A Fast Group Communication Mechanism for Large Scale Distributed Objects[1]

Hojjat Jafarpour and Nasser Yazdani

Department of Electrical and Computer Engineering
University of Tehran
Tehran, Iran
hjafarpour@ece.ut.ac.ir, yazdani@ut.ac.ir

Abstract. Group communication undoubtedly is a useful paradigm for many distributed object systems. Provision of a fast and scalable group communication mechanism can considerably improve the efficiency of these systems. In this paper, we propose a new group communication mechanism for large scale distributed objects over a huge heterogeneous inter-network. The proposed mechanism uses Message Dispatcher Objects, objects co-located with each group member to distribute and parallelize the group communication load among all group members. Our proposed mechanism also recovers from the host failure in the system. Experimental results show that our scheme scales well in a large number of group members.

1 Introduction

Group or multicast communication is an important communication model in distributed systems. A group can be a set of processors or objects. Group communication service's responsibility is to deliver messages to all the group members [2]. Indeed, they are powerful building blocks that facilitate the development of many distributed applications [1].

One of the main applications of group communication services is to provide fault tolerant capabilities for distributed object systems such as CORBA. These systems achieve fault tolerance and highly availability by means of object replication. They use group communication services to maintain consistency of replicated objects. For instance, OMG's Fault-Tolerant CORBA specification [3] recommends using group communication services to support active object replication in CORBA.

Group communication mechanisms are used in applications such as distributed transactions and database replication [4], highly available services [5], support for distributed and clustered operating systems [6] and collaborative computing [7].

[1] This work is supported as a research project by the Faculty of Engineering, University of Tehran, Tehran, Iran

R. Meersman and Z. Tari (Eds.): OTM Workshops 2003, LNCS 2889, pp. 1036–1044, 2003.

Many group communication mechanisms for distributed systems have been proposed among them to mention Totem [8], Hourus [9], Transis [10] and Spread [11].

Among the main requirements of these systems are scalability of the group communication mechanism and low message delivery time. These features can considerably improve the performance and efficiency of applications using them. Unfortunately, majority of previously proposed mechanisms do not scale well with respect to the number of the group members on a huge heterogeneous inter-network like the Internet.

In this paper, we propose a fast and scalable group communication mechanism for distributed objects on a heterogeneous inter-network using application level communication techniques such as remote method invocation. The proposed approach achieves scalability and high message delivery speed by distributing and parallelizing the multicasting load among hosts in the system. It uses Message Dispatcher Objects (MDOs), which are objects co-located with each group member, to route and deliver multicast messages to the group members. A message is sent to a MDO which is the proxy of the group and is disseminated among MDOs using an efficient algorithm. Then, each MDO delivers the message to its co-located group members. The proposed mechanism also detects host failure and recovers from it.

The rest of the paper is structured as follow. Section 2 describes the proposed group communication mechanism. Section 3 presents how the mechanism recovers from host failure. In section 4, some characteristics of the mechanism and experimental results are presented. Section 5 reveals some directions for future work and concludes the paper.

2 The Proposed Group Communication Mechanism

2.1 System Model

The proposed mechanism assumes that group members are on different hosts in a huge heterogeneous inter-network such as the Internet and there is a logical path between every two hosts. The communication channels between every two hosts are reliable and communications are done using application level techniques such as remote method invocations.

Message Dispatcher Objects (MDOs): The main components of the mechanism are special objects co-located with group members which we call them MDOs. The mechanism supposes there is one MDO on each host in the system. Each MDO knows all MDOs and their addresses. This information is stored in a list called *MDO list*. The index of each MDO is the same in all MDO lists in the system. Figure 1 depicts a MDO list for a group with 16 members on 16 hosts. In such a system, there is one MDO object on each host and all of them have the depicted list. The first MDO in the list is called the *proxy MDO*. Each MDO also has a message storage queue that incoming messages are stored in. Using this queue, MDOs store a message for a limited period of time to ensure that all group members have received the message. If there are more than one group member on a host, its MDO knows all of them too.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MDO ID	MDO 0	MDO 1	MDO 2	MDO 3	MDO 4	MDO 5	MDO 6	MDO 7	MDO 8	MDO 9	MDO 10	MDO 11	MDO 12	MDO 13	MDO 14	MDO 15
Address	Host 0	Host 1	Host 2	Host 3	Host 4	Host 5	Host 6	Host 7	Host 8	Host 9	Host 10	Host 11	Host 12	Host 13	Host 14	Host 15

Fig. 1. MDO list which is the same for all MDOs in a system with 16 group members on 16 hosts.

2.2 Message Propagation Mechanism

When a group member wants to send a message to the group it does not send the message to the members directly. Instead, it sends the message to one of the MDOs which is called the proxy MDO. Then, MDOs propagate the message among themselves. In the next step each MDO delivers the message to its local group member(s). Message dissemination among MDOs is done in a parallel manner and the propagation load is distributed among hosts in the system.

The message propagation process among MDOs is as follow. Suppose the proxy MDO receives a message to propagate among MDOs. It first sends the message to one of the MDOs. Now two MDOs have the message, the proxy and the MDO, which just received the message from the proxy. Thus the second MDO can also participate in message propagation process. In the next step, both MDOs send the message to two other MDOs. Therefore the number of MDOs which have the message will be four. Then, all the four MDOs deliver the message to four other MDOs and this process continues until all MDOs receive the message. As can be seen, the number of MDOs which know the message is doubled after each step and each MDO receiving the message participates in message delivery. The mechanism does not assume any particular service in the underlying network and can be applied in all kinds of heterogeneous inter-networks.

2.3 Message Propagation Algorithm

In order to find the proper MDO to deliver the message in each step and to prevent double message delivery to an MDO, MDOs use an algorithm to determine the MDO which they should send the message to in each step of message propagation. The algorithm is shown in figure 2. The algorithm in each iteration works with a portion of the MDO list which is called *customized MDO list*. The customized MDO list is determined by two boundary indices and involves the MDOs which are between these indices in the original MDO list.

As mentioned, the proxy MDO receives the message first. The customized MDO list for the proxy in the first step of the message delivery is the original MDO list except itself. The proxy finds the median of its customized MDO list using the formula depicted in 4th step of the algorithm. Then, it sands the message and the boundary indices of the second half of the customized MDO list, which are calculated in the

6th step of the algorithm, to the median MDO, and makes it responsible for propagating the message among MDOs located between indicated indices in the list, using the same algorithm. In the next iteration, the proxy continues executing the algorithm using the first half of the customized MDO list as its new customized MDO list. The boundary indices of the new customized MDO list are calculated in the 5th step of the algorithm. Figure 3.a shows how a message is propagated among 16 hosts. The message delivery process in figure 3.a can be depicted as a special tree structure. Figure 3.b shows the message propagation tree in the sample system which presents a parent child relationship between MDOs.

Each MDO does the following steps after receiving a message.

Suppose the number of MDOs is $\boldsymbol{n}$ *and the boundaries of the Customized MDO list are (* $\boldsymbol{a}$ *,* $\boldsymbol{b}$ *)*

1. *Get the message and the boundaries for the MDO list and calculate the customized MDO list.*

2. *If there is no boundaries*
   a. *If you are the proxy MDO and your position in the MDO list is* $\boldsymbol{p}$ *set the boundaries as* $\boldsymbol{a} = (\boldsymbol{p}+1) \text{ MOD } \boldsymbol{n}$ *and* $\boldsymbol{b} = (\boldsymbol{p}-1) \text{ MOD } \boldsymbol{n}$.
   b. *Else finish*

3. *If* $\boldsymbol{b}-\boldsymbol{a} \text{ MOD } \boldsymbol{n} < 2$ *send the message to the MDOs which are at indices* $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *and finish.*

4. *Find the median component of the customized list. Assume its position in the customized list is* $\boldsymbol{m}$ *then*
   $\boldsymbol{m} = (\boldsymbol{a} + ((\boldsymbol{b}-\boldsymbol{a}) \text{ MOD } \boldsymbol{n})/2) \text{ MOD } \boldsymbol{n}$

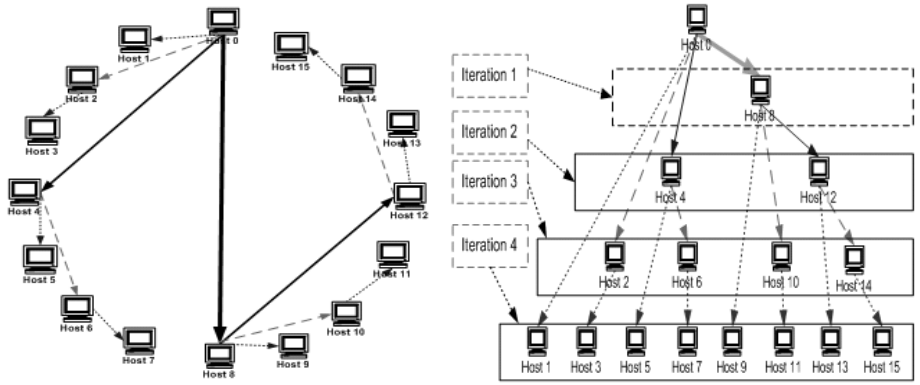
5. *Calculate the boundaries of the new customized list, which is the first half of the current list as follow:*
   $\boldsymbol{New\_a} = \boldsymbol{a}$, $\boldsymbol{New\_b} = (\boldsymbol{m}-1) \text{ MOD } \boldsymbol{n}$

6. *Send the message and the following boundaries to the MDO, which is at index* $\boldsymbol{m}$ *in the old MDO list.*
   $\boldsymbol{a\_for\_Median} = (\boldsymbol{m}+1) \text{ MOD } \boldsymbol{n}$, $\boldsymbol{b\_for\_Median} = \boldsymbol{b}$

7. *Set* $\boldsymbol{a} = \boldsymbol{New\_a}$, $\boldsymbol{b} = \boldsymbol{New\_b}$ *and go to step 1.*

Fig. 2. MDO selection algorithm

When a MDO sends the message to all of its corresponding MDOs and finishes the iterations of message propagation among MDOs, it delivers the message to its co-located group member(s). MDOs which are leaves of the tree after delivering the message to their co-located group member(s) send acknowledge to their parent MDO. Each parent MDO after delivery of the message to its local group member(s) and receiving acknowledge from its entire child MDOs sends acknowledge to its parent MDO. The acknowledge indicates that all group members co-located with it and other MDOs in its sub-tree have received the message. For instance, an acknowledge for a message from Host12 indicates that all group members and MDOs on Host12, Host 13, Host 14 and Host15 have received the message.



a) Message propagation among MDOs b) Message propagation tree

Fig. 3. Message propagation among MDOs

When the proxy MDO receives acknowledge from all of its child MDOs, it knows that all group members have received the message and there is no need to save the message in MDOs anymore. Then, the proxy MDO informs all other MDOs to delete the message from their message queue using a piggybacking method and the next incoming message. In other words, the proxy attaches the sequence number of the message to the next incoming message and propagates it among all MDOs. Each MDO after receiving the new message deletes the previously delivered message, which its sequence number is attached to the message, from its message queue. Then it continues the propagation of the new message using the mechanism.

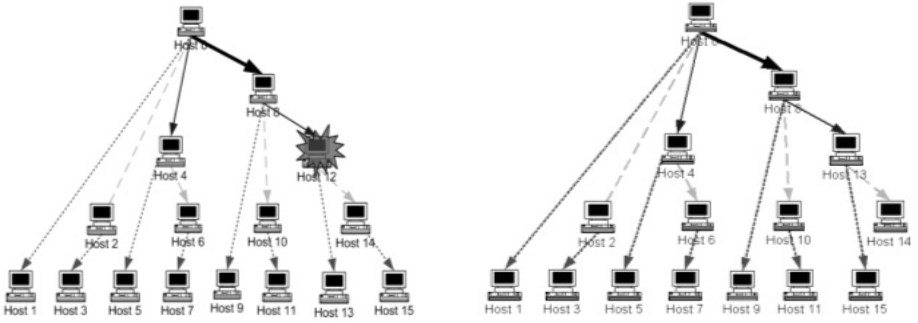
3 Recovery from the Host Failure

The proposed group communication mechanism can detect the host failure in the system and recover itself from it. We suppose that the failure model of hosts in the system is *fail-stop* [2] which can be detected by other MDOs. Consequently, if a

MDO wants to send a message to another MDO on a failed host the sender can detect the failure. Host failure can be categorized as the following.

1. Failure before receiving a message.
2. Failure after receiving a message and before sending acknowledge for it.
3. Failure of the proxy MDO before receiving a message.
4. Failure of the proxy after receiving a message and before receiving acknowledges from its entire child MDOs.

For the sake of simplicity, we suppose that just one host fails in the system. In case of multiple host failure, the mechanism takes the same steps for recovery. The recovery mechanism uses acknowledges and timeouts to recover. After receiving a message and before sending it to its child MDOs, each MDO sets a timer. The timer value is calculated using the size of its first customized MDO list.



a) Host12 has failed before receiving the message b) Propagation tree for recovery

Fig. 4. Recovery from failure of host 12.

In the first category of the failure, when a host fails before reception of a message by its MDO, its parent MDO which wants to send the message to it detects its failure. The parent MDO first informs its parent about the failure and requests it to reset its timer. Each MDO after reception of the timer reset request resets its timer and sends the request to its parent MDO. This process continues until reaching to the proxy. Then, the parent of the failed MDO selects the neighbor MDO of the failed MDO, which is in the next index in the customized MDO list, as the new median and continues the process. It also informs the proxy MDO about the failure. The proxy MDO informs all other MDOs about the failure using the same message propagation mechanism. Each MDO after reception of the failure report first deletes the failed host from its MDO list and then uses the new list to participate in informing MDOs about the failure. Figure 4 shows the message propagation tree before and during the

recovery process from failure of host 12 in the previous example. The propagation tree after recovery is shown in figure 5.

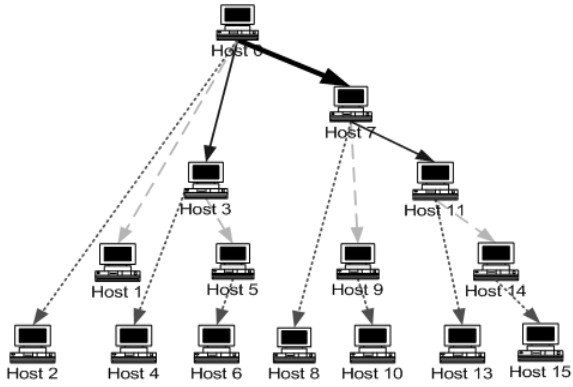


Fig. 5. Message propagation tree after recovery from failure.

For recovery from the second types of failures, the mechanism uses timers. As mentioned previously, before sending a message to its child MDOs, each MDO sets a timer. If the MDO does not receive acknowledge from its entire child MDOs before the expiration of the timer, it tries to detect which of the MDOs has not sent acknowledge. The MDO first sends a timer reset request to its parent as the previously described method. Then, it recovers from the failure using the same process described for the first type of failure.

Recovery from the proxy failure requires some different steps. If the host of proxy MDO fails before receiving a message the sender informs its local MDO about the failure and delivers the message to it. The MDO then chooses the second MDO in the MDO list, which is the neighbor of the proxy in the list, as the new proxy. Then, the new proxy deletes the failed MDO from its list and inform all other MDOs to delete the failed proxy from their MDO list using the same message propagation mechanism. Next, the message is sent to the new proxy by the MDO.

The last form of failure is detected by the child MDOs of the failed proxy. When the child MDOs want to send acknowledge to their parent MDO the failure of the proxy is detected. Then, the detector MDO chooses the second MDO in the list as the new proxy and informs all other MDOs about it using the previously described method. In the next step, the new proxy asks all MDOs to find out which messages have not been delivered to all group members and then resends them to all MDOs. All communications in these sections are done using the same group communication mechanism.

4 Some Characteristics of the Mechanism

Message Delivery Time. The proposed approach reduces multicast message delivery time in large scale systems. As mentioned before, the message delivery operation among MDOs is done in a parallel manner, which considerably improves multicasting speed. To calculate the maximum amount of time taken to disseminate a message among all MDOs, we can use formula (1).

$$\text{Maximum Time for Propagation Among MDOs} = \text{MMTT} * \log_2 (N_{MDOs}) \quad (1)$$

In the formula MMTT indicates the maximum message transfer time from one host to another and N_{MDOs} shows the number of MDOs in the system which is equal to the number of hosts that have group members. The maximum amount of time takes to deliver a message to all group members can be achieved by just adding message delivery time to group members by MDOs to the previously calculated amount.

$$\text{MaximumDeliveryTime} = \text{MMTT} * \text{Log}_2 (N_{MDOs}) + \text{LMDT} * N_{Local\_Members} \quad (2)$$

In formula 2, LMDT indicates the local message delivery time and $N_{Local\_Members}$ indicates the maximum number of group members in one host.

Open and Close Groups. The groups can be open or close. In an open group, objects, which are not group members, can send message to the group. In contrast, in a close group only group members can send messages to the group [2]. To support an open group, we make the proxy accessible from the outside. Then, messages can be sent to the proxy to be delivered to the group members.

Transparent Multicast. Transparent message multicasting is another property of our mechanism. In transparent multicast, the sender need not to know the group members and it just sends a message to the group and the mechanism will deliver the message to all group members [2]. This is a very important characteristic that makes implementation of the sender and the group members easy.

For the sake of simplicity, we supposed that each host sends one message at each time. By some changes in the mechanism the method can exploit multithreading capabilities of hosts to send more than one message simultaneously. For instance suppose each host can send 3 messages at the same time. Thus, each MDO instead of dividing its customized list into two halves by finding median; divides it into four sections and finds three MDOs to send the message. Consequently, message propagation among MDOs can be done in fewer steps.

We have tested our mechanism with 16 hosts and with one member on each host. The results have been compared to the mechanism that sequentially sends a message to all group members. On a 100Mbps LAN and with a 20KB message the proposed mechanism delivered the message to all group members in about 420 ms while the other one did the same task in about 1700 ms.

5 Conclusions and Future Work

We proposed a group communication mechanism for large scale distributed objects on heterogeneous inter-networks. The proposed mechanism disseminates message among group members in a considerably low time without any special assumption about the underlying network. Using special objects called Message Dispatcher Objects (MDOs), which are co-located with each group member; the mechanism parallelizes message propagation by distributing the propagation load among all hosts. Our proposed mechanism also detects host failure in the systems and recovers from it.

One of the main parts of the future work is devoted to issues such as membership protocols and recovery from other kinds of failures like network partitioning. More comprehensive evaluations and comparison of the mechanism with other similar mechanisms are planned to be done.

References

1. Gregory V. Chockler, Idit Keidar, and Roman Vitenberg. : Group Communication Specifications: A Comprehensive Study. In ACM Computing Surveys 33(4), pages 1–43, December 2001.
2. G.Coulouris, J. Dollimore and T. Kindberg. : Distributed Systems – Concepts and Design, 3rd edition Addison-Wesley, 2001.
3. OMG. Fault Tolerant CORBA Specification. OMG document ptc/2000-04-04, 2000
4. Schiper, A and Rynal, M.: From Group Communication to Transactions in Distributed Systems. In Communications of the ACM 39(4), pages 84–87, April 1996.
5. Fekete, A. and Kediar, I.: A Framework for Highly Available Services Based on Group Communication. In IEEE 21st International Conference on Distributed Computing Systems Workshops(ICDCS-21W 2001); the International Workshop on Applied Reliable Group Communication. April 2001
6. Goft, G and Yeger Lotem, E.: The AS/400 Cluster Engine: A Case Study. In IGCC 1999 in conjunction with ICPP 1999.
7. Birman, K., Friedman, R., Hyden, M. and Rhee, I.: Middleware Support for Distributed Multimedia and Collaborative Computing. In Multimedia Computing and Networking (MMCN98) 1998.
8. L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos.: Totem: A fault-tolerant multicast group communication system. Communications of the ACM, 39(4): 54–63, April 1996.
9. van Renesse,R., Birman, K. and Maffeis,S.: Hourus: a Flexible Group Communication System. Communications of the ACM, 39(4), 76–83, April 1996.
10. Dolev,D. and Malki,D.: The Transis approach to high availability cluster communication. Communications of the ACM, 39(4), 64–70, April 1996.
11. Yair Amir and Jonathan Stanton.: The Spread Wide Area Group Communication System. Technical Report CNDS-98-4.Johns Hopkins University. 1998