a), messages m_1 and m_2 originating from the same source site are sent to the same group of processes. m_1 and m_2 must be received by all destination processes in the same order as they are sent out.

(2) Single source multiple group (*SSMG*) ordering. See Fig. 1(b), messages m_1 and m_2 originating from the same source site are sent to two multicast groups which may overlap each other. m_1 and m_2 must be received by the members common to both groups in the same relative order.

(3) Multiple source single group (*MSSG*) ordering. See Fig. 1(c), messages m_1 and m_2 originating from different source sites are sent to the same group of processes. m_1 and m_2 must be received by all destination processes in the same relative order.

(4) Multiple source multiple group (*MSMG*) ordering. See Fig. 1(d), messages m_1 and m_2 originating from different source sites are sent to two multicast groups which may overlap each other. m_1 and m_2 must be received by the members common to both groups in the same relative order.

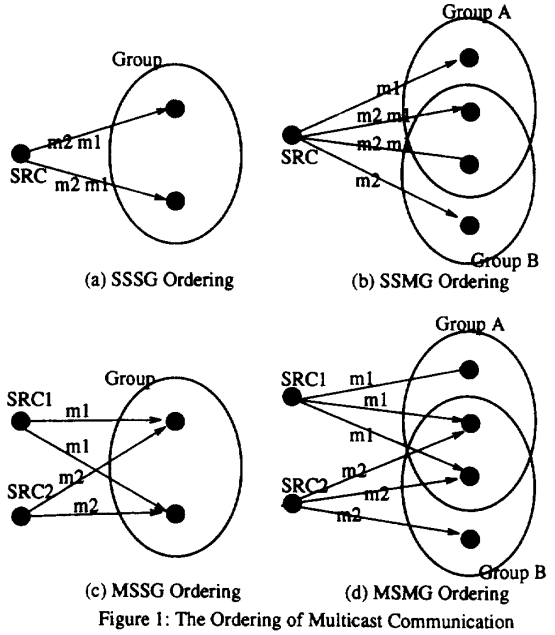

Figure 1: The Ordering of Multicast Communication

If all the four types of ordering are guaranteed then the multicast communication is regarded as being totally ordered. As surveyed in [8], there are many distributed applications, as well as distributed operating systems, which require multicast communication facilities, such as the coordination of replicated servers, distributed process control systems and distributed transaction management. However, it is not easy to achieve all these ordering

0-8186-3670-X/93 $03.00

properties of multicast communication.

The *SSSG* ordering is relatively easy to achieve and sometimes can be done by the underlying communication network. The basic idea is to number the messages at the source and to have the messages delivered to destination processes in that order.

The *MSSG* ordering can also be guaranteed by some relatively easy methods. One simple solution is to assign the site on which one (or more) member process resides to be the primary site for each group. Messages from various sources to a group are first sent to the primary site where they are ordered. The primary site then propagates the messages further to the other members of the group.

The *SSMG* and *MSMG* orderings is much harder to guarantee because destination groups may overlap each other. For example, in Fig. 1(d), a process belonging to both group A and B receives message m_1 first, then m_2. But another process in both group A and B could receive message m_1 and m_2 in the reverse order, because sending message m_1 to group A is a separate operation from sending message m_2 to group B.

We propose an algorithm which regroups the processes according to the situations of overlapping among multicast groups so that the resulting process groups (called *meta-groups*) do not overlap each other. Those meta-groups are organized into a tree (or a forest of trees). Messages multicast to a group are propagated to the meta-groups, which are the subsets of the group, along the tree. The meta-groups, in turn, send the messages to their members. Since the meta-groups do not overlap each other, the *SSMG* and *MSMG* orderings on common members of overlapping groups are simplified to the problem of single group orderings (*SSSG* and *MSSG*).

Our approach is inspired by Garcia's propagation tree mechanism[5], which organizes sites in multicast groups into trees and then propagates messages along the trees to destinations. Our approach has solved several major problems left in Garcia's mechanism and improved the performance considerablely. It has the following advantages over the propagation tree mechanism:

(1) Greater parallelism. The nodes on propagation trees are meta-groups, rather than members of multicast groups. Sending messages to meta-group members can be done by broadcast networks, taking the advantage of underlying network hardware.

(2) Shorter propagation tree. The depth of the subtree corresponding to a multicast group is very short, usually 2. Thus it takes much less latency time to propagate a message to all the destinations.

(3) More flexible to changes of group membership. Any change in the membership of a multicast group will not affect the tree structure provided this membership change does not incur any creation or deletion of a meta-group, thus resulting in a greater flexibility for handling dynamic group membership.

2. Related Work

Many multicast mechanisms have been proposed and implemented. According to their design approaches, those mechanisms can be classified into centralized, distributed and the combination of the both. Centralized mechanisms [7,10] (including token based mechanism [3]) use one site in the system as the central site (or the token site which is changeable when the token is passed from one to another). All multicast cast messages are first sent to the central site where they are ordered, then the central site sends the messages to group members. Total ordering is guaranteed in these types of mechanisms. Another type of centralized mechanism [8] guarantees the ordering of messages within a group. It elects a group manager for each group. Any messages to a group are first sent to the group manager which, in turn, broadcasts the messages to other group members. The centralized mechanisms are usually easy to understand and easy to implement. However, they have disadvantages such as the performance bottleneck at a central site (token site), poor fault tolerancy (crash of a central site), low scalability, and so on.

A set of group communication mechanisms *ABCAST*, *CBCAST* and *GBCAST* were proposed for and implemented in the *ISIS* system [1,6]. *ABCAST* is for transmitting messages to process groups which need total ordering. *CBCAST* is for guaranteeing causal ordering among messages. *GBCAST* is a protocol specially for handling the change of group memberships. The idea of *ABCAST* is based on a two-phase protocol. An *ABCAST* message received by each receiver (protocol process) will be withheld in one of the *ABCAST* queues. A local timestamp is assigned to the message and this timestamp is sent back to the sender (protocol process). After collecting the timestamps from all the receivers, the sender chooses the maximum to be the global timestamp and sends it to all the receivers. On the receiving the global timestamp, the receiver marks the message as deliverable. But the message will not be delivered to application processes until all messages ahead of it in the pending queue have been delivered. The protocol is very costly and incurs a long latency to deliver an *ABCAST* message. A protocol *TOMP* [4] aiming to reduce the latency time of the *ABCAST* in the last phase was proposed by Dasser later.

CBCAST is less expensive compared with *ABCAST*. To guarantee the causal ordering among messages, the protocol piggybacks all causally related messages (the

messages sent or received by it) to any *CBCAST* message to be sent out. When a destination receives such a packet of messages, it discards the messages it received already and delivers the others. The transfer packet size would become very large if *CBCAST* is called frequently. Schiper proposes another algorithm [9] for causal ordering, which is later implemented as a new version of *CBCAST*[2]. It uses timestamp vectors to order messages. Each site maintains a timestamp vector, one component for each group member. Any message to be sent out is attached with a vector of the sender. A receiver, when receives a message with a vector, compares its own timestamp vector with the incoming message's. It will block the delivery of the message until each element of its own vector is great than or equal to the corresponding element of the message's. This mechanism is only applicable to closed group communications where multicasting only happens within members of a group. As we see, although distributed mechanisms have many advantages over centralized ones, they are much more complex, and usually more costly in both computing time and network communications.

An approach combining both centralized and distributed control was proposed by Garcia-Molina [5]. The mechanism guarantees total ordering, especially the ordering on overlapping groups. The protocol imposes a logical tree structure on the sites in the system, where the intersections of multicast groups are chosen as the intermediary nodes, or roots of trees. Messages addressed to a group are first sent to the primary destination of the group. Then, they are propagated down to the members of the group along the tree. The messages are merged with other messages destined for other groups and ordered by the intermediary nodes all the way along the propagation path. Thus the messages at the intermediary nodes (representing sites belonging to several groups) are ordered. The mechanism suffers from several disadvantages such as low parallelism, low flexibility in the change of group memberships, and the existence of extra nodes (the nodes which are not the members of the destination group on the message propagation path to the destination members).

3. Basic Idea of the Approach

Def.1. Multicast group: a group of processes which are the destinations of a message. It is treated as a unique entity (object) from an application's point of view.

In the design of a multicast protocol, we are mainly concerned about the sites where processes in a group reside, rather than the processes, because to deliver a message to multiple *local* processes is virtually the same as to deliver it to one local process. For simplicity, we use sites where the group processes reside to be the members of a multicast group.

Multicast communication guarantees the ordering of message delivery. That is, all the processes in the system observe the messages arriving in the same sequential order. The difficulty of achieving this ordering feature lies in guaranteeing the message ordering on the members common to overlapping groups. To overcome this difficulty, we introduce the concept of meta-groups.

Def.2. Meta-group: a collection of processes which are the members common to the same set of multicast groups. Any two meta-groups do not overlap each other.

Consider multicast groups *A*, *B*, *C* and *D* (represented by circles) overlapping each other as shown in Fig. 2. The processes in each small area defined by overlapping group boundaries form a meta-group, because they are the members common to the same set of groups. A meta-group is labeled by a string of capital letters enclosed by angle brackets. The letters in the label of a meta-groups indicate the multicast groups of which the meta-group is a subset. For example, the processes in the dark area (common to *A*, *B* and *C*) form a meta-group labeled as <*ABC*>. The processes in the lined area (common to *only A* and *C*, and nothing else) form another meta-group labeled as <*AC*>. All the meta-groups which are subsets of a multicast group make up that group, such as <*A*>, <*AB*>, <*AC*>, <*AD*> and <*ABC*> making up group *A*.

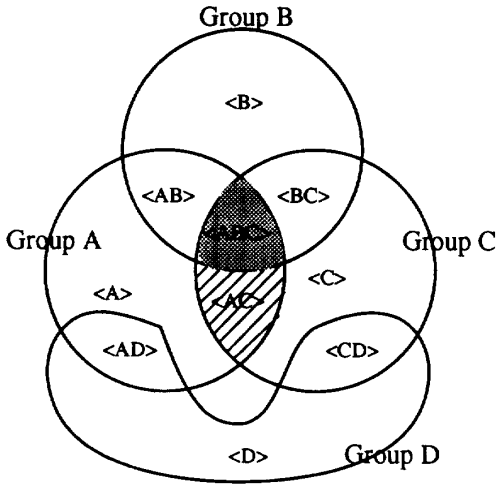

Figure 2: Overlapping Multicast Groups

By using meta-groups, it becomes much easier to guarantee the message ordering on the common members of overlapping groups. Multicasting a message to a group is done by sending the message separately to the meta-groups making up the destination group. Because meta-groups do not overlap each other, the ordering in sending the messages to meta-groups is the case of *SSSG* or *MSSG*

ordering. Each meta-group has a member assigned to be primary site. To send a message to a meta-group, the message is sent to the primary site which in turn multicasts the message to the other members. Fig. 1(d) is alternatively represented by Fig. 3(a) which sets the overlapped area apart from groups A and B. Group A's messages and group B's messages, to their common members, would first be sent to $<AB>$ where they are ordered by the primary site and then multicast to the other members in $<AB>$. All the members in $<AB>$ receive messages in the same ordering as the primary site, provided the *SSSG* ordering is guaranteed.

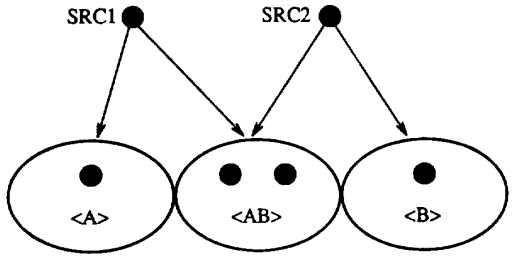

Figure 3(a): An Alternative Representation of Fig.1(d)

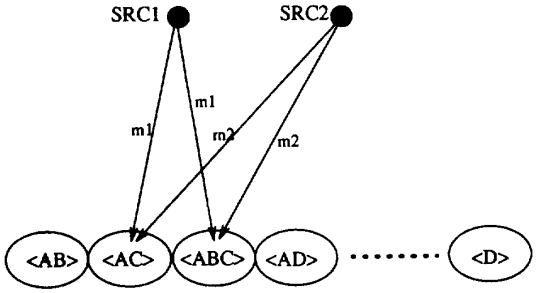

Figure 3(b): An Alternative Representation of Fig.2

However, in a more complex situation, the solution would not be so simple. Consider Fig. 3(b) which is an alternative representation of Fig. 2 by setting all meta-groups apart from each other. Suppose messages m_1 and m_2 are sent to groups A and C respectively. The ordering of m_1 and m_2 observed by the primary site of $<AC>$ could be different from that observed by the primary site of $<ABC>$. Because $<AC>$ and $<ABC>$ are subsets common to both group A and C, messages for A and C will be received by members in both $<AC>$ and $<ABC>$. To guarantee the message ordering at two (or more) meta-groups, messages to the meta-groups must be ordered at some point before they are sent to the meta-groups separately. In the above example, if we make a path from $<ABC>$ to $<AC>$, any messages to $<AC>$ are passed down by $<ABC>$. m_1 and m_2 would be first ordered at $<ABC>$ and then be passed to $<AC>$ in the same order. The paths connecting meta-groups for message propagation would eventually lead to a tree structure. Our mechanism is to organize meta-groups into a propagation tree (or a forest of trees), where messages are ordered at each non-leaf node and then passed down to its child nodes.

Def.3. Primary Meta-group (PM): the PM of a multicast group is its meta-group which is the ancestor of all its other meta-groups on the propagation tree.

Each multicast group has only one PM. A meta-group can be the PM of several groups. Any messages to the multicast group are first sent to the PM and then the PM propagates the message to the other meta-groups.

Rule1. Every multicast group has a unique primary meta-group.

Rule2. Every meta-group has a unique path to it from any PMs of the multicast groups of which it is a subset.

Theorem: Given a set of multicast groups, organize the meta-groups formed from the multicast groups into a tree (or a forest of trees) which satisfies Rule1 and Rule2. For any message destined to a group, if the message is first sent to the PM of the group, and then propagated down to the other meta-groups along the tree, the total ordering of message delivery is guaranteed.

Proof: Suppose messages m_1 and m_2 are destined to group G_1 and G_2 respectively. Consider two processes p_1 and p_2 which receive both m_1 and m_2. p_1 and p_2 must be the members common to G_1 and G_2, otherwise they would not receive both m_1 and m_2. We will prove that p_1 and p_2 receive m_1 and m_2 in the same relative order.

Suppose p_1 is in meta-group $<\alpha G_1 G_2 \beta>$ and p_2 is in $<\alpha' G_1 G_2 \beta'>$ (notice that labels $<\alpha G_1 G_2 \beta>$ and $<\alpha' G_1 G_2 \beta'>$ indicate they are subsets of both G_1 and G_2). If p_1 and p_2 are in the same meta-group, say $<\alpha G_1 G_2 \beta>$, then m_1 and m_2 would be ordered by the primary site of meta-group $<\alpha G_1 G_2 \beta>$, and this order of m_1 and m_2 is preserved when the primary site multicasts m_1 and m_2 to the other members in $<\alpha G_1 G_2 \beta>$. Thus p_1 and p_2, as the members of $<\alpha G_1 G_2 \beta>$, must receive m_1 and m_2 in the same order as the primary site.

Consider that p_1 and p_2 are in different meta-groups. Let PM_1 and PM_2 be the PMs of group G_1 and G_2 respectively. According to Rule2, to $<\alpha G_1 G_2 \beta>$ there is one path from PM_1 ($<\alpha G_1 G_2 \beta>$ is a subset of G_1) and another path from PM_2 ($<\alpha G_1 G_2 \beta>$ is a subset of G_2 as well). By the properties of trees, there is only one path from the root of the tree to any node. PM_1 and PM_2 must be either the same, or on the same path. If $PM_1 = PM_2$, any messages for G_1 and G_2 would be ordered at PM_1 (by the primary site of PM_1), and this order is preserved when PM_1

propagates the messages down to the other meta-groups of G_1 and G_2. Thus $<\alpha G_1 G_2 \beta>$ and $<\alpha' G_1 G_2 \beta'>$ must observe m_1 and m_2 in the same order as PM_1. So p_1 and p_2 receive m_1 and m_2 in the same order as the primary site of PM_1. If $PM_1 \neq PM_2$, suppose PM_1 is ancestor to PM_2. Since $<\alpha G_1 G_2 \beta>$ and $<\alpha' G_1 G_2 \beta'>$ are subsets of G_2, they are children of PM_2 (according to the definition of PM). G_1's message (i.e., m_1) for $<\alpha G_1 G_2 \beta>$ and $<\alpha' G_1 G_2 \beta'>$ must come through PM_2. G_2's messages (i.e., m_2) will be merged with those G_1's messages at PM_2 where they are ordered (by the primary site of PM_2). Following the same discussion as above, we can see p_1 and p_2 receive m_1 and m_2 in the same order.

Our multicast mechanism consists of two aspects. One is building propagation trees, which defines the paths for propagating messages to all expected destinations. The other is message propagation, which propagates messages down to the destinations along the propagation trees.

4. Algorithm for Building Propagation Trees

4.1. Building Propagation Trees

There are two steps for building propagation trees. The first step is to form meta-groups from multicast groups. The second step is to organize meta-groups into propagation trees. The constructed propagation trees must satisfy Rule1 and Rule2. For meta-groups, we define:

Def.4. Cardinality: the cardinality of a meta-group is the number of multicast groups of which it is a subset. In Fig. 2, the cardinality of <*ABC*> is 3, and the cardinality of <*A*> is 1.

Def.5. Subsume: a meta-group subsumes another if, when the second meta-group is a subset of a multicast group, then the first one must be a subset of the multicast group as well. In Fig. 2, <*AC*> subsumes <*A*> and <*C*>.

Def.6. Joint: a meta-group joints with another if there exists at least one multicast group of which both meta-groups are subsets. In Fig. 2, <*ABC*> joints with <*CD*> because both <*ABC*> and <*CD*> are the subsets of *C*.

To establish the meta-groups we use a two dimensional boolean array holding information about the multicast groups. Rows represent the n sites (or processes), columns the m multicast groups. The member processes of a meta-group are all those processes whose row values are identical. The procedure to identify all members of each meta-group is straight forward. Namely, the rows are treated as being representation of integers. If these integers are used as keys to sort the rows then the processes comprising each meta-group will be brought together. Meta-groups can be formed easily and efficiently.

To illustrate how to organize meta-groups into trees, observe the example in Fig. 4 which is the propagation tree for overlapping groups in Fig. 2.

Pick one multicast group, say *A*. We start from the meta-group with the highest cardinality in group *A*, i.e. <*ABC*>, making it the root of the tree. The <*ABC*> would be the PM of group *A*, *B* and *C*. Then we make all the meta-groups subsumed by <*ABC*>, i.e. <*A*>, <*B*>, <*C*>, <*AB*>, <*AC*> and <*BC*>, to be the children of <*ABC*>. After that, we consider the meta-groups which joint with <*ABC*>, i.e. <*AD*>. For each of those meta-groups, make it the child of the root, and start from this meta-group recursively by regarding it as the subroot. This subroot is the PM of the multicast groups of which it is a subset and which do not have a PM already. In this example, <*AD*> would be the PM of group *D* (*A* already has its PM at the root). Then we come down to consider meta-groups in *D* the same way as was done for group *A*. The recursive call terminates when there is no more meta-group which is not on the tree. We start from the highest cardinality meta-group when we consider a multicast group so as to make the overall propagation tree more balanced (i.e., the tree becomes shorter).

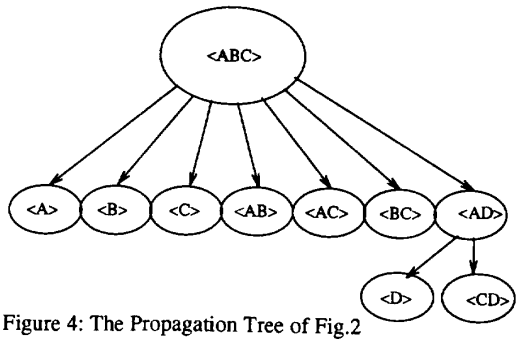

Figure 4: The Propagation Tree of Fig.2

We use a vector *pm*, one component for each multicast group, to record the PMs of multicast groups. All components of *pm* are initialized to empty. The pseudo-code of the algorithm for constructing propagation trees from meta-groups is as below. In the program of the pseudo-code, the `main` procedure is a loop for the multicast groups whose PM is empty. Each repetition starts building a tree from the root by calling `build_tree`, where the root is the meta-group with the highest cardinality of the concerned multicast group. All the meta-groups connected together by a *joint* relationship appear on the same tree. The whole propagation graph is the forest of these trees. The procedure `build_tree` calls itself recursively when it meets a meta-group which is a subset of a new multicast group which has not yet been considered. To reduce the overall depth of a tree, the list of *joint* meta-groups, *JointL*, is ordered by descending

cardinality. Therefore, the one with the highest cardinality would be considered first. A meta-group with a higher cardinality would be the PM of more multicast groups.

```
main ()
{
 for each group G_i whose pm(G_i) = ∅ do:
    make the highest cardinal meta-group MG of G_i as
      the root;
    mark MG;
    build_tree(MG);
 endfor
}

build_tree(PM)
{
 Let RootG be the set of multicast groups of
    which PM is a subset;
 for each G_i in RootG do:
    if pm(G_i) = ∅ then pm(G_i)= PM;
 endfor;

 for any unmarked meta-group subsumed by PM do:
    make it the child of PM and mark it;
 endfor;

 Let JointL be a list of unmarked meta-groups
    which joints with PM;
 for any unmarked meta-group MG_i in JointL do:
    make MG_i the child of PM and mark MG_i;
    build_tree(MG_i); /*recursive call*/
 endfor
}
```

The propagation trees built by our algorithm have the following two properties:

(1) Each multicast group has one and only one PM. This is obvious, because the vector *pm* records only one PM for each multicast group, and the loop in the `main` procedure ensures every multicast group would have one PM.

(2) Each meta-group has one and only path to it from any PMs of its multicast groups. Suppose a meta-group is a subset of a multicast group G_i. When G_i's PM is considered in the `build_tree`, all its meta-groups are put on the tree as the children of the PM, because they are either subsumed by the PM or joint with the PM. It is also easy to see that each meta-group is linked to the tree exactly once.

The above properties of the propagation trees obviously satisfy Rule1 and Rule2. Thus the message ordering is guaranteed if messages are propagated to the destinations along the trees built.

4.2. Optimization of Propagation Trees

There is one problem in the algorithm described above. Consider the propagation tree in Fig. 4 again. When a message is multicast to *C*, the message has to come through <*AD*> to reach <*CD*>. However, <*AD*> has nothing to do with group *C*. <*AD*> is called an intermediary of group C. The existence of intermediaries degrades the efficiency of message propagation. The intermediaries can be eliminated, or at least minimized.

As the general method of optimization, suppose group G_i has some intermediaries to reach one of its meta-groups $<\alpha G_i\beta>$ in Fig. 5(a). Notice that each non-leaf node on the tree is a PM, because a new level of a subtree is built only when an unconsidered multicast group's meta-group is met, and this meta-group becomes the PM of it (refer to the algorithm in section 4.1). An intermediary must be the PM of some multicast group. Traveling down along the path from $PM(G_i)$ to $<\alpha G_i\beta>$, examine each intermediary on the path. If there is no other path from this intermediary to other G_i's meta-groups except the one to $<\alpha G_i\beta>$, then go to the next intermediary along the path. If a $PM(G_j)$ is found that has a child leading to a G_i's meta-group other than $<\alpha G_i\beta>$, say $<\alpha' G_i\beta'>$ in Fig. 5(a), we cannot come down any further, linking $PM(G_j)$ to $PM(G_i)$. Messages for G_i would be propagated to $PM(G_j)$ directly, bypassing all the intermediaries above it. The intermediaries on the path from $PM(G_i)$ to $PM(G_j)$ do not have any other branch leading to G_i's meta-groups. They have nothing to do with G_i's messages except passing them down. If we order G_i's message with other messages merged into the path at the intermediaries at $PM(G_j)$, G_i's meta-groups under $PM(G_j)$ would observe G_i's messages and other messages in the same relative order.

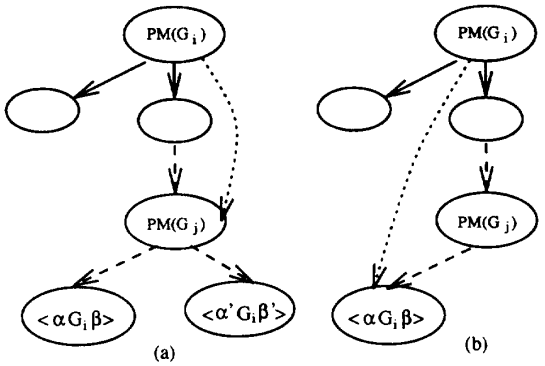

Figure 5: The Optimization of Propagation Trees

If there is no intermediary which has two (or more) children leading to any G_i's meta-group before reaching $<\alpha G_i\beta>$ as shown in Fig. 5(b), we simply link $<\alpha G_i\beta>$ to $PM(G_i)$. Messages for group G_i can be passed down to

$<\alpha G_i\beta>$ directly from $PM(G_i)$. Since there is no other G_i's meta-group branched off from the intermediaries, the messages merged in at the intermediaries can be ordered with G_i's messages at $<\alpha G_i\beta>$ and then passed down. There is no danger that any meta-group above $<\alpha G_i\beta>$ can receive these merged-in messages and G_i's messages in the different order from $<\alpha G_i\beta>$ (actually there is no meta-group above $<\alpha G_i\beta>$ even can receive both these merged-in messages and G_i's messages).

Back to the example in Fig. 4, $<AD>$ is the intermediary from $<ABC>$ to $<CD>$. $<CD>$ does not have other subtrees leading to other C's meta-groups, except to $<CD>$. We link $<CD>$ directly to $<ABC>$. Group C's messages and group D's messages, to their common members, would be ordered at $<CD>$ (by its primary site), bypassing $<AD>$. There is no other meta-groups branched off from $<AD>$ can receive both C's messages and D's messages. Messages for C and D would always be received in the same relative order.

5. Algorithm for Message Propagation

The propagation trees specifies the paths of message flow from source to all the destinations. A multicast server (MS) residing at each site is responsible for actual message passing along the propagation trees, and delivering messages to the local user processes if they are in the destination groups. An MS receives messages from both users which request multicasting messages and other MSs which request propagating messages further to the final destinations.

For serving users' multicast requests, each MS keeps information about the location of the primary meta-group for each multicast group. When a user multicasts a message to a group, it traps into the local MS. Then the MS passes the message to the PM of the group through point-to-point communication. From this PM, the message would be propagated to all the destinations along the tree. When an MS receives a message from another MS, it acts as the following:

1. If it is the final destination of the message, it delivers the message to the local user processes which are the members of the group.

2. If it is the primary site of a meta-group which is a subset of the destination group, it multicasts the message to the other members of its meta-group. A primary site of a meta-group maintains the memberships of the meta-group.

3. If it has any child meta-groups leading to the members of the destination group, it multicasts the message to those child meta-groups (the primary sites of them). An MS which is the primary site of the PM of some multicast groups keeps a list of its child meta-groups. Each element in the child list records the multicast groups it is heading for, so that given the destination group of a message the MS knows which child meta-groups it should pass the message to.

We mentioned before that multicasting to individual groups with only *SSSG* ordering could be done by the underlying network hardwares/protocols. If this support is not available, each MS maintains sequence numbers for each site to which it sends messages, and records a sequence number for each MS from which it receives messages. This guarantees that a receiver can order the messages from a sender correctly if they arrive out of order. When a site receives a message, it checks the sequence number against the number it expects from that sender. If they do match, the message is in correct order. If they do not match, the message would be queued up waiting for the arrival of earlier ones. This method also allows the receiver to detect missing messages. Thus, our multicast protocol guarantees both total ordering and the reliable delivery of messages.

6. Performance Evaluations

There are several important criteria to evaluate a multicast protocol. One is the number of messages required to multicast a messages to all group members. Another is the latency time for a message to be received by all group members. And finally, the flexibility of membership changes among multicast groups. We will compare our protocol with other approaches based on these criteria.

Consider the message cost first. In a centralized mechanism, one message is required to the central site and n messages from the central site to the n group members. The token passing mechanism requires another L-1 messages to pass the token to another L -1 site before delivering the messages to users, making the total number of messages up to $n + L$. A two-phase based algorithm, like *ABCAST*, requires three rounds of message exchanges, n messages to the destinations initially, another n messages carrying local timestamps back to the sender from destinations, and the final n messages to the destinations for commitment . In total, $3n$ messages. In Garcia's propagation tree method, it requires $n + \varepsilon$ messages, where ε is the number of extra nodes leading to destinations on the tree. The number of messages required in our mechanism is similar to Garcia's method. But it is improved by the optimization of propagation trees, which reduces the extra nodes (intermediaries) considerablely.

To compute the latency time, we assume the underlying multicast network is available, by which a message sent to multiple sites takes the same order of time as sent to one site. Suppose the time delay of sending a message to a destination by the network is T. The centralized

mechanisms need $2T$ to have the message received by all destinations, one T to the central site and another T to the destinations from the central site. The delay for the token passing mechanism is $(L + 1)*T$, an extra $L-1$ unit delay for passing the token $L - 1$ times. A two-phase based algorithm requires at least $3*T$ to complete a multicast operation, one T for messages to all the receivers, at least another T for receivers replying to the sender (notice that the sender receives the n replies sequentially, which is the bottleneck of performance), and another T to all the receivers finally. In Garcia's method, it takes $(d + 1)*T$ to propagate messages to all the members, where d ($d \geq 2$) is the depth of the subtree of the corresponding multicast group. In our mechanism, the depth of the subtree of a multicast group is much smaller than the tree in Garcia's method, because the nodes on our trees are meta-groups, not sites as in Garcia's method. If there are no intermediaries, the depth of a subtree in our mechanism is 2 (one primary meta-group as the subroot and a level of child meta-groups). It usually takes $3*T$ to propagate a message to all members, one T to the primary meta-group, another T from primary meta-group to its child meta-groups, and another T to the members of meta-groups.

In a distributed system, processes often leave or join a group. A multicast protocol must allow the changes of group memberships to be done efficiently. The centralized and distributed protocols discussed do not have particular difficulties for the membership changes, because the message delivery mechanisms do not rely on the group memberships of processes. The construction of propagation trees in Garcia's protocol depends on the membership of each process. Each change of a group membership would result in the restructuring of the propagation tree, which is very costly. The propagation tree in our mechanism is built from meta-groups. The change of a group membership does not change the structure of the propagation tree provided it does not lead to the deletion or creation of a meta-group. Thus, it has a greater flexibility to changes of group memberships.

References

[1] Birman, K.P. & Joseph, T.A., Reliable Communication in the Presence of Failures, *ACM Transactions on Computer Systems* 5, 1 (Feb. 1987), pp. 47-76.

[2] Birman, K., Schiper, A. & Stephenson, P., Lightweight Causal and Atomic Group Multicast, *ACM Transactions on Computer Systems* 9, 3 (Aug. 1991), pp. 272-314.

[3] Chang, J. & Maxemchuk, N.F., Reliable Broadcast Protocols, *ACM Transactions on Computer Systems* 2, 3 (Aug. 1984), pp. 251-273.

[4] Dasser, M., TOMP: A Total Ordering Multicast Protocol, *Laboratorire d'Iformatique Technique*, Lausanne, Switzerland, 1992, pp. 32-40.

[5] Garcia-Molina, H. & Spauster, A., Ordered and Reliable Multicast Communication, *ACM Transactions on Computer Systems* 9, 3 (Aug. 1991), pp. 242-271.

[6] Joseph, T.A. & Birman, K.P., Reliable Broadcast Protocols, *Lecture Notes of Arctic 88* - An Advanced Course on Operating Systems, Tromso, Norway, Jul. 1988, pp. 293-317.

[7] Kaashoek, M.F., Tanenbaum, A.S., Hummel, S.F. & Bal, H.E., An Efficient Reliable Broadcast Protocol, *Operating Systems Review* 23, 4 (1989), pp. 5-19.

[8] Liang, L., Chanson, S.T. & Neufeld, G.W., Process Groups and Group Communications: Classifications and Requirements, *IEEE Computer* 23, 2 (Feb. 1990), pp. 56-66.

[9] Navaratnam, S., Chanson, S. & Neufeld, G., Reliable Group Communication in Distributed Systems, *In Proc. of the 8th IEEE Inter. Conf. on Distributed Computer Systems*, June 1988, pp. 439-446.

[10] Schiper, A., Eggli, J. & Sandoz, A., A New Algorithm to Implement Causal Ordering, In Proceedings of the 3rd International Workshop on Distributed Algorithms, *Lectures Notes on Computer Science 392, Springer-Verlag*, New York, 1989, pp. 219-232.

[11] Tanenbaum, A.S, M.F. Kaashoek, H.E.Bal, Parallel Programming Using Shared Objects and Broadcasting, *IEEE Computer*, Aug. 1992, pp.10-19