International Journal of Computer Applications (0975 – 888)
Volume 48– No.6, June 2012

$62.5 US The Modeling and Implementation of Reliable Mobile Agent Systems using Group Communication Services

Alaa Eldeen Sayed Ahmed
Computer Engineering Department
Benha University
Shoubra-Cairo, Egypt

Rania Ramadan Abd El-dayem
Computer Engineering Department
Benha University
Shoubra-Cairo, Egypt

ABSTRACT
A mobile agent is a computer program that runs autonomously on behalf of a user and travels through a certain itinerary in a network of computers. When compared with normal client/server architecture, mobile agent paradigm adds up additional reliability problems since agents programs could be totally or partially lost due to failures that come from bad communication or computer agent's crash with the recent increase of considering mobile agent in different E-World applications, reliability is considered as a crucial issue to be faced. Most of existing mobile agent systems considers check pointing or replication as a mechanism in achieving reliable and fault tolerant execution. In this paper we present new model which employs the benefits gained from combing both mechanisms to achieve reliable mobile agent execution. Our model uses group communication services to avail different essential issues such as agent's synchronization to facilitate the implementation the protocol. The proposed approach is dynamic in the sense that it allows a flexible membership mechanism to join or leave a mobile agent groups used in achieving the reliable execution.

General Terms
Mobile agent, Distributed computing, Reliability et al.

Keywords
Mobile agent, fault tolerance, reliability, replication, check-pointing, group communication.

1. INTRODUCTION
In this paper, we introduce The Modeling and Implementation of Reliable Mobile Agent systems using group communication services. Mobile agent is a computer program that travels within a heterogeneous network of computer systems. Mobile agent systems offer advantages such as better performance, lower usage of network bandwidth and asynchronous processing [1]. They can suspend their execution on an arbitrary point and transport themselves to another computer system. During this migration the agent is transmitted completely, as a set of code, data, and execution state. At the destination computer system, an agent's execution is resumed at exactly the agent point where it was suspended before.

During the agent life cycle, there is a lot of unexpected errors may occurs. These may include failure of the computer running the agent or failure of networks nodes so tar the agent is lost. The longer the mobile agent' itinerary means higher possibility of getting failures. Thus it is important to make a mobile agent reliable. Reliable means the ability to overcome the presence of failures. This requires a mechanism for detecting the failures then recovery. As far as we know, most of prior works have considered either check-pointing or replication mechanism separately to achieve mobile agent fault tolerance. In this paper we introduce a new approach that combines the two mechanisms in an integrated fashion to achieve the best of both.

A previous version of this paper introduced the basic concept of proposed system and related membership and election protocols [2]. Here, we extend our work by further discussing and refining the concepts regarding the proposed membership and election protocol and presenting failure detection and performance data from a number of experiments; where we measured the agent round trip time, also the experiments insure the agent reliability even in case of crashes.

The paper is organized as follows. In section 2, related work is described. In section 3 we present the definition of group communication. The proposed system modeling is given in section 4. The new protocol design assumption is presented in section 5 then followed by Model Implementation and results in section 6. Finally, section 7 introduces the conclusion.

2. RELATED WORK
Reliable Fault tolerance services are implemented with several techniques and methods. One of these models is called Primary-Backup protocol. It is based on the use of multiple servers where one of the servers is designated as the primary and the others are designated as backups. [3]In this protocol the client sends its request to the primary which executes the request and sends back the result to the clients. The primary Multicasts results to all backups in order to maintain consistency. This procedure occurs with when each request is processed. In case of primary failure, one backup take over as the primary and inform the clients so that all future requests should be sent to new primary. This model is server driven model in which clients has no role in identifying the new or the faulty primary [4]. This protocol is extended to become a client active by having the clients to maintain an ordered list of computers and uses it to detect failures then elect a new leader or primary computer.

Reliable Fault-tolerant mobile agent execution must satisfy two basic properties [5]: The non-blocking property which ensures that the agent execution can make progress at any time and exactly-once property which prohibits multiple executions of the agent. Clearly, mobile agent applications have need of an agent to be executed exactly once. For example assume we have a mobile agent for buying a book with the lowest price. The mobile agent should have the exactly once property valid to be able to just buy only the book with the lowest price. On the other hand, the non-blocking is another important property that need to be valid for the reliable mobile agent execution. For example, when a mobile agent is launched, by its owner, it is expected to get result.

Agent execution proceeds in steps, where a new step is initiated whenever an agent migrates to the next node in its itinerary. A step of an agent is defined to be the set of operations performed by the agent while it visits this node. In the execution model, resources are encapsulated in resource managers. So each step may change the agent's state as well as the state of the local resources.

Now we give the definition of exactly-once. Assume the agent is performing a task T through different stages. Let P(I) be the number of nodes in the agent's itinerary I=[N_1, N_2, ..., N_P(I)] and S_i be the stage represented the node N_i (1 <= i <= P(I)). Then the execution of an agent is defined to be exactly-once if the agent executes stage Si before stage S_{i+1}, 1<= i <P(I), and each stage S_i, 1<= i <= P(I), is executed exactly once, independent of communication and node failures.

The non-blocking property ensures that the failure of an infrastructure component (e.g., a machine, place, agent, or communication link) does not prevent progress in the agent execution. A blocking execution is undesirable because it can lead the agent owner to potentially wait a long time for the return of the agent.

To achieve fault tolerance mobile agent execution most of the existing protocols use check-pointing, replication or mobile group's mechanism.

Check-pointing is a technique for achieving fault tolerance. It consists of periodically saving the state of the computation on stable storage; in case of a crash, the computation is restarted from the most recently saved state. The technique has been developed for long running computations, e.g., simulations that last for days or weeks, and run on multiple machines. These computations are modeled as a set of processes communicating by exchanging messages. There are two famous protocols to implement check-pointing technique to fault tolerant mobile agent in [6] [7]. These protocols assume that no hardware failure occurs to maintain the log entries because it can't be recorded in permanent storage. Here check-pointing can prevent the loss of the agent and ensures the exactly once execution property but it survived from blocking. Unfortunately, unreliable failure detection may lead to a violation of the exactly-once execution property.

Another fault tolerance mechanism is replication. Replication prevents the loss of the agent and avoids the blocking problem by adding redundancy masks failures and enables the agent to continue the execution despite failures. Without replication the mobile agent migration leads to blocking, as a failure may cause the loss of the agent. To prevent blocking, the agent at any stage (S_i) is sent to a set of places (M_{i+1}) at (S_{i+1}), instead of only one place. In other words, the place p^j_i hosts the agent replica a^j_i of agent a_i. There are two approaches to implement replication temporal-replication-based (TRB) and spatial-replication-based (SRB) approaches, as in [5] [8]. In general, the disadvantages of replication is not guaranteeing the exactly-once execution property while the advantages is preventing the loss and blocking of the agent.

The last mechanism is mobile groups. Mobile groups were presented as a mechanism for mobile agent reliable coordination using group communication system. Group communication systems enable mobile agents that share a collective interest to identify themselves as a single logical communication entity and are responsible for constructing a group of coordinated replicated mobile agents. A mobile group is an extension of the traditional concept of a process group that can directly support migrating processes as members of the group. With mobile groups, a migrating process has the ability to change its location in the distributed environment while belonging to a group. Mobile groups also provide message delivery guarantees and a sort of virtual synchrony. However, mobile groups provide these guarantees despite the mobility of their members [9] and [10]. There are two famous protocols to implement mobile group protocols to fault tolerant mobile agent. These protocols concerned mainly with group communication services implementation, but here the mobile agent reliability is the most important issue. These protocols don't test with any common mobile agent platform such as Voyager, Jade or Aglets but it tested with its own implementation of agent system as java classes, and these protocols can't afford too many continuous failures. However the approach in [9] is mostly conceptual and does not support atomic and totally ordered message delivery. Also, they require that each mobile agent installs a group view and updates the group view whenever any mobile agent migrates. This will result in high migration costs and is not practical. Also the protocol in [10]suffers from total failure and handles only coordinator failure not any other group member.

3. GROUP COMMUNICATION

Group communication middleware is a layer between the communication layer and the layer that implements replications. A group consists of a set of processes with an identifier. Messages can be sent to all group members by multicast the message referring to that identifier. Group communication layer consists of a membership service and a communication service. The communication service is tightly integrated with the membership service so as to provide properties that are particularly useful for reasoning of message deliveries in the context of crashes, recoveries, wrong failure beliefs, network partitions and mergers[9].Membership service supports the group membership issues. A process may join the group by invoking Join() operation, a process may leave the group by invoking Leave() operation or by crash. The membership service gives a view (a set of the currently group members) of the group to each group member and tracks the group membership and reports membership changes to members (send a view change message describing the new view to group members).The view change message generated automatically as a result of join or leave or crash.

When a group g is created, every group member a_i installs the view v_i^1 [9].After the initial view is installed, any modification on the group members (join or leave) will result in new views being installed, forming the sequence v_i^1, v_i^2, . . . , v_i^n where n represents a specific group view. This group needs to implement group communication properties:

i. View Safety Properties:

Validity1: if an agent $a_i \in$ g installs a view v_i^n (g), then $a_i \in v_i^n$. This states that only the members of a group view install the corresponding view.

Validity2: if an agent $a_j \in v_i^n$ and $a_j \notin v_i^{n-1}$ then a_j asked to join the group g.

Validity3: if an agent $a_j \notin v_i^n$ and $a_j \in v_i^{n-1}$, then a_j asked to leave g or it has been suspected of crashing by some group member.

ii. View Liveness Properties

Termination1: If an agent a $\in v_i^n$ asks to leave g or crashes and there exist at least two correct agents in v_i^n , then there will be a view v_j^{n+1} installed by an agent a_j, such that $a \notin v_j^{n+1}$ and $a \in v_j^s$ for all s such that $k \leq s < r$.

Termination2: If an agent a tries to join g, then there will be a view v_j^r installed by an agent a_j, such that $a \in v_j^r$ and $a \notin v_j^s$ for all s such $s < r$.

The advantages of group communication layer in building replication [11][12]:

i. A process does not (attempt to) track whether other processes are alive or not, as this job is done transparently by the underlying membership service.
ii. Each process is confident that its own perception of the membership is identical to that of all other (non-crashed) view members.
iii. Taking the leader over does not require a dedicated protocol for figuring out which updates have been carried out so far: it simply requires electing a new leader. Even if the leader crashed while sending an update, it is guaranteed that either all surviving backups received the update, or none of them did.
iv. When the leader is taken over, the new leader does not worry about whether the backups will receive new updates after or before observing the left of the previous leader. It is guaranteed that messages sent after a certain view change will be delivered everywhere after that view change (if delivered at all).

4. PROPOSED SYSTEM MODELING

In section 2 we introduced the desired properties for fault tolerant mobile agent execution, exactly-once and non-blocking, and also the mechanisms used to fault tolerant the mobile agent, check-pointing and replication, which achieves one property and suffers from the other or suffers from total failure as mobile groups mechanism. Also the properties of group communication middleware and its usefulness in building replication are mentioned in section 3. Using the group communication middleware to fault tolerant mobile agent execution in our proposed system aimed to achieve the two desired properties, exactly-once and non-blocking. Group communication supports replication (non-blocking) when crash occurs, also it always executes the group leader agent only (exactly-once). Group leader is the oldest member in the group (current group view).

4.1 System model and architecture:

The architecture of the proposed system is shown in **Error! Reference source not found.**. The system is built upon the mobile agent computing network and provides group communication services and reliability for mobile agent applications. There are three components in the system: (1) membership protocol, (2) election mechanism and (3) failure detection mechanism.

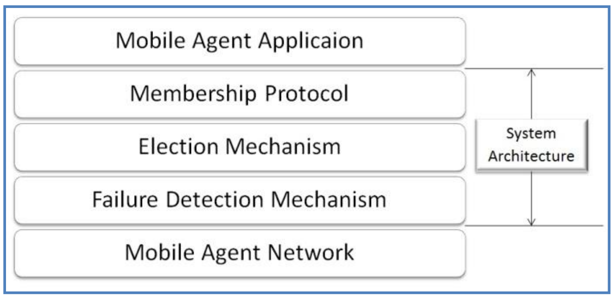

Fig 1: System Architecture

4.1.1 Membership Protocol:

A mobile agent may become a member of or depart a group by issuing a JOIN/LEAVE request. It is assumed that mobile agents leave the group due to crashes. At the point of interaction between mobile agents and the membership protocol, there are two different operations as follows:

• Join operation: A mobile agent wishing to become a group member constructs a JOIN message. Then mobile agent request an update message from group to update its state and data to the state and data of group
• Abnormal leave operation: When the failure detection mechanism finds that a mobile agent is suspected to crash, this agent is deleted from group message list and the group will not send any messages to it.

4.1.2 Election Mechanism:

There are many algorithms to elect a leader but here the simplest leader election algorithm are used, this algorithm based on selecting the oldest active member in group to be the leader. Election Algorithm properties:

• Safety: the elected agent=a_i, where a_i is chosen as the non-crashed agent at the end of the run with the smallest identifier (oldest member in the group (current view)).
• Liveness: if the agent a_i is the oldest member in the current view and elected $\neq a_i$, then a_i crash.

The election algorithm used here is bully algorithm. The bully algorithm is a method in distributed computing for dynamically selecting a coordinator by process ID number. When a process P determines that the current coordinator is down because of message timeouts or failure of the coordinator to initiate a handshake, it performs the following actions:

• P broadcasts an election message, inquiry, to all other processes with its time.
• If P hears from no process that it is older than P, it wins the election and broadcasts victory.
• If P hears from a process that it is older than P, P waits a certain amount of time for that process to broadcast itself as the leader. If it does not receive this message in time, it re-broadcasts the election message.
• If P gets an election message, inquiry, from another process with a time smaller than its time, it sends an "I am alive" message back and starts new elections.
• If P receives a victory message from a process with a time smaller than its time, it immediately initiates a new election.

This is how the algorithm gets its name; a process with older time will bully a newer time process out of the coordinator position as soon as it comes online. The election mechanism takes 10 seconds approximately until leader elected depends on number of group members.

4.1.3 Failure Detection Mechanism:

The failure detector is equipped to all group members. Failure detection is based on check-alive messages that keep executing continuously. Agents call each other with check-alive message, when the agent doesn't receive a replay from an agent this agent is marked as crashed. Failure mechanism distinguishes between leader crash and any other group member crash. If the crashed agent is the leader, the failure detection mechanism calls election mechanism to elect new group leader. If any other group member crashes nothing is done until the number of agents per group reaches the critical number =2, then the failure detection mechanism calls the membership protocol to create and join new agents; so this algorithm can prevents total failure case.

5. THE PROPOSED SYSTEM DESIGN

Before describe the algorithm design, some terms used with the algorithm must be defined:

•Create: Create an agent
•Join: Join the agent to the group.
•Multicast: Multicast the agent data to all group members to update their data version.
•Execute: Execute the agent application.
•Move(l): Move the agent to location l in the itinerary.

When the agent moves from stage$_i$ to stage$_{i+1}$, the next place in the itinerary, the leader does the following tasks:

•Send the agent to the next stage in the itinerary.
•Create new group in the next stage in the itinerary.
•Delete the previous stage group.

In this section the two scenarios for failure models are described which using in implementing the proposed system. The first scenario is dealing with the case of no failure occurs and the other scenario considers the case with a failure. The second scenario includes three different failure models. These models are:

*i.*The leader of group crashes.
*ii.*Non- leader member crashes.
*iii.*A hybrid model, where any member crash leader or not.

5.1 No Failure Scenario:

The steps of the algorithm in the failure free case works as following:

1.Leader agent of stage si travels to stage si+1 as in **Error! Reference source not found.**.
2.The leader agent at stage s$_i$, the first and only agent at s$_{i+1}$, creates other agents at stage s$_{i+}$1according to maximum number of group members as in **Error! Reference source not found.**.

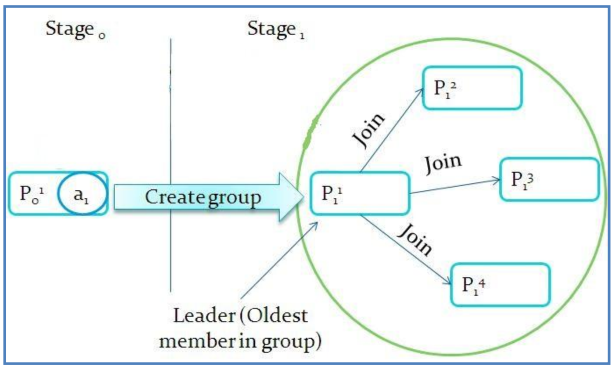

Fig 2: Group Construction

3.After creation of agents, the new agents update their data from the leader of group at s$_i$ as in **Error! Reference source not found.**.
4.At stage s$_{i+}$1agents construct and join group, each member in this group must has the same state, and group members must be greater than minimum number of group members (K), the minimum number = 2.
5.Delete the group at s$_i$.
6.Elect the leader of group which is the oldest one in the group.
7.The leader agent performs the application.
8.The leader agent sends (Multicast) update message with the new agent data to all group members, as in **Error! Reference source not found.**.

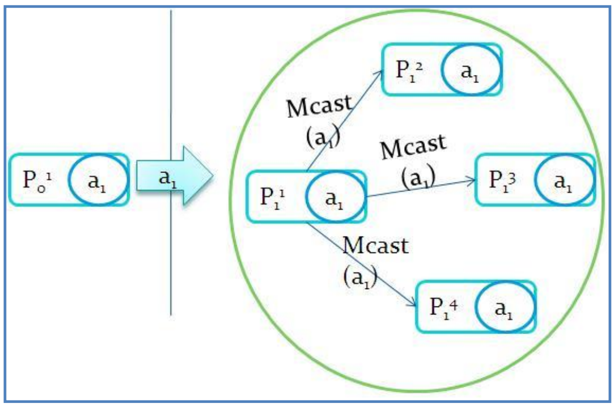

Fig 3: Multicast the agent after group construction

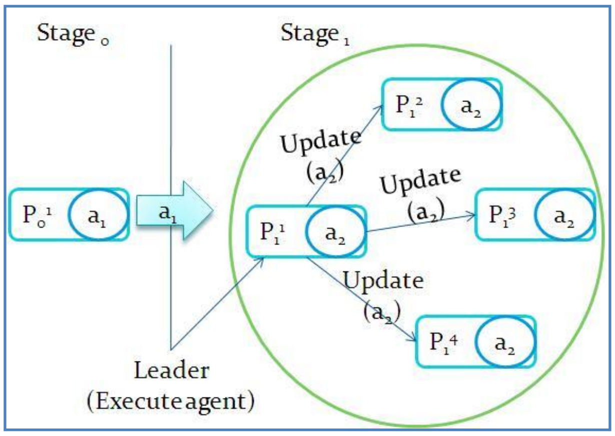

Fig 4: Multicast the agent after agent execution

9.After the leader of the group completes its job at this stage, then
a)If this is not the last stage the leader agent requests to move to the next stage in itinerary, as in **Error! Reference source not found.**.
b)If this is the last step the leader agent returns the application result to the application caller.

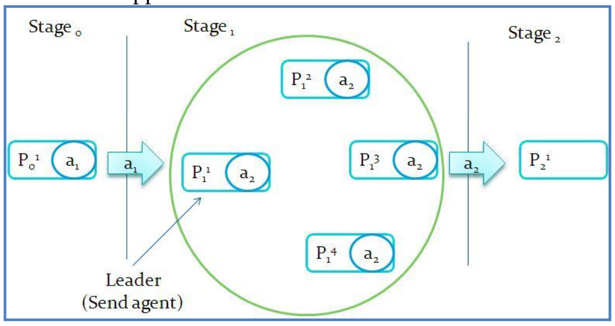

Fig 5: Agent travels to next stage in itenrary

5.2 Failure occurrence Scenarios:

Any group member can leave the group by executing leave operation or crash. The worst case is happening when all group member's crash. If this happens we get the total failure case that causes the group to disappear. Reaching this case will not enable the algorithm to continue and we must restart the system. To overcome this case, [13]discusses a method to store the state of current group member in a stable storage. But in our proposed system we limit the minimum numbers of group member to two members, critical number of group members, to prevent total failure. Failure models covered in the proposed system:

5.2.1 When the group leader crashes:

When the leader crashes, the failure detection mechanism recognizes the change. So failure detection mechanism calls the Leader election mechanism to elect a new leader of group and inform the group members with the new leader. Then the new leader continue the execution, but if the number of active group members is equivalent to the critical number of group members then the leader must create new members to avoid total failure case. Leader crashes in one of the following cases:

a)Before agent execution, so the new leader will execute the agent, as in **Error! Reference source not found.**.

b)During agent execution, so the new leader will execute the agent from scratch, discarding the execution done by crashed leader, as in **Error! Reference source not found.**.

c)After execution but before sending the new agent data to other members, so the new leader will execute the agent from scratch, discarding the execution done by crashed leader, as in **Error! Reference source not found.**.

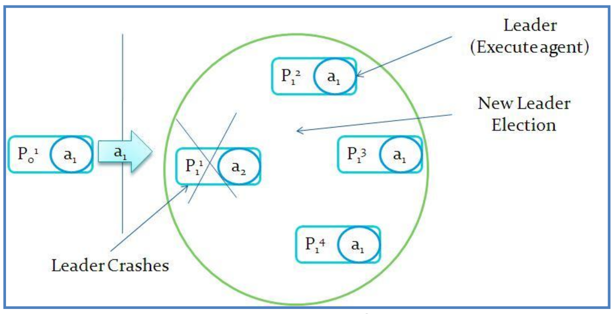

Fig 6: Leader crashes before update message

d)After update the agent to all members but before sending to next stage, so the new leader will transmit the agent, as in **Error! Reference source not found.**.

e)During transmission, after updating the group members, so the new leader will retransmit the agent, as in **Error! Reference source not found.**.

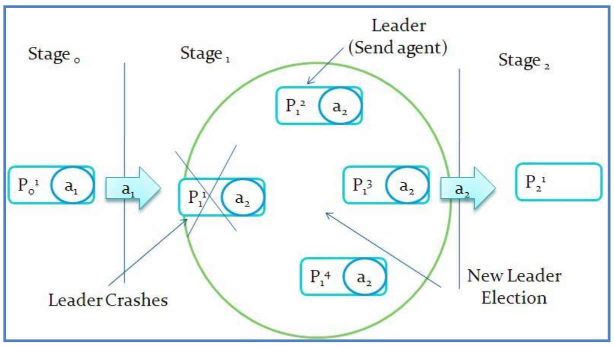

Fig 7: Leader crashes after update message

5.2.2 When a Non-leader group member crashes:

When any member of group crashes except leader the failure detection mechanism knows that the crashed agent isn't the leader and informs the other members in the group. Since the leader is not allowed to crash, The leader continue the execution regardless the changes, as in **Error! Reference source not found.** and **Error! Reference source not found.**, unless the number of active group members is equivalent to critical number of group members then the leader calls the membership protocol to create new agents reaching the maximum number allowed of the group and this agents join the group to avoid the total failure case and to achieve fault tolerance at any number of crashes then the leader continue performing the application as in **Error! Reference source not found.**.

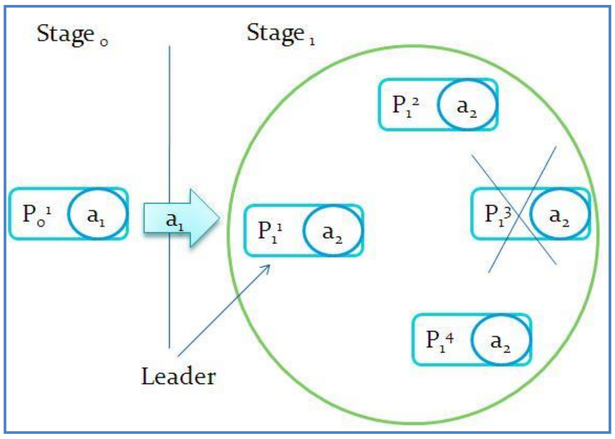

Fig 8: Non-leader agent crash and K>2

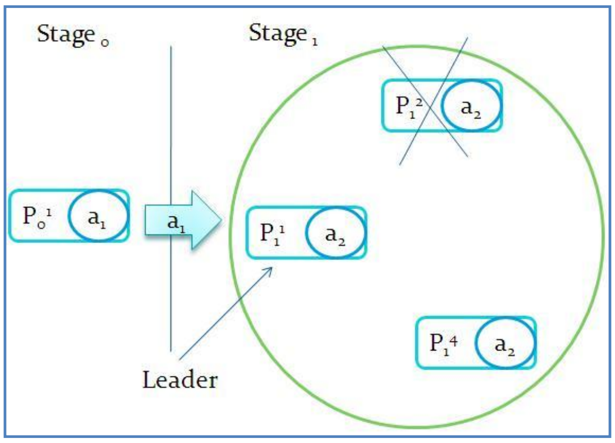

Fig: 9 Non-leader agent crash and K=2

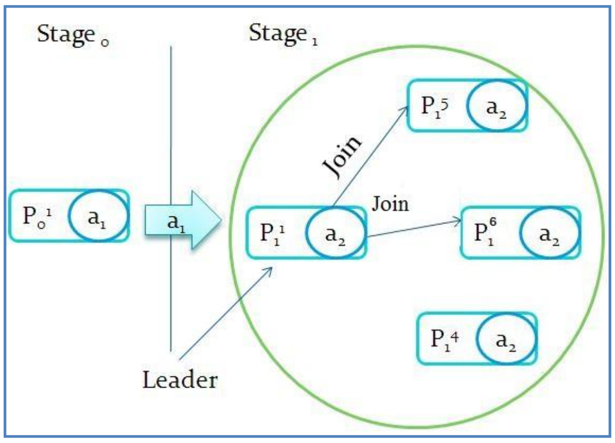

Fig: 10 Joining new agents to group when K=2

5.2.3 When any member in group member crashes (Hybrid case):

When any member of group crashes the failure detection mechanism knows and distinguishes the crashed agent is the leader or not and informs the other members in the group. If this crashed agent isn't the leader, the system acts as explained above when a non-leader agent crashes. Else if the crashed agent is the leader agent the, the system acts as explained above when leader agent crashes.

6. IMPLEMENATION AND RESULTS

The protocol just described had been implemented in a JAVA/Windows environment. The experimental environment consisted of Intel Core 2 Duo Processor T5470, 2MB L2 cache, and 800MHz FSB, and 2 GB RAM computer. Our experiments are conducted by implementing agent code using the mobile agent environment JADE [14].In this section we present the experiments we applied in our implementation to evaluate the efficiency of our proposed algorithm. This evaluating we calculated the complete task execution time and the system reliability based on different scenarios. In These scenarios we compute the effect of changing different parameters such as group members' size and failure rate on both the agent round-trip execution time and the system reliability. Reliability in our experiments is measured by applying the exponential reliability function:

Reliability = $e^{-(\frac{time}{MTBF})}$MTBF= $\frac{1}{failureratee}$

Where *MTBF: mean time between failures.

In our implementation we assumed that the Number of stages of the mobile agent itinerary is 4. An agent placed at the caller agent place stage, called controller agent, is responsible for transmitting agents when agents start roaming the network for results. At last stage, the leader agent is responsible for sending the results to caller agent when the agent job is done at all stages in the itinerary. The critical number of group members is 2, so at each stage the number of group members is greater than 2.The maximum number of the group member is equal to 10, to check the agent survivability in case of failures. When the total number of group members is equal to the critical number, the algorithm automatically adds new members to the group according to maximum number of group in this case to overcome total failure case. Failures are injected into each stage by creating a daemon that runs together with the agent code. The daemon will randomly kill one of the members of the agents group. For a proper test of the algorithm, we run the experiment for three different Failure models, Hybrid failure model, Leader failure model and None-Leader failure model. In the first, all group members are allowed to fail (including the leader). In the second, only the leader is allowed to fail. In the third, only no-leader is allowed to fail.

6.1 Effect of failure rate on the agent round trip execution time

For each of the previous models, the effect of changing the mean time between failures occurrence (MTBF) on the round trip execution time is measured and evaluated. Figure 11 shows this effect:

a.In the leader failure model case, for any group members number the round trip execution time decreases as a result of increasing the mean time between failures (MTBF) as in **Error! Reference source not found.** upper line. For example, in case of MTBF=5 the leader crash every 5 seconds that will lead to extra time consumed in the leader election process (10 second approximately). This extra time happens every 5 seconds which cause a significant increase in execution time, however the proposed algorithm can overcome this situation and complete its task successfully. As the MTBF increases the failure rate decreases and the extra execution time decreases. In case of MTBF = 60 seconds the system performed as the failure free case; because the time between failures becomes approximately equal to the time elapsed for executing the task.

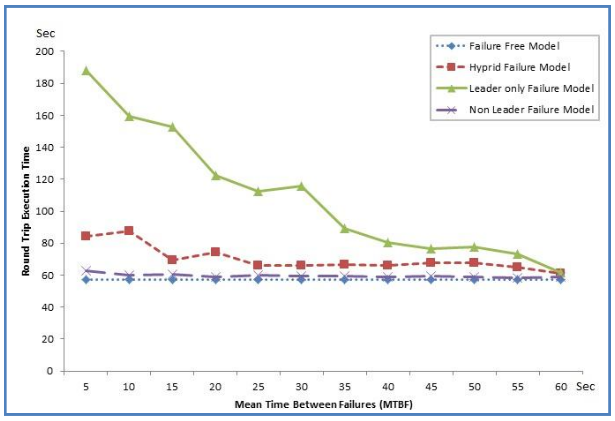

Fig 11: Execution Time with different MTBF

b.In hybrid failure model case, for any number of K (number of group members) as MTBF is increasing the agent round trip time decrease as in **Error! Reference source not found.** second line. For example, in case of MTBF=5 there are a member crash every 5 seconds. If this member is leader that means there are an extra election time (approximately 10 sec) every 5 seconds which leads to significant increase in execution time. If the crashed member is not leader and K>= 3 (minimum number of members in group) the system will continue without any time increasing despite the presence of failures .if this member is not leader and K <3 the system will create new group members until reach the group maximum number of members and join them to the group to overcome any other failures that can occur. As MTBF increases, the round trip execution time increases by an amount less than the time needed with leader failure model. When MTBF =60, the model acts as a fault free model because the time between failures becomes approximately equal to the time elapsed for executing the task.

c.In the non-leader failure model case, for any number of K when increasing MTBF the round trip time decreases as in **Error! Reference source not found.** third line. For example, in case of MTBF=5 there are a non-leader crash every 5 seconds that means non-extra election time needed. If K >= 3, the system will continue ordinary despite the presence of failures. But if K <3the system will regenerate the group to overcome any other failures occurrence. The model acts as a fault free model because no extra election was needed, however a creation time is added when K < 3, and this time is a small amount of time.

As a conclusion, in the previous three failure models, both the exactly once and non-blocking properties are achieved.

6.2 Effect of number of agents per group on the agent round trip execution time

Any group should at least 3 members and not more than 10 members. Figure 12 shows the results measured by evaluating the effect of changing the number of agents per group on the round trip execution time:

a.For leader failure model case, failures are allowed only for leaders. Any number of group members can overcome crash and complete the task, as shown in the lowest line in Figure 12.

b.For Non-leader failure model case, the leader is not allowed to fail so in case of low K and high failure rate; failure cannot be tolerated, as shown in the middle line in Figure 12. As the failure rate decrease and consequently MTBF increase the system becomes able to overcome the presence of failures. In this case, six is the minimum k that enables the system to tolerate faults when the MTBF is five. As MTBF increases the number of group members decreases until reaching the minimum number.

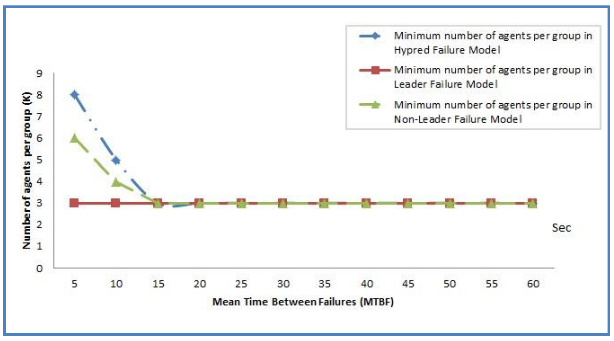

Fig 12: Minimum number of agents per group at each MTBF

For hybrid failure model case, failure is allowed for all group members; the low and high failure rate will not help in tolerating failures, as shown in **Error! Reference source not found.** the upper line. As the system starts work, eight is the minimum k that enables the system to tolerate faults when the MTBF is five. As MTBF increases the group number decreases until reach the minimum number.

From **Error! Reference source not found.** we notice that the intersection point between all failure models is happened at MTBF =15 sec, and this time is greater than the time needed for election. So, having a MTBF greater than the time for election will lead to complete the task successfully and reliably with the minimum number of group members.

6.3 Group members creation time effect

In this subsection we show how would the time needed for creating groups is affected by increasing the number of groups. As shown in **Error! Reference source not found.**, Group creation time increases slightly with the increasing the number of group members. with the maximum number of group members (10) the creation time needed still less than 1 sec. as a conclusion, the group creation time dramatically does not affect the total execution time because it increases by a small fraction with the increase in number of agents per group.

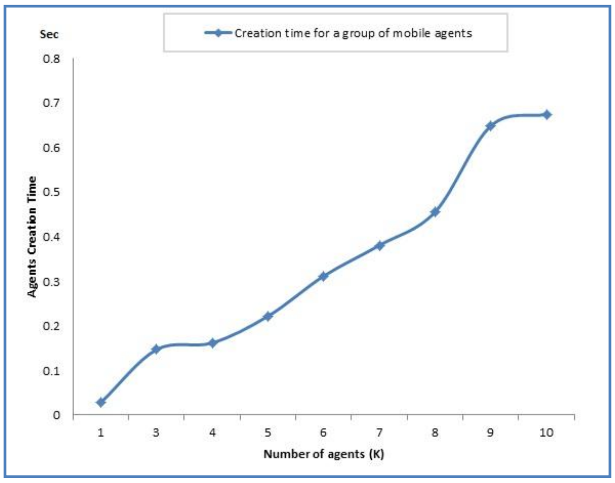

Fig 13: Group Creation Time

6.4 Effect of MTBF on reliability

For each of the previous models we measure and evaluate the effect on reliability when changing the mean time between failures occurrence (MTBF). In The following figure:

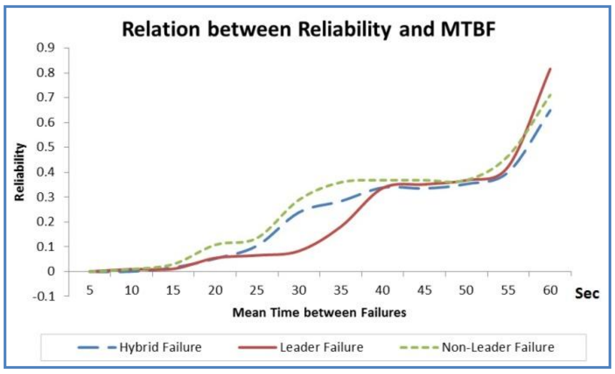

Fig 14: Average System's Reliability

As shown in **Error! Reference source not found.**, reliability improved at every point from its preceded as time increases, also it shows that in the three failure models, leader failure, non-leader failure or hybrid case, the reliability values are close to each other at any MTBF; this means that our system was not affected significantly with which agent crashed because it deals with almost all failure models in the same way. When the reliability is approximately equal to zero this means that our algorithm completes its task but the extra time needed to complete the agent task and ensuring the reliability properties is very large amount of time.

6.5 Multiple stages in the itinerary effect

Simply in all the experiments we assume that the number of stages in the itinerary is four stages. **Error! Reference source not found.** shows that the system enables mobile agent to achieve its task reliably, exactly-once execution and blocking is not occurred, in multi stages up to 17 stages in the itinerary as tested. **Error! Reference source not found.** also shows that the execution time increases linearly with the increase of the itinerary path; that means this time depends only on the time the agent consumed in each stage in the itinerary and movement time.

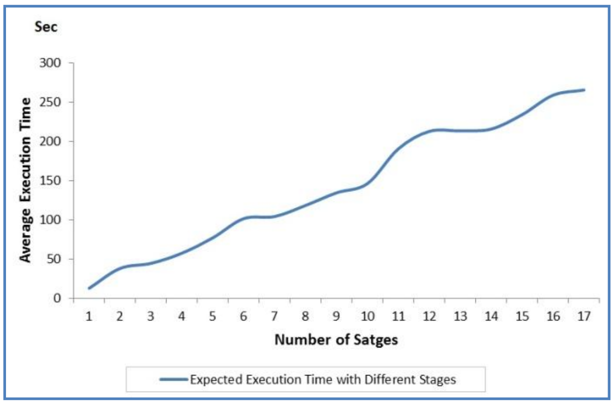

Fig 15: execution time with multiple stages in itinerary

The time needed for our reliability system is constant in each stage which consists of time consumed by membership protocol which responsible for constructing the group members and time needed by election mechanism until producing the group leader.

6.6 Various applications effect

The application used in experiments is a searching application, search for a book with the lowest price in many serves then sell it. Also we tested the system with another application which increments a number in each stage of the itinerary and get the last number after the journey .In both applications we get the same extra time needed for our algorithm with both applications; this time is the time for membership protocol and election mechanism. And this indicates that the time needed for implementing our algorithm with any mobile agent application in each stage depends only on number of agents per group and the used election mechanism, and this time independent of itinerary path stages or application type or time.

7. CONCLUSION

In this paper, a new design method based on using group communication services was presented to solve the design problem of achieving reliable Mobile Agents systems using group communication services. Reliable systems would be achieved by validating two properties, the exactly one property and the none-blocking property. The methods were tested for cases of failure free and failure occurrence. The failure free case was introduced and implemented to enable us to compare it with the failure occurrence case. As a future work there are three lines can be considered. The first line is to consider another leader election algorithm instead of the bully algorithm. The effect of changing the election algorithm may minimize the time taken to elect a leader in case of failures. The second line is to use a failure model other than the crash stop failure model. For example, crash recovery model may be used where stopped agent may restart either in determined or non-determined period of time to rejoin the group and this of course may cause dangerous effect on the results. The final line is to study the effect of the proposed work when dealing with non-trusted agents.

8. REFERENCES

[1] Peter Braun, and Wilhelm Rossak.Mobile Agents Basic Concepts, Mobility Models, and the Tracy Toolkit. s.l. : Morgan Kaufmann, 2005. 1558608176.

[2] A dynamic approach to reliable mobile agents systems using group communication services. Alaa Eldeen Sayed Ahmed, Rania Ramadan Abd El-dayem. 2009. Signal Processing and Information Technology (ISSPIT), 2009 IEEE International Symposium on. pp. 71 - 76. 978-1-4244-5949-0.

[3] Tradeoffs in implementing primary-backup protocols. Navin Budhiraja, Keith Marzullo. 1995. Parallel and Distributed Processing, 1995. Proceedings. Seventh IEEE Symposium on . pp. 280 - 288. 0-81867195-5 .

[4] Using active clients to minimize replication in primary-backup protocols. Daniel J. Rosenkrantz, S.S. Ravi Parvathi Chundi, Ragini Narasimhan,. 1996. Computers and Communications, 1996., Conference Proceedings of the 1996 IEEE Fifteenth Annual International Phoenix Conference on. pp. 96 - 102. 0-7803-3255-5 .

[5] Stefan Pleisch, Andr´e Schiper.Approaches to Fault-Tolerant Mobile Agent Execution. 2001. p. 13.

[6] Design and Evaluation of a Fault-Tolerant Mobile-Agent System. Michael R. Lyu, Xinyu Chen, and Tsz Yeung Wong. 5, s.l. : IEEE INTELLIGENT SYSTEMS, 2004, Vol. 19.

[7] Evaluation and Checkpointing of Fault Tolerant Mobile Agents Execution in Distributed Systems. Hodjatollah Hamidi, Abbas Vafaei, Seyed Amirhassan Monadjemi. Isfahan, Iran : JOURNAL OF NETWORKS, 2010, Vol. 5.

[8] FATOMAS - A Fault-Tolerant Mobile Agent System Based on the Agent-Dependent Approach. Stefan Pleisch, Andr´e Schiper,. s.l. : IEEE Computer Society Press, Conference on Dependable Systems and Networks, 2001. pp. 215 - 224. ISBN: 0-7695-1101-5 .

[9] The mobile groups approach for the coordination of mobile agents. Raimundo J. A. Macêdo, Flávio M. Assis Silva. 3, s.l. : Parallel and Distributed Computing, 2005, Vol. 65.

[10] GCS-MA: A group communication system for mobile agents. Wei Xu, Jiannong Cao, Beihong Jin , Jing Li, Liang Zhang. 3, Hong Kong, China : Journal of Network and Computer Applications, 2007, Vol. 30.

[11] Group Communication: From Practice to Theory. Schiper, Andr´e. Berlin - Heidelberg : J. Wiedermann et al. (Eds.): SOFSEM, 2006.

[12] Dynamic Group Communication. Andr´e Schiper. s.l. : Infoscience, 2003. Large Scale Distributed Systems - ACM Proc of the European SIGOPS Workshop.

[13] Lampson, Butler W. Atomic Transactions. Computer Science 105. 1981.

[14] JADE. [Online] http://jade.tilab.com/.