Provision of Recovery from Host Failure for Sama Group Communication Middleware for Mobile Agents

Hojjat Jafarpour and Nasser Yazdani
Dept. of Electrical & Computer Engineering
University of Tehran
Tehran, Iran
hjafarpour@ece.ut.ac.ir , yazdani@ut.ac.ir

Abstract

A high speed group communication middleware can considerably improve performance of multi-agent systems. Sama[1] is a fast and scalable group communication middleware for mobile agents. It achieves scalability and low message delivery time by distributing message dissemination load among all mobile agent servers. Sama uses Message Dispatcher Objects (MDOs), which are special objects on each agent server, to parallelize message propagation process. In this paper, we describe the Sama group communication middleware and its recovery from host failure feature. We categorize host failures in the mechanism and show how Sama overcomes these failures. Using these features Sama can provide more reliable message propagation infrastructure for mobile agent groups.

1. Introduction

Mobile Agents are executing programs or objects that can migrate from one machine to another in a heterogeneous network to perform tasks on behalf of their user [1]. This characteristic of mobile agents have made them suitable to be used in various distributed applications like distributed information retrieval [2], network management [3], E-commerce [4] etc. Many mobile agent platforms such as Voyager [5], Aglet [6] and Grasshopper [7] have been developed to provide facilities for developing mobile agent based applications.

Undoubtedly, communication among agents plays a crucial role in many mobile agent applications. Different communication models such as broadcasting, forwarding and central server have been proposed [8]. An important communication model which is used in many multi-agent applications is group communication. Scalability and speed of a group communication mechanism can definitely improve performance. Applications like E-commerce or distributed simulations [9] can easily have hundreds of mobile agents distributed over a continental distance requiring a scalable and fast mechanism for communication among agents.

Sama [10] is a fast and scalable group communication middleware for mobile agents. It provides a message propagation infrastructure on heterogeneous internetworks such as the Internet which involves all mobile agent servers in the system. By mobile agent server, we mean programs that are run on hosts in order to enable them to accept messages or requests for mobile agents. Consequently, only the hosts that have agent servers, which our agents can migrate on, are considered in our mechanism. The building blocks of this infrastructure are special objects on each agent server called Message Dispatcher Objects (MDOs). Sama achieves scalability and rapid message delivery by distributing and parallelizing message propagation process among all MDOs. Sama propagates a message among a group of agents using constant number of remote messages respect to the number of mobile agent servers in the system. Message delivery in Sama is independent of agents' locations.

In addition to scalability and high message propagation speed, dealing with failures also has an important role in a group communication mechanism. A common type of failure in heterogeneous internetworks is Host Failure. Unfortunately, majority of previously proposed mechanisms do not provide fault tolerance features to recover from this kind of failures.

In this paper, we propose the methods that enable Sama to deal with host failures in the system. We categorize different types of host failures in Sama and show how it recovers from each of them. Using these new features, Sama can provide more reliable communication for mobile agent groups.

The rest of the paper is organized as follows. In the next section, we provide an overview on Sama group communication middleware. Section 3 describes how Sama recovers from host failures. In this section we

[1] This work has been supported by the Univ. of Tehran as a research project.

propose different host failure categories in Sama and methods for recovering from them. Section 4 reviews related work. Section 5 provides some directions for future work and concludes the paper.

2. Sama Group Communication Middleware for Mobile Agents

Sama provides group communication infrastructure on a heterogeneous internetwork. It does not assume any special feature for the underlying network and communications are done using application layer mechanisms such as Remote Method Invocation.

Message Dispatcher Objects (MDOs): The main components of the mechanism are special objects on each mobile agent server, which we call them MDOs. They can be assumed as a part of mobile agent servers which are running on some of the hosts in the internetwork. Each MDO knows all MDOs and their addresses in the system. This information is stored in an array which is called MDO List. It also knows MMTT, the Maximum Message Transfer Time, and MAMT which is the Maximum Agent Migration Time between two hosts. These values are used to calculate threshold values to detect failures. Each MDO also has a message storage queue to store incoming messages.

Using this queue, MDOs store a message for a limited period of time to ensure that all group members, specially migrating members during message propagation, will receive them. Each MDO also has the list of all its co-located group members, which is called Local Agent List. MDOs provide facilities for migrating agents to register and un-register in their Local Agent List. A group member before migration should un-register itself from the list of the source MDO and after migration it should register itself in the list of destination MDO. Messages not received by a migration agent are delivered to it by the destination MDO.

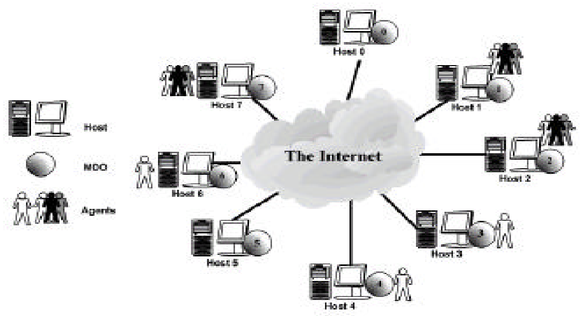

Figure 1. A sample system with 8 hosts with agent servers.

The first MDO in the MDO List is called the Proxy MDO. It coordinates all MDOs in the system and plays the role of the group proxy. The proxy MDO also assigns sequence numbers for incoming messages which are used by MDOs to detect the correct order of received messages.

Figure 1 depicts a sample system with 8 hosts. As it can be seen, hosts that have mobile agent servers and agents can migrate on them are connected through the Internet and there is one Message Dispatcher Object (MDO) on each host. There also exist one or few mobile agents on some of these hosts.

Message Propagation Mechanism: In Sama, 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 the group proxy via its local MDO and MDOs propagate the message among themselves. Then each MDO delivers the message to its local group members using its Local Agent List.

```
Each MDO does the following steps after receiving a message.
Suppose the number of MDOs is n and the boundaries of the
Customized MDO list are ( a , 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 first receiver MDO and your position in
          the MDO list is p set the boundaries as a = (p+1)
          MOD n and b = (p-1) MOD n.
      b.  Finish
3.  If b-a MOD n < 2 send the message to the MDOs which are at
    indices a and b and finish.
4.  Find the median component of the customized list. Assume its
    position in the customized list is m then
          m = (a + ((b-a) MOD n)/2) MOD n
5.  Calculate the boundaries of the new customized list, which is
    the first half of the current list as follow:
                a = a , b = (m-1) MOD n
6.  Send the message and the following boundaries to the MDO,
    which is at index m in the old MDO list.
                a = (m+1) MOD n , b = b
7.  Go to step 1.
```

Figure 2. Tree Generating Algorithm

Message dissemination among MDOs is done in a parallel manner and the propagation load is distributed among hosts in the system. It can be described as follow. Suppose that 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 and 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 will continue until all MDOs receive the message. As it can be seen the number of MDOs which know the message is doubled after each step and each MDO that receives the message participates in message delivery. In fact, messages are propagated among MDOs using a binomial tree structure and then each MDO delivers the message to its local mobile agents.

To construct the binomial tree for message dissemination, each MDO runs a distributed tree generating algorithm using its MDO List which contains information about all MDOs in the system and is the same for all MDOs. The algorithm is shown in figure 2. The customized MDO List in the algorithm is a portion of the main MDO List. Figure 3 shows the message propagation tree among MDOs in a system with 16 MDOs and MDO0 as the proxy of the group. As can be observed there, in each step, the number of MDOs that receive the message is doubled and the number of communication steps required for dissemination of a message among all MDOs is [log_2 n] where n is the number of MDOs in the system. A communication step is the time required for a message to be sent form one MDO to another.

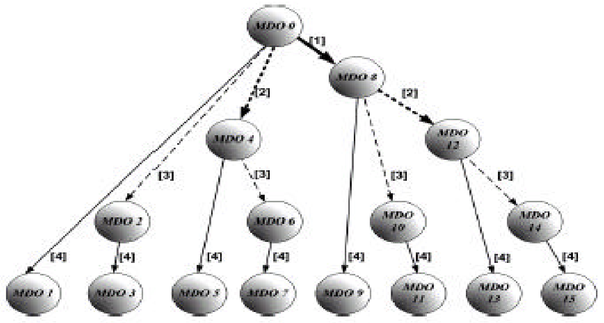

Figure 3. Message Propagation Tree among MDOs

3. Recovery from Host Failure

To ensure reception of a message by all group members, Sama uses an acknowledgement mechanism. MDOs, which are leaves of the message dissemination tree, after receiving a message and delivering it to their local group members using their Local Agent List, inform their parent MDO in the tree by sending an acknowledgement. Each non-leaf MDO also, after delivering the message to its local group members and receiving acknowledgements from its entire child MDOs in the tree, sends an acknowledgement to its parent MDO. This acknowledgement informs the parent about correct propagation of the message among all MDOs located in the sub-tree rooted by the child MDO. Finally, reception of acknowledgements from its entire child MDOs by the proxy MDO indicates that all the MDOs have received the message correctly. However, MDOs store the message for a limited period of time to ensure that the migrating group members have also received it. This period of time is calculated using MMTT and MAMT values and should not be less than the following amount.

1 $$\mathbf{MMTT * (\log_2 N_{MDOs}) + MAMT}$$

In formula 1, N_{MDOs} presents the number of MDOs in the system.

Each MDO before sending a message to its child MDO sets a timer. The child MDOs should send the acknowledgement before the expiration of the timer in its parent. If an MDO does not receive acknowledgement from at least one of its child MDOs after expiration of the timer it infers that there is a failure in the sub-tree rooted by the child MDO. The timer value is calculated using the MMTT and the first Customized MDO List in the tree generating algorithm. The value is an estimation of the time that all child MDOs should send the acknowledgement to their parent MDO. MDOs use the following formula to calculate the timer value for their child MDOs in the tree generation algorithm.

2 $$\mathbf{Timer >= 2*MMTT*(\log_2 FirstCustomizedListSize) + LMDT * N_{Agents}}$$

In formula 2, LMDT is the maximum amount of time that takes an MDO delivers the message to one of its local group members and N_{Agents} is the number of group members. Formula 2 can be inserted in the tree generation algorithm.

Using the mentioned features Sama can detect host failures in the system and recover from them. Regarding the location of an MDO in the propagation tree and the failure time, we categorize host failures in the system into four different groups.

1. Host failure before and during receiving a message
2. Host failure after receiving a message and before sending acknowledgement for it.
3. Host failure of the proxy MDO before and during receiving a message
4. Host failure of the proxy MDO after sending a message and before receiving its acknowledgements

3.1. Host Failure before Receiving a Message

As mentioned in section 2, communications in Sama are performed using application layer mechanisms such as Remote Method Invocation. Our mechanism interprets failure in establishing this kind of connection as the failure of the receiver host. Consequently, when a host fails in the system, its parent MDO can detect this failure when it wants to send a message to the failed MDO. After detection of the failure, the parent MDO first informs its own parent about the failure and requests it to reset its timer. By receiving a timer reset request after resetting its timer, every MDO sends a timer reset request to its own parent MDO until the request reaches to the root of the tree, the proxy MDO. Then the failure detector MDO sends the message to the MDO which is located next to the failed MDO in the Customized MDO List.

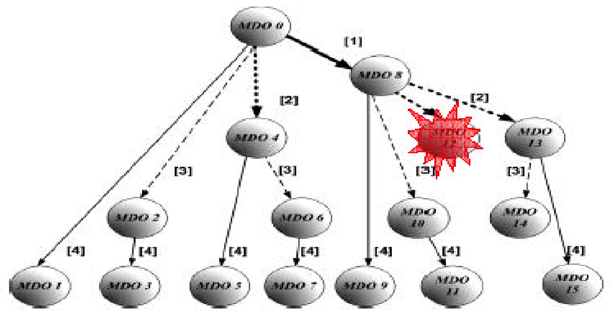

Figure 4. Message Propagation Tree during recovery form failure of Host 12

After sending the message, the detector MDO informs the proxy MDO about the failure by sending the name of the failed MDO. The proxy first deletes the failed MDO from its MDO List and then generates a correction message to inform all other MDOs about the failure. The correction message is propagated among all MDOs using the same group communication mechanism. Every MDO after receiving the message first removes the failed MDO from its MDO List and then executes the tree generating algorithm to construct the new message propagation tree using the new MDO List.

Figure 4 shows the message propagation tree during the recovery from the failure of Host 12 in the previous example. As it can be observed, MDO 8 after detecting the failure of Host12 sends the message to MDO 13 which is located next to the failed MDO in the MDO List. After propagation of the correction message and removal of the failed MDO from all MDO Lists, the new message propagation tree will be generated as shown in figure 5.

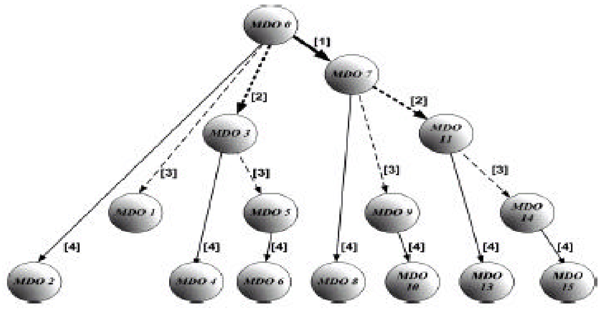

Figure 5. Message Propagation Tree after recovery form failure of Host 12

3.2. Host Failure after Receiving a Message and before Sending Acknowledgement for It

To recover from the second category of failures, our mechanism uses timers. As explained previously, before sending a message to its child MDO, each MDO sets a timer. If the MDO does not receive the acknowledgement from its entire child MDOs before expiration of the timer, it infers that there is a failure and finds the MDO that has not sent acknowledgement. The MDO first resets its timer and then sends timers reset request to its parent MDO. All the MDOs in the path from the detector to the proxy MDO in the root of the tree reset their timers by receiving requests from their child MDOs. The MDO then resends the message to the failed MDO. Now the failure is similar to the first category of failure and the mechanism recovers from the failure using the same process described in the previous subsection. In this kind of recovery, the MDOs can detect duplicate messages using the message sequence numbers.

3.3. Host Failure of the Proxy MDO before Receiving a Message

Our middleware deals with the failure of the proxy in a different way. When an agent wants to send a message to the proxy in order to be propagated among group members, it can detect the failure of the proxy MDO. Then the agent informs its local MDO about the proxy failure and delivers the message to it. The local MDO informs the first available MDO after the failed proxy in the MDO List about the proxy failure and sends the message to it and makes it the new proxy for the group. The new proxy first removes the failed proxy from its MDO List. Then generates a correction message and informs all other MDOs about the proxy failure by sending the correction message to them using the new MDO List. Every MDO after receiving the correction message first deletes the failed proxy from its MDO List and then cooperate in generation of the new message propagation tree using its new MDO List.

After the generation of the new tree, the new proxy sends the message to the group using the new tree. Figure 6 depicts the message propagation tree for the sample system in section 2 after recovery from the failure of the proxy.

As it can be observed in the figure, MDO 1 has been selected as the new proxy and is the root of the propagation tree.

3.4. Host Failure of the Proxy MDO after Sending a Message and before Receiving Its Acknowledgements

The last form of the host failure is detected by the child MDOs of the failed proxy. If the host of the proxy MDO fails before receiving acknowledgements from its entire child MDOs, the child MDOs can detect its failure when they try to send acknowledgement. The child MDO then selects the next available MDO in the MDO List as the new proxy for the group. The new proxy first deletes the failed proxy from their MDO List and generates new propagation tree similar to the described method in the third category of failure. Then the new proxy asks all MDOs about the highest received message sequence number. The proxy then receives all the messages which have not been received by all MDOs from one of them which has the messages and then resends the messages to the group using the new message propagation tree. The new propagation tree for this kind of failure for a system with 16 MDOs is similar to the tree in figure 6.

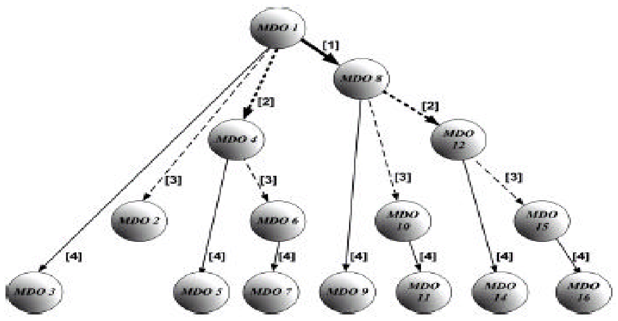

Figure 6. Message Propagation Tree after recovery form failure of Host 0 (the proxy)

In case of multiple host failure, Sama uses a mixture of the described methods for recovery. We have implemented the Sama using the described recovery from host failure feature in Java language and it automatically recovers from all of the mentioned failure categories.

4. Related Work

Many researches have been done on the group communication in distributed systems; however, almost all of them assume static group members, i.e. they remain at the same host in the distributed environment during their whole life in the system. Among these mechanisms, the reader can refer to [20].

A few mechanisms have been developed for the group communication among mobile agents. We can categorize them into two main groups.

1. Mechanisms in which an agent has a proxy such that it informs its proxy about its current location on migration. These mechanisms should know the current locations of the agents to deliver message to them. This property restricts autonomy of mobile agents.
2. Mechanisms in which there is no proxy for agents and they can migrate autonomously.

These mechanisms do not need to know the current location of the agents.

Mobile Process Groups [12] and Voyager Spaces [5] are among the first category. Mobile Process Groups are process groups that support migrating processes [12]. In this method, each process installs a view, which is a mapping between processes and their locations. To maintain consistent views, agents need to update their views in case of any change in the group such as migration of a member. Clearly, this is costly in large systems. In this approach, when an agent wants to send a message to the group members, it sequentially sends the message to all agents in its installed view. This approach provides features for recovering from host failures but does not scale well to the large group members.

Some mobile agent platforms provide group communication mechanism for agent groups [13]. Voyager uses a specialized architecture with superspaces and subspaces to deliver the messages [5]. In Voyager, a space is a logical container that contains some subspaces objects. A message is sent into a space by publishing it into one of its sub-spaces. Then, it is cloned in all neighboring subspaces and is also delivered to every object in the local subspace. Space users have to connect subspaces to form arbitrary topologies. The mechanism has negative impact of sending many unnecessary messages and consuming high bandwidth for a large number of connected subspaces. Indeed, many nodes might receive a message several times. Because members of a subspace can migrate to different locations, the number of remote messages can also increase rapidly. Voyager does not provide any mechanism for recovering from host failures.

Among the mechanisms in the second category, we can mention the mechanism proposed in [14] and group communication using IP multicast [15]. In [15], the mechanism interprets the migration of a group member as a change in multicast group. It uses Multicast Backbone (MBone) [16] as it communication infrastructure. However, MBone comprises only a small fraction of the Internet routers which considerably restricts the applicability of the method.

A multicast mechanism using reliable communication in a fault-free environment has been proposed in [14]. It attempts to deliver a message to every agent using an approach similar to the distributed snapshot [17]. However, only agents who are group members, actually, accept the message which makes it slow in large scale systems. In this approach, a message may be delivered several times to an agent. The mechanism does not provide any feature for detecting and recovering from host failures.

An event multicasting among mobile agents has been proposed in [18], which is similar to the event model of Java. The method uses 'EventTransceiverServers' to distribute messages over the network. However, the sender should broadcast the message to 'EventTransceiv-erServers' sequentially, which is time consuming in large scales.

4. Conclusions and Future Work

We have proposed Sama, a distributed and scalable application level group communication mechanism for large scale mobile agent applications which delivers messages in a considerably low time. Sama uses Message Dispatcher Objects (MDOs), which are special stationary agents, to parallelize and speed up message delivery to the group members. We then provide the methods that Sama uses to deal with host failure in the system. Sama uses an acknowledgement and timeout mechanism to recover form host failures. We categorized host failures into four different groups and showed how Sama recovers from them.

We have implemented Sama in Java and using Voyager mobility features. We tested Sama on a LAN with 16 hosts and compared it with Mobile Process Groups [11]. Our experimental results showed that Sama scales well and propagates messages in considerably lower time in comparison to Mobile Process Groups; however, we planned to make more comprehensive evaluation on the mechanism using NS simulator [21].

References

[1] A.Fuggetta, G.P.Picco and G.Vigna. "Understanding Code Mbility", IEEE Transactions on Software Engineering. Vol.24, No.5. May, 1998

[2] B. Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko and D. Rus. "Mobile Agents in Distributed Information Retrieval", In Intelligent Information Agents, pages 355-395, 1999.

[3] Bieszczad, A., White, T. and Pagurek, B., "Mobile Agents for Network Management", In IEEE Communications Surveys, September, 1998.

[4] P. Dasgupta, N.Narasimhan, L.E. Moser and P.M. Melliar-Smith, "MAgNET: Mobile Agents for Networked Electronic Trading", IEEE Transactions on Knowledge and Data Engineering, Special Issue on Web Technologies, vol. 24, no. 6, July/August 1999, pp 509-525

[5] Recursion Software, Inc. Voyager ORB Developer's Guide, 2003. www.recursionsw.com.

[6] IBM Japan Research Group Aglets Workbench, web site: http:// aglets.trl.ibm.co.jp

[7] Grasshopper, Release 2.2, Basics and Concepts (Revision 1.0), March 2001. http://www.Grasshopper.de

[8] Pawel T.Wojiehowski. "Algorithms for Location-Independent Communication between Mobile Agents". Technical Report DSC-2001/13, Département Systèmes de Communication, EPFL, March 2001.

[9] Langton C., Minar N.,and Burkhart R. ,"The Swarm Simulation System: A tool for stuying complex systems". (Draft available at: http://www.santafe.edu/projects/swarm/swarmdocs /swarmdoc.html). Santafe Institute,1995

[10] Hojjat Jafarpour and Nasser Yazdani, "Sama: A Scalable Group Communication Mechanism for Mobile Agents", In Proc. of SNPD 2003, Lubeck, Germany, Oct. 2003, pp. 506-511.

[11] 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.

[12] Flávio M. Assis Silva, Raimundo J. A. Macêdo. Reliable Communication for Mobile Agents with Mobile Groups. In the Proceedings of the Workshop on Software Egineering and Mobility (co-located with IEEE/ACM ICSE 2001). Toronto, Ontario, Canada. May 13-14, 2001.

[13] R. Broos, B. Dillenseger, P. Dini, T. Hong, A. Leichsenring, M. Leith, E. Malville, M. Nietfeld, K. Sadi and M. Zell. "Mobile Agent Platform Assessment Report", http://www.fokus.gmd.de/research/cc/ecco/climate/ ap-documents/miami-agplatf.pdf

[14] A.L. Murphy and G.P. Picco, "Reliable Communication for Highly Mobile Agents", Journal of Autonomous Agents and Multi-Agent Systems, Special issue on Mobile Agents, pp 81-100, 2002

[15] Hartroth and M. Hofmann, "Using IP Multicast to Improve Communication in Large-Scale Mobile Agent Systems", In Proceedings of 31st Annual Hawaii International Conference on System Sciences (HICSS), Volume VII, Page 64-73, Hawaii, January 6-9, 1998.

[16] K. Almeroth, "The Evolution of Multicast: From the MBone to Inter-Domain Multicast to Internet2 Deployment", IEEE Network Special Issue on Multicasting, January/February 2000.

[17] K.M. Chandy and L. Lamport. "Distributed Snapshots: Determining Global States of Distributed Sys-tems", ACM Trans. on Computer Systems, 3(1):63-75,February 1985.

[18] J. McCormick, D. Chacón, S. McGrath, and C. Stoneking, "A Distributed Event Messaging System for Mobile Agent Communication", Technical Report TR-01-02(Lockheed Martin Advanced Technology Laboratories) March 2000.

[19] G.Coulouris, J. Dollimore and T. Kindberg, Distributed Systems - Concepts and Design, 3rd edition Addison-Wesley, 2001.

[20] 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.

[21] Network Simulator. www.isi.edu/nsnam/ns/