Sama: A Scalable Group Communication Mechanism for Mobile Agents

Hojjat Jafarpour and Nasser Yazdani

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

Phone:

Fax:

Principal Author: Hojjat Jafarpour

Abstract. *Provision of fast and scalable group communication for mobile agents can considerably improve their efficiency. Unfortunately, most of the existing approaches do not scale well when the number of agents grows. In this paper, we propose Sama, a new group communication mechanism, to speed up message delivery to a group of mobile agents. The main contribution of Sama is distribution and parallelization of message propagation in an efficient way to achieve scalability and speed up message delivery to the group members. Sama uses Message Dispatcher Objects (MDOs), which are stationary agents on each host, to propagate messages in a parallel manner. The proposed mechanism is independent of agent locations and transparently delivers messages to the group using a constant number of remote messages. Experimental results show that message delivery time is significantly reduced in Sama compared to the previously proposed methods.*

Keywords: Mobile Agents, Group Communication, Large-scale Mobile Agent Systems, Mobile Agent Communication

1 Introduction

Mobile agent technology has introduced an attractive model for distributed systems [2]. Mobile agents are executing programs that can migrate, at time of their own choosing, from one machine to another in a heterogeneous network. Their ability to migrate and perform their tasks locally reduces the network load and execution time considerably [1]. Mobile agents have been used in various distributed applications such as distributed information retrieval [3], network management [4], ecommerce [5], etc. There are many mobile agent platforms which provide facilities to develop agent based applications; among them to mention Voyager [13], Aglets [14] and Grasshopper [15].

Communication of agents is a fundamental issue in many multi agent systems. Different models for agent communication have been proposed including broadcasting, forwarding and central server [16]. Group communication is an important and widely used communication model in distributed systems. In this kind of communication, a message is delivered to all group members, whereas the group membership is transparent to the sender. In the mobile agent realm, mobility of the group members introduces new challenges. Clearly, in scenarios that fully exploit mobility, where objects are rapidly moving or their migration is not as tightly controlled, most of the conventional techniques are inapplicable. Scalability of group communication mechanisms is also a critical factor in large-scale agent systems. Examples of large-scale mobile agent systems include e-business applications and Internet-wide data warehouses [17].

In this paper, we propose a scalable group communication mechanism for mobile agent systems. Our approach, called Sama, considerably speeds up message delivery to the group members by parallelizing the message dissemination task using an efficient algorithm. It distributes the load of message propagation among network nodes and uses a constant number of remote messages. Sama uses Message Dispatcher Objects (MDOs), which are objects on each host, to parallelize message dissemination process. It also delivers messages in a considerably low time in comparison to the previously proposed mechanisms for mobile agents.

The rest of paper is structured as follows. Section 2 reviews the related work. In section 3, we propose our group communication mechanism. In this section, the system model and message propagation algorithm of Sama are presented. Section 4 discusses some characteristics of the mechanism. In section 5, we compare Sama with some of the existing mechanisms and present our experimental results. Section 6 suggests some directions for future work and concludes the paper.

2 Related Work

Group communication has been one of the hot research areas in distributed systems. Many mechanisms have been proposed for distributed systems including [18], [19] and [20]; however, none of them have considered mobility of the group members.

Several mechanisms for the mobile agent group communication have also been developed. An effective group communication service for mobile agent systems can considerably improve the overall efficiency of these systems. Generally, we can divide agent group communication mechanisms into two categories:

1. Mechanisms that depend on agent locations and restrict migration of agents.
2. Mechanisms that are independent of agent locations and agents can migrate autonomously.

In the first category, an agent can be reached using a stationary proxy which knows its current location. Upon migration, agent has to inform its corresponding proxy about its new location. In the second category, on the other hand, there is no proxy for agents and they can migrate freely. This approach provides high rate of migration and autonomy for agents in the system.

Among the first category we can name Mobile Process Groups [10] and Voyager Spaces [13]. Mobile Process Groups are process groups that support migrating processes [10]. Each process installs a view, which is a mapping between all processes and their locations. This implies that each process knows all other group members and their locations. This also enforces the agents to maintain consistent views of the system and update them periodically, which is clearly costly in large scales. In this approach, message propagation is not transparent which means the sender should know all group members [9]. 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 does not scale well to large group members.

Some mobile agent platforms provide group communication mechanism for agent groups as well [12]. Voyager uses a specialized architecture with spaces and subspaces to deliver the messages [13]. In Voyager, a space is a logical container that can span multiple virtual machines across the network. Subspace class is the basic element and building block of a space. A message is sent into a space by publishing it into one of its subspaces. Then, it is cloned in all neighboring subspaces. In addition, the message is delivered to every object in the local subspace, resulting in a rapid fan out of the message to every member of the space. As the message propagates, it leaves behind a marker unique to that message which prevents the message from being repropagated into subspaces which have already been sent. 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.

Some examples of the location independent category are the mechanism proposed in [6] and group communication using IP multicast. A group communication mechanism using reliable communication in fault-free environment is proposed in [6]. The mechanism attempts to deliver a message to every agent using a method similar to the distributed snapshot [21]. However, only agents whose identifiers match the message target, 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 method also assumes nodes are connected through FIFO channels. Implementation of these channels is another challenge for the mechanism.

In [7] a group communication mechanism for mobile agents based on IP multicast has been proposed. The method captures the inherent agility of mobile agents in a scheme of dynamically adapted multicast groups. When an agent migrates to a new location, the group is changed and the new location is added to the group. If there was no agent in the previous location any more, it is discarded from the group. The method uses Multicast Backbone (MBone) [8] as an infrastructure for multicast. Unfortunately, MBone comprises only a small fraction of the Internet routers. This considerably restricts the applicability of the method.

An event propagation mechanism among mobile agents has been proposed in [11], which is similar to the event model of Java. The method uses 'EventTransceiverServers' to distribute messages over the network. However, the sender should send the message to 'EventTransceiverServers' sequentially, which is time consuming.

Our mechanism, Sama, can be classified as a location independent mechanism. We do not assume proxy for mobile agents and a sender need not to know all group members. The mechanism delivers messages to group members without knowing their locations and does not restrict their migration.

3 The Proposed Group Communication Mechanism

3.1 System Model in Sama

Sama is an application level group communication mechanism. We assume a heterogeneous network model such as the Internet where there exists at least one path between every two hosts. In order to be able to accept mobile agents there should be an agent server running on each host in our system. We also assume that the underlying mobile agent framework provides communication features among system components. All communications are done in application level and use techniques such as remote method invocation. For instance, we can use Voyager's messaging service for communication between system components [13].

For the sake of simplicity, we assume hosts do not use multithreading. Thus, each host can send one message at a time. By small changes in the algorithm which are discussed in the next subsection, Sama can exploit multithreading capability of hosts too.

For the proposed mechanism knowing the following parameters is critical:

- Maximum Message Transfer Time (MMTT)
- Maximum Agent Migration Time (MAMT)

MMTT is the maximum amount of time takes a message to be transferred between two hosts. We can calculate MMTT using the round trip application level delay between hosts in the system. MAMT is the maximum time which takes an agent migrates from one host to another. MAMT can be calculated in the same way as MMTT. Having these parameters, Sama can guarantee message delivery to all group members in a fault-free environment. On the other hand, without MMTT and MAMT some highly mobile group members may not receive the message. These members move freely and frequently from one host to another. Some scenarios for this case have been discussed in [6].

Message Dispatcher Object (MDO). MDOs are the main components in our mechanism which route and deliver messages to the group members. There is one MDO on each host in the system. They can be imagined as a part of agent servers that are running on every host. MDOs are created, set up and sent on all hosts by the system administrator before the mechanism starts its work. This can easily be done using a MDO creator program. Each MDO has the following components.

- MDO List
- Message Storage Queue
- List of the Local Group Members
- MMTT
- MAMT

Each MDO knows all MDOs and their locations on the network. MDOs store this information in a list which can be a typical data structure such as an array. Because we store minimum required information to reach MDOs, size of the list is scalable to the large number of MDOs. The position of each MDO is the same in all MDO lists in the system. Each MDO also has a message storage queue, which is used to store incoming messages. A timeout value is assigned for each incoming message which is calculated using MMTT and MAMT. MDOs use the timeout values to discard messages from their message storage queues. They also have a list of all local agents, which are members of the group. Using information stored in the list of local group members, MDOs can deliver incoming messages to their local group members.

MDOs also provide facilities for mobile agents to join, leave, register or unregister to the group. Agents use register and unregister methods of MDOs when they want to migrate to another host. Before migration, an agent unregisters itself from the list of local group members of the MDO of the source host. Then, it migrates to destination host. After migration, the agent registers itself to the MDO of the destination host and receives all messages which it could not receive during its migration.

Mobile agents should have methods to join, leave, register or un-register to a group. Group members also know their local MDO and send messages to the group by passing them to their local MDO.

Figure 1 depicts a sample system with 16 hosts and some mobile agents.

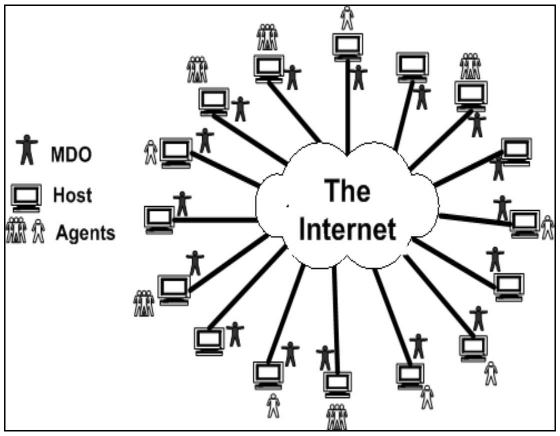


Fig. 1. A sample system model with 16 hosts connected through the Internet.

3.2 Message Propagation Mechanism

Sama uses an efficient algorithm to speed up message delivery. Each MDO executes the algorithm upon receiving a message. The main idea of the algorithm can be described as follows. Suppose there is a group of objects and one of them wants to send a message to the group. It first sends the message to one of the objects. Now, two objects know the message and the second object can contribute in message delivery process hereafter. Then, the two objects send the message to two other objects. At this time the number of objects that know the message is four. Accordingly in the next step eight objects will know the message and so on. Our proposed algorithm indicates the order of message delivery in the described scenario. As it can be seen, the number of objects which have received the message is doubled in each step and this causes exponentially propagation of the message among group members.

Our mechanism has two phases. In the first phase, when a message is sent to the group, it is propagated among MDOs using the previously described idea. In the second phase, each MDO delivers the message to its local group members. The first phase is done in a logarithmic order and a parallel manner. The algorithm is shown in Listing 1.

Upon receiving a message from the sender by the first MDO, it associates a sequence number to the message using a local counter. Using this sequence number, the group members can avoid receiving double messages. Then, it finds the beginning and ending indices of its customized MDO list. The customized MDO list is a sub-list of the main MDO list, which is customized for execution of the algorithm and is generated in each iteration of execution of the algorithm. At this time, the customized MDO list for the first receiver is the original list of MDOs. Step 2 in the algorithm describes the calculations required to find the customized list in the first receiver. After finding the customized list, the MDO finds the median component of the customized list. The fourth step shows how the index of the median in the customized list can be found. The algorithm assumes the customized MDO list as a circular list which the boundary indices indicate the front and tail of the list. The median is the middle component in this list.

```
Each MDO does the following phases after receiving a message.
Suppose the number of MDOs is n and the boundaries of the Customized MDO list are ( a , b )
Phase 1:
    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
           $a = (p+1) \bmod n$ and  $b = (p-1) \bmod n$.
        b. Else go to Phase 2.

    3. If $b-a \bmod n < 2$ send the message to the MDOs which are at indices a and b and go to Phase 2.

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

    5. Calculate the boundaries of the new customized list, which is the first half of the current list as follow:
    $a = a$ ,  $b = (m-1) \bmod n$

    6. Send the message and the following boundaries to the MDO, which is at index m in the old MDO list.
    $a = (m+1) \bmod n$ ,  $b = b$

    7. Go to step 1.

Phase 2:
    1. Pick up agents from the corresponding agents list and send the message to them.
```

Listing. 1. The Message Propagating Algorithm

Then, using the index of the median MDO, the MDO divides its customized list into two sections and finds the boundary indices of them. All calculations are done in modulo ***n*** where ***n*** is the number of MDOs in the system. Then, it sends the message and the boundary indices of the second half of the divided list to the median. The sender MDO is considered as the parent of the receiver MDO. Now the median is responsible for delivering the message to the MDOs listed in the second half of the list and starts sending the message to them using the same algorithm. The main MDO is responsible for the first half of the list which forms its new customized list for the next iteration of the algorithm execution. The MDO continues this task until the size of its customized list becomes one or two. At this time, it just sends the message to the MDO(s) in its customized list and does not repeat previous steps. Then, the MDO executes the second phase of the algorithm and delivers the message to its corresponding local agents sequentially. MDOs which are not parent of any MDO after delivering the message to their local group members acknowledge the message delivery to their parents. Parent MDOs also acknowledge to their own parents after delivering the message to their local group members and receiving acknowledge from their child MDOs.

By making some changes in the algorithm, Sama can exploit multithreading capability of hosts as well. Suppose each host can send three messages simultaneously. Now, instead of dividing customized list into two sections, the algorithm divides the customized list into four sections and sends the message to three MDOs simultaneously.

3.3 Example

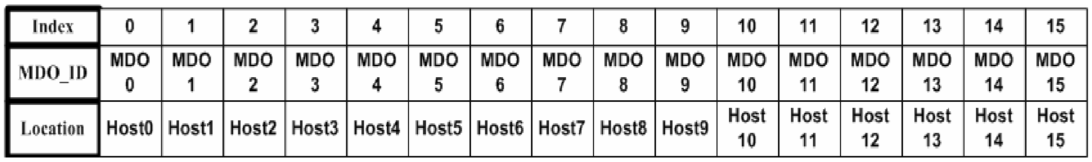
We assume a system with 16 hosts, and consequently, with 16 MDOs. We also assume there is only one group member on each host. Each MDO has the list of all MDOs in the system including itself. Suppose that the MDOs can be identified by the index of their position in the MDO list. Figure 2 illustrates the list. For the sake of simplicity, we assume the MDO in position 0 receives the message first. We call it *MDO0*. Accordingly, ***n*** is 16 in the first iteration. Because there is no boundary, *MDO0* calculates the boundaries of its customized MDO list as:

$$a = (0 + 1) \bmod 16 = 1$$
$$b = (0 - 1) \bmod 16 = -1 \bmod 16 = 15 \qquad (1)$$

Then, *MDO0* finds the median MDO in its customized list, which contains the MDOs that are in indices ***a*** to ***b***. Here, the median MDO is in the index ***m***, which is calculated by:

$$m = (a + ((b-a) \bmod x)/2) \bmod x = (1+((15-1) \bmod 16/2) \bmod 16 = 8 \qquad (2)$$

Formula 2, which is proposed in step 4 of the algorithm, supposes the MDO list as a circular list and finds the median of the list using calculations in modulo ***n***. Now, *MDO0* calculates the new boundaries for its customized MDO list using the formulas in step 5. It then sends the message and the boundaries of the second half of the customized MDO list to the median and makes it responsible to deliver the message to them. *MDO0* then repeats the algorithm with the new customized MDO list. After receiving the message, each MDO executes the algorithm. When the size of the customized list is one or two, the MDO



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
Location	Host0	Host1	Host2	Host3	Host4	Host5	Host6	Host7	Host8	Host9	Host 10	Host 11	Host 12	Host 13	Host 14	Host 15

Fig. 2. The list of MDOs which is stored in all MDO.

just sends the message to the MDO(s) and goes to the second phase implying delivering the message to the corresponding agents sequentially.

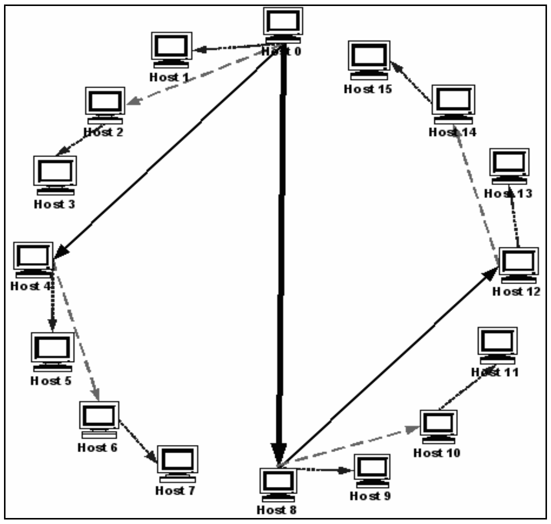


Fig. 3. Message propagation process among MDOs

Figures 3 and 4 illustrate the message propagation process among the MDOs in our example system. In figure 3, the overall process of the message propagation is depicted. Figure 4, arranges the hosts according to the message reception order. Each level of the figure corresponds to an iteration and shows the MDOs that receive the message in that iteration. As it can be seen there, the total operation is done in four iterations and all of the MDOs have an instance of the message by the fourth iteration. The thick arrow shows the first iteration in which *MDO0* sends the message to *MDO8*. As it can be seen, after this iteration, *MDO8* is responsible for message delivery to half of the MDOs. Then, *MDO0* and *MDO8* repeat the first phase of the algorithm with the new customized MDO list boundaries, (1, 7) and (9, 15) respectively. The thin arrows show the second iteration and the dashed arrows are for the third iteration. By the forth iteration which is depicted by the dotted arrows, the message has been delivered to all MDOs in the system. In this iteration, each MDO has only one MDO in its list to send the message and there is no need to calculate the boundaries.

After the fourth iteration, the first phase of the algorithm terminates and all MDOs execute the second phase of the algorithm. In this phase, each MDO delivers the message to its local agents sequentially.

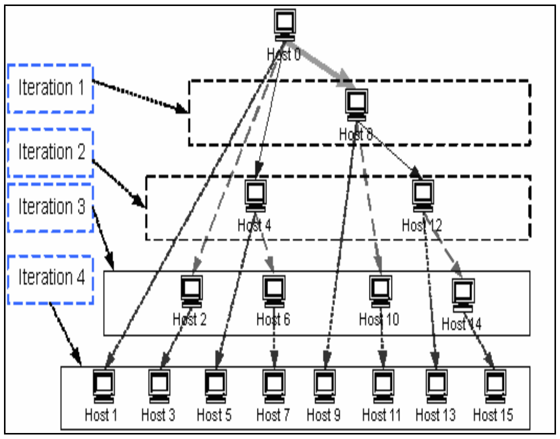


Fig. 4. Reception levels

4 The Main Characteristics of Sama

In this section, we review the main characteristics of Sama group communication mechanism.

Guaranteed Message Delivery. Sama guarantees message delivery to the all group members in a fault-free environment. Some scenarios are presented in [6] that show in some situations in a fault-free environment, some highly mobile agents might not receive the message. Sama stores messages in MDOs for a limited period of time to ensure that all agents receive the message from at least one MDO. MDOs store messages in their Message Storage Queues. If the number of MDOs in the system presented by N_{MDOs}, the amount of time each message should be stored in each MDO is calculated by:

$$MessageStorageTime = MMTT * \log_2(N_{MDOs}) + MAMT \qquad (3)$$

To understand formula 3, let assume that, in the previous example, the message transfer time for each of the following channels is MMTT and other links in the system have a very low message transfer time that can be ignored.

(0, 8), (8, 12), (12, 14) and (14, 15)

Consequently, *MDO0* is the first receiver of the message and *MDO15* is the last receiver among MDOs and the message reaches from 0 to 15 after *4*MMTT*. If an agent migrates from host 15 to host 0 just before receiving the message by *MDO15* it should receive the message from the MDO in host 0. *MDO0* should hold a copy of the message to ensure that it can deliver it to the incoming agents, which have not received the message yet. Obviously, if *MDO0* stores the message for at least *(4*MMTT+MAMT)*, it can deliver the message to the new arriving agent. After this amount of time, there is no need to store the message and MDO can discard the message from its queue. Figure 5 shows the scenario and agent migration path.

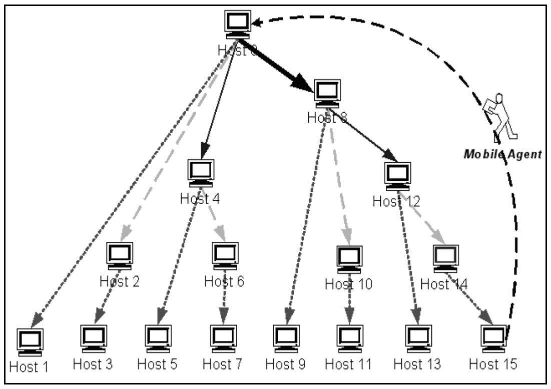


Fig. 5. An agent migrates from host 15 to 0 before receiving message from *MDO15*.

Constant Number of Remote Messages. Sending remote messages takes considerably more time than local messages, especially on the Internet. As it can be inferred from the algorithm, each MDO receives the message once and, then, delivers it locally to its corresponding agents. Consequently, the number of remote messages is equal to the number of MDOs in the system. For instance, the number of remote messages for the previous example is 16. This number can grow rapidly for previously proposed mechanisms.

Message Delivery Independent of Agent Locations. An important characteristic of Sama is that it delivers messages to the group members independent of their locations. This approach does not restrict agents' migration and their independence [6].

Message Delivery Time. Our approach reduces message delivery time in large scale mobile agent systems. As mentioned before, the message delivery operation among MDOs is done in a logarithmic order, which considerably improves message propagation speed. To calculate the maximum amount of time taken to disseminate a message among all MDOs, we can use formula (4).

$$\textit{Maximum Time for MDOs} = MMTT * \log_2 (N_{MDOs}) \quad \textbf{(4)}$$

After this amount of time all MDOs have an instance of the message. If we assume that the Local Message Delivery Time (LMDT) shows the amount of time to deliver a local message and N_{Agents} shows the number of all agents in the system, we can calculate maximum message delivery time in the worst case scenario, when all agents are located on the host with the slowest path from the source of the message, using formula (5).

$$\textit{MaximumDeliveryTime} = MMTT * \log_2 (N_{MDOs}) + LMDT * N_{Agents} + MAMT \quad \textbf{(5)}$$

After this amount of time all MDOs have delivered the message to their local agents.

Transparent Message Delivery. Transparent message delivery to a group is another property of our mechanism. By transparent message delivery we mean 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 agents [9]. This is a very important characteristic that makes implementation of the sender and the agents easy.

Open and Close Groups. Groups can be open or close. In an open group, agents, which are not group members, can send message to the group members. In contrast, in a close group just group members can send messages to the group [9]. To support an open group, we can choose a MDO as group proxy and make it accessible from the outside. Then, messages can be sent to the proxy to be delivered to the group members.

5 Comparison with the Other Approaches

In this section, we briefly compare Sama with Voyager^{TM} [13] and Mobile Process Groups [10].

Among the related works, Voyager's spaces approach is more similar to ours. In comparison with Voyager's spaces, our mechanism is simpler and faster and utilizes the resources more efficiently in systems with a large number of mobile agents. Unlike Voyager, Sama does not restrict agent migration. In Voyager, users should define communication channels between subspaces [13]. Each subspace also sends the message to all neighbors except to the subspace the message is received from. Consequently, in fully connected subspaces, each subspace receives a message several times which wastes the bandwidth and increases the load of the system. On the other hand, if the subspaces are connected via a few channels the message delivery time will increase. Finding the proper connections among the subspaces is another challenge. In our approach, the proposed algorithm finds the proper communication channels among MDOs itself. Since each MDO receives a message just once, the number of extra messages is low and the network load does not increase compared to the Voyager's approach. Furthermore, the number of remote messages is constant in our approach, which can be high in Voyager's approach.

In Mobile Process Groups each agent should install views of the system, which is difficult to maintain. The message propagation time, compared to our algorithm, is considerably high since the sender sends the message to all group members directly. This results in a high load on the sender in their method, while it is distributed among all hosts in our approach. The number of remote messages can also rise dramatically in large scale systems. In contrast, Sama is more scalable and faster when the number of group members grows.

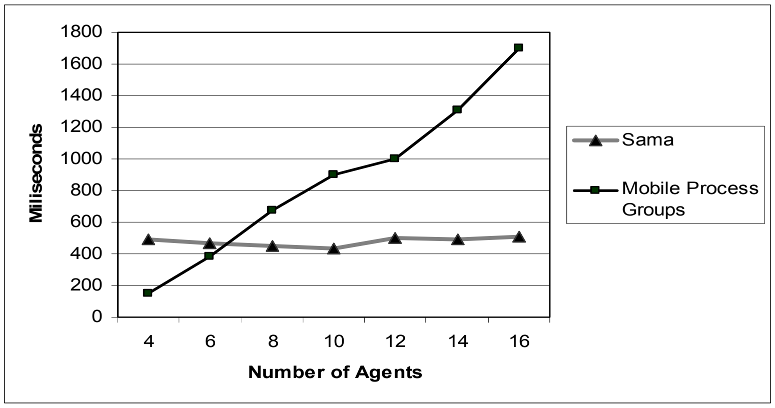


Fig. 6. The Message delivery time in Sama and Mobile Process Groups based on the number of agents in a system with 16 hosts.

We have implemented our mechanism using Voyager's messaging features which provide communication using techniques similar to Remote Method Invocation. We have compared Sama to the Mobile Process Groups approach with respect to the different number of agents. The test configuration was made of 16 hosts connected via a 100 Mbps Ethernet network and with a 20 KB string message. Each mechanism was executed 10 times and the average message delivery times were measured. The calculated time is the time between sending the message and receiving acknowledges from all group members. As it can be seen in figure 6, our proposed mechanism is very scalable in comparison with Mobile Process Groups since it shows considerably less delay when the number of group members grows. Sama tries to parallelize message propagation process and use a constant number of remote messages. Consequently, the difference between Sama and Mobile Process Groups mechanism is more considerable in low speed networks and hosts.

Both of the discussed approaches, Voyager's spaces and Mobile Process Groups, restrict agent migration and autonomous behavior by forcing them to inform their proxies in migration time. However, this may not be acceptable in autonomous agent systems. There is no such restriction in our proposed solution.

6 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 objects on every host, to parallelize and speed up message delivery to the group members. It first propagates messages among MDOs using an efficient algorithm and, then, each MDO delivers the messages to its local agents. Our approach uses a constant number of remote messages and transparently delivers messages to group members. Sama does not restrict agent migrations and they can autonomously move among hosts. It does not assume any special network model and can be applied on all networks specially the Internet. Experimental results show that Sama scales well when the number of group members grows.

There are unsolved issues regarding to a host failure. We intend to add fault tolerance features to the mechanism. We also try on applying our method to other distributed object systems such as CORBA in near future.

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] S. Parsa and H. Jafarpour. "Developing and Implementing Applications for the Internet", In Proceedings of the International Electronic and Internet Cities Conference. Kish Island, Iran. May 2001

[3] 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

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

[5] 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 Engineer-ing, Special Issue on Web Technologies, vol. 24, no. 6, July/August 1999, pp 509-525

[6] 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.

[7] 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.

[8] Bruce S. Davie and Larry L. Peterson, Computer Networks: A Systems Approach 2nd edition Morgan Kaufmann •1999

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

[10] 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.

[11] 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.

[12] 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

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

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

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

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

[17] Wijngaards, N.J.E. , Overeinder, B.J. , Steen and M. van , Brazier, "F.M.T. Supporting Internet-Scale Multi-Agent Systems", In Data and Knowledge Engineering, Vol. 41, Number 2-3, pp. 229-245, June 2002

[18] 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, Vol. 39, No 4, April 1996.

[19] van Renesse,R., Birman, K. and Maffeis,S. "Hourus: a Flexible Group Communication System", Communications of the ACM, Vol. 39, No 4, April 1996.

[20] Dolev,D. and Malki,D. "The Transis approach to high availability cluster communication", Communications of the ACM, Vol. 39, No. 4, April 1996.

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