Fault Tolerance Approach in Mobile Agents for Information Retrieval Applications Using Check Points

Rahul Hans¹, Ramandeep kaur²
¹Guru Nanak Dev University, Amritsar, ²Guru Nanak Dev University, Amritsar
¹rahulhans@gmail.com, ²ramansidhu1985@gmail.com

Abstract

Mobile agents have emerged as major programming paradigm for distributed applications. Mobile agents are the intelligent programs that act autonomously on behalf of a user and can migrate from one host to another host in a network in order to satisfy the requests made by their clients. A prerequisite for their use, however, is that they should be executed reliably independent of failures. Improving the survivability of mobile agents in presence of agent server failures is an important issue in order to guarantee continuous execution of mobile agents. Thus it is very important to make mobile agents fault tolerant. In this paper, we propose fault tolerance mechanism for the scenarios where the agent stops its execution due to fault on any server in the itinerary. Our approach makes use of checkpointing, partial results or data retrieved and the address of last host visited is saved prior before the agent visits the next host in the itinerary. The proposed mechanism has been implemented on the Aglets mobile agent system and evaluated in terms of parameters such as round trip time, Reliable migration time, Check point time. The results show the improvement in reliability and performance, especially for mobile agents in Internet application.

1. Introduction

All An agent-based computer system is a distributed computing environment in which mobile autonomous processes called mobile agents operate on behalf of users [1]. Mobile agents are programs which are dispatched from a source computer and run among a set of networked servers until they are able to accomplish their task. Mobile agent computing paradigm is different from others because not only data but the code acting on the data is also transported among the nodes. This transportation of the code makes the application developed more flexible. Mobile agents are proactive, reactive and cognitive [4]. An agent can suspend its execution, migrate to other node and restart its execution there at the other node. There are many issues related to reliability of mobile agents. Like an agent should not fail due to any failure in software or hardware components.

Agents can fail if host fails or agent might not reach the desired host. These failures may lead to a partial or complete loss of the agent. So the fault tolerant mobile agent systems should be created [9]. In this paper, we propose fault tolerance mechanism for information retrieval applications. An information retrieval mobile agent visits a sequence of remote hosts consuming information that satisfies criteria provided by its user [12]. In which the agent stops its execution due to fault on any server in the itinerary.

Most of the techniques that have emerged so far employ a form of replication to provide fault tolerance in mobile agent execution. Some of the desired properties for the fault tolerant execution of mobile agents are non-blocking and exactly once. Non-blocking property ensures that the agent execution can make progress at any time and exactly-once execution property prohibits multiple executions of the agent. As many of mobile agent applications require an agent to be executed exactly once [3].

The rest of the paper is organized as follows. Section 2 presents an overview of some related work for the fault tolerance in mobile agents and discusses some of the existing fault tolerant techniques proposed by various authors in mobile agents system section 3 briefly discusses about aglets platform for mobile agents section 4 describes the proposed fault tolerant approach section 5 discusses implementation and performance study section 6 briefly gives us conclusion and section 7 discusses future work.

2. Related Work

Distributed systems today are ubiquitous and enable many applications, including client-server systems, transaction processing, World Wide Web, and scientific computing and many others. The vast computing potential of these systems is often hampered by their susceptibility to failures [5].
In mobile agent computing environment any component of the network machine, link, or agent may fail at any time; thus may preventing mobile agents from continuing their executions. Therefore, fault-tolerance is a vital issue for the deployment of mobile agent systems. Fault tolerance schemes for mobile agents to survive agent server crash failures are complex since there is no control over remote agent servers. Many techniques have been developed to add reliability and high availability to distributed systems which can be broadly classified into two kinds replication and check pointing.

In replication scheme an agent is replicated and sent to several sites for each stage so that the agent can survive site failures [2]. When one server is down, it can use the results from other servers in order to continue the computation. The advantage of this approach is that the computation will not be blocked when a failure happens. But this fault-tolerance scheme is expensive since it has to maintain multiple physical servers for just one logical server and it is not cost-effective to maintain multiple servers.

2.1 Using the CAMA Framework

In [8] author introduces the CAMA the Context-Aware Mobile Agents framework which supports application-level fault tolerance by providing a set of abstractions and a supporting middleware that allow developers to design elective error detection and recovery mechanisms.

CAMA supports system fault tolerance through exception handling and structured agent coordination. There are three basic operations available to the CAMA agents for catching and raising inter-agent exceptions: raise, check and wait. These functionalities are complementary and orthogonal to the application level mechanism used for programming internal agent behavior.

The advantage of this approach is that the exception handling allows fast and effective application recovery by supporting flexible choice of the handling scope and of the exception propagation policy and also it deals with agent’s failures and connection disconnection problems. Its drawback is that it can be blocking in the case when an exception is raised to the agent which has left the scope.

2.2 Chameleon: Adaptive fault tolerance using mobile agent

Fault tolerance is usually provided through dedicated hardware or dedicated software. Unfortunately, dedicated fault tolerant architectures offer a static level of fault tolerance and these architectures are often oriented towards specific classes of applications. It is not cost effective to provide dedicated hardware based fault tolerance to each application. The pressing issue then becomes the best way in which to achieve high dependability with off-the-shelf, unreliable hardware and off-the-shelf applications.

Chameleon provides an adaptive Infrastructure that supports different levels of availability requirements simultaneously in a single, heterogeneous, clustered environment [11].

The advantage of this approach is that provides a flexible architecture through which adaptive fault tolerance may be achieved in an unreliable and heterogeneous network and it deals with both agent and system failure. It has a disadvantage that it suffers from blocking if any of the nodes fails during execution.

2.3 Transient Fault Tolerance in Mobile Agent

Mobile agents code often experience transient faults resulting in a partial or complete loss during execution at a host machine [10]. Author describes how to detect and recover random transient bit-errors at an agent before starting its execution at a host after its arrival at a host in order to maintain availability of an agent by comparing an agent’s states by using time and space redundancy. It can be blocking if bit error cannot be recovered by any of the replicas. This technique provides high performance as provide fault tolerance at low level.

The advantage of this technique is that it is good enough to detect multiple soft errors and corrections thereof with an affordable redundancy in both time and memory space for gaining higher fault-tolerance.

2.4 Region-based Stage Construction Protocol

The replication based fault tolerant protocols are classified into two approaches spatial replication based approach and Temporal replication based approach [4,21].

So region based stage construction protocol is used for fault tolerant execution of mobile agents in a multi-region mobile agent computing environment. It uses new concepts of quasi-participant and sub stage in order to put together some places located in different regions within a stage in the same region.

A mobile agent $a_i$ executes tasks on a sequence of nodes. Each action that $a_i$ execute on a place $p_s$ is called a step each step consists of a set of places called a stage $S_i$ [6]. $p^s_i$ at $S_i$ is called a worker, the others are called participants. When a worker fails, one of participants is elected as a new worker and takes over the action of the previous worker. To
provide the exactly once property of a mobile agent execution, voting and agreement protocols are needed at each stage [7, 3]. In a multi-region mobile agent computing environment, places within a stage can be located in the same or different regions [7].

The main advantage of this protocol is that this approach reduces the overhead of stage works about two times as low as previous protocols so that it decreases the total execution time of mobile agents.

2.5 Using the Witness Agents in Linear Network

In this approach server and agent failures are detected and recovered by the cooperation of agents with each other. In [9], in order to detect the failures of an actual agent as well as recover the failed agent, another types of agent are used, namely the witness agent, to monitor whether the actual agent is alive or dead.

A communication between both types of agents is done by sending Direct and Indirect messages. The actual agent assumes that the witness agent is at the server that it has just previously visited and communication is done by passing direct messages. When actual agent is unable to send a direct message to a witness agent for this purpose there is a mailbox at each server that keeps those unattended messages. These type of messages are called the Indirect Messages. Every server has to log the actions performed by an agent. This protocol is based on message passing as well as message logging to achieve failure detection.

As long as the witness-dependency is preserved, agent failure detection and recovery can always be achieved. In order to handle this failure series, the owner of the actual agent can send a witness agent to the first server $s_0$, in the itinerary of the agent with a timeout mechanism.

This approach has a drawback that the existing procedure consumes a lot of resources along the itinerary of the actual agent as the itinerary becomes longer, more witness agents and probes are necessary, so system complexity increases.

2.6 Adaptive Mobile Agent System using Dynamic Role based Access Control

Adaptive Mobile Agents are designed to accept additional roles [1], while working inside a special environment called context-aware environment which performs the task of sharing and allocating the roles to the mobile agents present in the environment. It generates the rules based on conditions and the mobile agents acquire roles based on the instructions given by the environment, the Adaptive Mobile Agents must cooperate with one another and with the environment to acquire roles.

Roles are being assigned to restrict or grant access to a resource. This mode of restricting or granting access to a resource is called Role Based Access Control (RBAC) which plays a main role in managing security of data. The communication between various components is carried through communication messages [1].

The advantage of this technique is that as mobile agents are already inside the system, it does not require any sort of external communication. As a result, the time to create and dispatch a new mobile agent is saved and the response time becomes less.

2.7 Exception Handling Approach for Information Retrieval Applications.

In this approach authors assume that a mobile agent crashes when its current local agent server halts execution, thus terminating all active mobile agents. Such an event is encountered when the host running the agent server platform crashes or a fault is encountered in the agent server process. The author has proposed two exception handler designs the mobile time out design mobile shadow design [12].

An agent server AG offers a set of services $\{s_1, s_2, \ldots, s_n\}$. A service $s_i$ is a software component that a mobile agent manipulates by issuing method calls. Both a service and mobile agent define its own set of internal or local exceptions $I = \{e_1, e_2, \ldots, e_n\}$ and associated handlers $H = \{h_1, h_2, \ldots, h_n\}$ that serve to provide corrective action. An internal exception occurrence $e_i$ triggers the exceptional activity $h_i$ within the service or mobile agent. If the exception is successfully handled normal activity resumes. A service completes its execution by providing a response to the mobile agent that made the service request.

The advantage of this approach is that coordination among the replicas of the agent is directly through message passing and deals with both agents and node failures. Also it is highly dependable and efficient technique.

3. Aglets Mobile Agent Platform

Aglets is a Java mobile agent platform and library that eases the development of agent based applications. An aglet is a Java agent able to autonomously and spontaneously move from one host to another. The term aglet is indeed a portmanteau word combining agent and applet[13]. Aglets are completely made in Java, granting an high portability of both the agents and the platform. Aglets include both a complete Java mobile agent...
platform, with a stand-alone server called Tahiti, and a library that allows developer to build mobile agents and to embed the aglets technology in their applications.

This model was designed to benefit from the agent characteristics of Java while overcoming some of the above-mentioned deficiencies in the language system. Most notably, the model defines a set of abstractions and the behavior needed to leverage mobile agent technology in Internet-like open wide-area networks. The key abstractions are aglet, proxy, context, message, future reply, and identifier[13].

When aglets are well and running they take up resources. To reduce their resource consumption, aglets can go to sleep temporarily, releasing their resources (deactivation), and later be brought back into running mode (activation). Finally, multiple aglets may exchange information to accomplish a given task (messaging). The aglets’ fundamental operations, namely, creation, cloning, dispatching, retraction, deactivation, activation, disposal, and messaging.

4. Fault Tolerance in mobile agents using Check points

4.1 Failure assumptions
The following failure assumptions are used:

- A mobile agent crashes when its current local agent server halts execution, thus terminating all active mobile agents. Such an event is encountered when the host running the agent server platform crashes or a fault is encountered in the agent server process.
- No stable storage mechanism is provided at visited agent servers for the recovery of executing agents.
- Reliable communication links are assumed.
- All agent servers are correct and trustworthy.
- The home agent server is always available.
- A mobile agent consumes information at agent servers. The state of agent servers is not modified.

4.2 Notations

- $S_o$: Originator host
- $S_i$: Hosts visited by agent during its movement in the network ($1 \leq i \leq n$)
- $MA_i$: Original Mobile agent conating information from ith server originally launched
- $MA_{rep}$: Replicated copy of original Mobile agent
- $MA_p$: Mobile agent carrying partial results
- $MSG_{fault}$: Message sent to host about the occurrence of fault.
- $LT_{MA}$: Life time of mobile agent[14]
- $RT_{wftma}$:Normal round trip time without Fault Tolerance mechanism.
- $RT_{ftma}$: Round trip time with Fault Tolerance mechanism.
  - $I_i$: Information collected from host $S_i$
  - CP Time: Check point time
  - RM time: Reliable migration time

In our work, we implemented our proposed mechanism on aglets-2.0.2 for experimental evaluation. The scheme was implemented to ensure that the host server which dispatches the mobile agent at any point of time should receive the information from the remote server in minimum amount of time.

The scenario considered is the web based e-marketplace that provides user with the information on the products for sale by collecting the prices and comparing the prices of the set of products like computers as specified by the user [14]. Sometimes the information needs to be collected in real time for various applications such as stock market, online shopping, etc. from different hosts.

Servers are selected dynamically by freely roaming mobile agent over the network. The address of the first server is assigned at the host and the address of the remaining servers is dynamically picked by the agent from the server on which it is currently executing.

The originator is assumed to be always connected to the network to collect the results. Implementing the proposed solution, an agent is originally launched from the originator host server. Under general operation of a mobile agent it returns to the originator after the expiry of its $LT_{ma}$.

The implementation scheme used as shown in Figure 2, requires that the server $S_i$ having received the mobile agent from the host server $S_o$, fetches the information $I_i$ from the server $S_i$ and after the execution of the agent on the server $S_i$, the agent moves to next server $S_{i+1}$ and again retrieves the value from server $S_{i+1}$ and after completing its execution it moves to next server $S_{i+2}$ and repeats the same process.
After completing its execution on the first three servers of the itinerary the agent puts the check point CHK on the server and MA moves to the host server So and saves the values retrieved from the first three servers on the host. After saving the values and adding check points the agent moves to the next server in the itinerary that is the S_{i+3}, and repeats the same process for every three servers in the itinerary and returns to the originator after the expiry of its LTMA.

![Fig 1.](image)

As the agent is collecting data from various servers in the itinerary and adds the checkpoints and saves the data to the host system. At some point of time agent stops its execution due to any fault on the server and the agent does not move further in the itinerary. At this situation immediately a message MSG_{fault} is send to the host.

![Fig 2.](image)

To mask the effect of the fault when ever host receives the message MSG_{fault}, the host immediately sends the replicated copy MA_{rep} of the original agent to the immediate check point before the faulty server. The replicated agent is intelligent enough that it already knows the location of the fault and the immediate checkpoint before the fault. The replicated agent moves in the itinerary and repeats the same process as of the original agent MA and executes till the expiry of its LTMA. In the same way whenever a fault occurs on any sever the same process is repeated to achieve fault tolerance.

5. Implementation and Analysis

Proposed scheme had been implemented in AGLETS-2.0.2 by conducting three experiments on a setup of network containing 12 different nodes each having same configuration and installed AGLETS-2.0.2 on each server. For gauging the performance of the implemented scheme we intentionally made some Servers behave as Faulty and got the agent execution stop.

**Experiment I: Effect on Round trip time without any fault.**

Round trip time is the time taken by an agent to complete its itinerary by visiting each server S_1, S_2, ..., S_{12} and return back to the host server or the originator S_0. While visiting each server it collects the information I_i for which it is programmed from each server S_i. The normal execution time of the agent on each server is assumed to be 1 sec or 1000ms.

The normal round trip time of agent RT_{wftm} without any fault is compared to the round trip time of the agent with fault tolerance mechanism (FTMA) RT_{ftm} without any fault. The results show that the round trip time of the agent with FTMA increases as the time taken to checkpoint and save the information also adds in it.

![Round Trip Time](image)

In our experiment we have considered an itinerary consisting 6 and 12 servers and the normal round trip time without FTMA is compared with the round trip time having FTMA mechanism and adding checkpoints after every three servers in the itinerary which adds to the overheads and leads to the increase in the round trip time of the itinerary.

The overheads are compared for the itinerary of various lengths in the table below these overheads are all because of the time which an agents uses to check point the data and the location of the last server visited by the agent in the itinerary, which keeps on growing as the size of itinerary increase. It depends on after how many
servers the agent should check point the data at the host server.

These overheads are measured in terms of Reliable migration time (RM time) and Check point time (CP time). RM time is the time taken by the agent to complete its itinerary making the faults and CP time is the time taken by agent to go back to the originator or host server to check point the data retrieved and address of the next host in the itinerary.

Experiment 2: Effect on Round trip time when fault occurs on any fault.

In this experiment we compare the round trip times of the agent to complete its itinerary when fault occurs on various nodes. The normal round trip time to complete an itinerary when fault occurs on any server RT_{wftm} is more than RT_{ftm} because when fault occurs on any server the host is notified about the fault; it sends a replicated agent or a copy of the original agent MA_{rep}, which starts its itinerary from the beginning that is from the server S_{1} again.

The above experiment has been performed on 12 different servers and for RT_{ftm}. We have assumed checkpoint after every three servers. For implementation and result purpose an agent was manually killed by killing the thread of the agent on the particular server to create the fault. The time taken by the agent to visit again all the nodes which have been already visited by the MA that is the original agent adds to the overheads, so the time taken to complete the round trip increases in this case as compared to RT_{ftm}.

Initially both RT_{ftm} and RT_{wftm} are same when we assume a fault at server 4 but as we assume fault on any server after 4th sever the RT_{ftm} decreases as compared to RT_{wftm}. In case the replicated agent MA_{rep} starts its itinerary from the immediate check point before the faulty server so the there are no overheads to visit again all those servers which have been already visited by the original agent. Compare the performance of both the RT_{ftm} and RT_{wftm}. We have taken an itinerary of 12 servers, when fault occurs on the 4th server of the itinerary both RT_{ftm} and RT_{wftm} are same. When fault occurs on 7th server RT_{wftm} increases as compared to RT_{ftm} and same is the case when fault occurs on 9th server.

<table>
<thead>
<tr>
<th>Server number</th>
<th>4</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Without FTMA (RT_{wftm})</td>
<td>16000</td>
<td>19000</td>
<td>21000</td>
</tr>
<tr>
<td>Time with FTMA (RT_{ftm})</td>
<td>16000</td>
<td>16000</td>
<td>18000</td>
</tr>
</tbody>
</table>

Experiment 3: Effect on Round trip time when fault occurs on multiple servers in a single trip.

In this experiment we compare the round trip times of the mobile agent without FTMA (RT_{wftm}) and with FTMA.
(RT_{ftm}) when fault occurs on multiple servers in single trip. For RT_{ftm} when agent MA moves on the servers in the itinerary and whenever it finds a faulty server, the replicated agent MA_{rep} starts from the first server S_1 in the itinerary so the overheads of visiting those servers which have been already visited by original agent MA adds to the total round trip time. For RT_{ftm} the agent doesn’t rollback and visits those servers again which are already visited by the original agent MA because in this when fault occurs the replicated agent MA_{rep} starts its itinerary from the checkpoint immediately before the faulty server.

<table>
<thead>
<tr>
<th>Faults on multiple Servers</th>
<th>Server 4 and 7 in single trip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Without FTMA (RT_{ftm})</td>
<td>23000 ms</td>
</tr>
<tr>
<td>Time with FTMA (RT_{ftm})</td>
<td>17000 ms</td>
</tr>
</tbody>
</table>

6. Conclusions
In this paper, we have proposed a fault tolerance mechanism for the scenarios where the agent stops its execution due to fault on any server in the itinerary. Our approach makes use of check pointing, partial results and the address of last host visited is saved prior before the agent visits the next host in the itinerary.

Whenever a fault occurs, to mask the effect of the fault the host immediately sends the replicated copy of the original agent to the immediate check point before the faulty server. The in-depth analysis of this technique show us good results by improving the round trip time of the agent. Since after occurrence of fault, the replicated agent need not roll back to the first server as it starts moving from the checkpoint immediately before the faulty server.

Check pointing and saving the data repeatedly leads to increase in the communication overhead but for time sensitive applications the overhead may be bearable.

7. Future Work
From the future point of view, whenever an agent does not reaches the desired server due to network congestion the host assumes it to be failed and it sends a replicated copy of it mean while the original agent also reaches the destination which could lead to violation of exactly once property. So this approach should be developed further to avoid violation of exactly once property.

References


