Complete Analysis of Fault Tolerance Schemes in Mobile Agents for a Reliable Mobile Agent Computation

Elijah and Muhammad Naveed Dilber

Abstract— In recent times, mobile agents have come out as foremost programming standard for distributed applications. These agents are the smart programs that operate autonomously in support of a user and can move around from one host to another host in a network with the intention of satisfying the requests made by their clients. A precondition for their utilization, on the other hand, is that they must be carried out consistently independent of failures. Enhancing the survivability of mobile agents in existence of agent server failures is a significant issue with the aim of continuous execution of mobile agents. As a result, it is extremely significant to make mobile agents fault tolerant. Here, this paper surveys the state of the art of fault tolerance schemes in mobile agents, formulated by various authors. These schemes provide a consistent mobile agent computation even in existence of failures.

Keywords--- Mobile Agent, Fault Tolerance, Checkpointing, Rollback Recovery.

I. INTRODUCTION

An agent-based computer system is a distributed computing environment in which mobile autonomous processes called mobile agents operate on behalf of users [13]. 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 [11].

In this survey, we focus on various existing techniques to implement fault tolerance in mobile agents. 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 provides an overview of fault tolerance in mobile agents system. Section 3 discusses some of the existing fault tolerant techniques proposed by various authors in mobile agents system and section 4 briefly gives the conclusion and some future work.

II. FAULT TOLERANCE IN MOBILE AGENTS SYSTEM

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.

In the check pointing scheme, the intermediate states of an agent are saved into a stable storage so that the agent can resume the execution from the saved state in case of a failure [2]. When a failure occurs, the agent can resume the execution from the saved state, if an application takes periodic checkpoints, then in case of a failure, it can restart from the latest checkpoint, by avoiding losing all the computation that
was carried before that checkpoint. The check pointing scheme also shows a very stable performance and it is sensitive only to the blocking time when the failure rate is high.

III. EXISTING FAULT TOLERANCE TECHNIQUES IN MOBILE AGENTS

Fault tolerance techniques in mobile agents can also be classified as agent centric technique and system centric technique [19]. Agent centric technique provides fault tolerance within the mobile agents and system centric technique moves the monitoring and fault recovery into a separate software entity. Some of the fault tolerant techniques in mobile agents proposed by various authors are summarized below:

A. Improving Fault Tolerance by Replicating Agents

Agent replication is the act of creating one or more duplicates of one or more agents in a multi-agent system. In [19] author proposed that using redundancy by replication of individual agents within a multi-agent system is one possible approach for improving fault tolerance. Each of these duplicates is capable of performing the same task as the original agent. The group of duplicate agents is referred to as a replicate group and the individual agents within the replicate group are referred to as replicates. Once a replicate group is created in a mobile agent system, if that replicate group is visible to the rest of the mobile agent system, there are several ways that agents can interact with it by sending messages.

A transparent replicate group proxy acts as an interface between the replicates in a replicate group and the rest of the mobile agent system. Proxies provide two important functions, they make the replicate group appear to be a single entity and they control execution and state management of a replicate group. Using proxies means that replication is transparent to agent developers. No extra work is required to interact with a replicate group.

The advantage of this method is that an acceptable degree of overhead is imposed by replication on the system and it improved system reliability. It can be both blocking and non-blocking. Communication, read/write consistency and state synchronization issues can be catered as well by the technique using transparent proxies.

B. Using the CAMA Framework

In [10] 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 raise operation propagates an exception to an agent or a scope. Two other operations, check and wait are used to explicitly poll and wait for inter-agent exceptions.

Systematic use of exception handling allows developers to design mobile agent applications tolerating a broad range of faults, including disconnections, agent mismatches, malicious agent activity, violations of system properties, potentially harmful changes in the system environment and many more.

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. According to author it provides high performance and it is a dependable framework.

C. 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 [16]. Three broad classes of agents:

a. Managers: They supervises other agents and recovers from failures in their subordinates. Primary managers include the Fault Tolerance Manager (FTM).
b. Daemons: They allow Chameleon to access a node, provide agent error detection, and provide the means of communication among themselves.
c. Common Agents: Common agents implement specific techniques for providing application required dependability.

In the initialization phase, the FTM collects information about the system configuration. Initialization agents are sent to hosts to obtain this data and to install the host daemons on participating machines.

After the successful initialization, Chameleon is ready to accept user requests. When a request arrives, the FTM designates a query agent to acquire the necessary information on the application specifics, such as the required availability level, needed system resources etc. Based on this information, the FTM identifies the necessary fault tolerance strategy and designates set of agents to initiate and monitor the application. Creation of agents is performed according to a predefined procedure.

Agents designated to support the application execution migrate to the selected nodes, install themselves and initiate the application execution and also a Surrogate Manager is spawned by the FTM and associated with the application. It maintains copy of the system information supported by the FTM and provides reliable communications with the user.
Since the agent itself may fail, it is watched by the Host Daemon, which is capable of notifying the Surrogate Manager about agent failures. To operate reliably, the FTM must be resilient to errors. A possible solution is to support a passive backup FTM, which supports the system information and is updated each time the system state changes.

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.

D. 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 [14]. 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 establishes test conditions within a single version of a mobile agent program, rather than requiring multiple and distinct versions of the same mobile agent program running concurrently on many machines.

This scheme is able to detect and repair transient errors in mobile agent code upon its arrival at a host. This is effective at the cost of an affordable redundancy in both time and space, without an increased monetary budget.

Author assumes that byte-errors that may occur after a mobile agent starts execution at a host will be detected by a host's intrinsic error detection mechanism in the case when we do not execute all the three copies of an agent in a sequence in order to vote upon their answers for a majority one. On using extra time redundancy we compute run-time checksums of the image in execution and compare it with the previously computed (and saved) checksums, for detection of run-time errors that might occur once the execution of a mobile agent has started. The pseudo code of the proposed protocol is described in [14].

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.

It can be blocking if bit error cannot be recovered by any of the replicas but probability of the situation is very small. This technique provides high performance as provide fault tolerance at low level.

E. 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]. In spatial replication the agent scheme is replicated and sent to several sites for each stage so that the agent can survive site failures. The Temporal approach is based on the check pointing the code and state of agent on the previous site [11].

Spatial based approach has a drawback that when a mobile agent proceeds to the next stage, additional cost of communication is incurred. To reduce this some previous works have proposed stage construction protocols but when they are applied to a multi-region mobile agent computing environment, they construct stages regardless of regions, so that the overhead of stage works is increased.

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 ai executes tasks on a sequence of nodes. Each action that ai execute on a place pi is called a step each step consists of a set of places called a stage Si [6]. pwi at Si 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].

A quasi-participant qpi is a place which substitute for an original participant located in different regions in order to put some places together in the same region. If a worker fails, it hands over its delegation to the real-participant and terminates. It just participates in stage works without providing the service which is provided by a real-participant the action of quasi-participants and real-participants should be executed in one step, so they should be constructed into one stage. For this it uses sub stage SS, which is another stage within a stage. A SS should satisfy following property [18]:

\[
\begin{align*}
\text{a. } & (SSi \cup S_{i}) \ni \forall(p^w_i \in SSi) \text{ where } S_i = QS, URS_i \\
\text{b. } & SS_i \rightarrow SS_{i+1} \text{ if } p^w_i \text{ fails.}
\end{align*}
\]

The main advantage of this protocol is that this protocol 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

F. 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 [11], 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
The actual agent is labeled as \( \alpha \) and the witness agent as \( W_{i-1} \). After \( \alpha \) has arrived at \( S_i \), it logs a message \( \log_i^{\text{arrive}} \) on the permanent storage in \( S_i \) to let the coming witness agent know about its successful arrival at \( S_i \) and \( \log_i^{\text{leave}} \) message for completion of its computation on \( S_i \) and it is ready to travel to the next server \( S_{i+1} \).

The witness agent only listens to the messages coming from the actual agents. When \( W_{i-1} \) is in \( S_{i-1} \), it expects two messages \( \text{msg}_{i-1}^{\text{arrive}} \) and \( \text{msg}_{i-1}^{\text{leave}} \). If a server failure happens just before \( W_{i-1} \) migrates to \( S_{i} \), then no one can guarantee the availability of the actual agent. \( W_{i-1} \) spawns \( W_i \) and travels to \( S_i \). The procedure goes on until \( \alpha \) reaches the last destination in its itinerary. The youngest witness agent is witnessing the actual agent. On the other hand, the elder witness agents are neither idle nor terminated, they have another important responsibility: an earlier witness agent monitors the witness agent that is just one server closer to the actual agent in its itinerary. That is:

\[
W_0 \rightarrow W_1 \rightarrow W_2 \rightarrow \ldots \ldots . W_{i-1} \rightarrow W_i \rightarrow \alpha
\]

Where “\( \rightarrow \)” represents the monitoring relation. This Witness-Dependency cannot be broken as \( W_{i-1} \) is monitoring \( \alpha \) and \( W_{i-2} \) is monitoring \( W_{i-1} \). Let \( S_{i-1} \) crashes and then also \( S_i \) crashes then \( W_{i-1} \) is lost, hence no one monitoring \( \alpha \). Therefore, a mechanism to monitor and to recover the failed witness agent is needed. It is achieved by preserving the witness dependency: the recovery of \( W_{i-1} \) can be performed by \( W_{i-2} \), and \( \alpha \) can be recovered by \( W_{i-1} \). It is assumed that the owner of the actual agent hold an array, \( W \), for the location of the witness agent.

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.

### G. Adaptive Mobile Agent System Using Dynamic Role based Access Control

Adaptive Mobile Agents are designed to accept additional roles [20], 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 Dynamic rule-role based access control system in an agent environment consists of a Context-aware environment, The Dynamic Rule Generator, The Role Controller and An Adaptive Mobile Agent [20]. The roles are assigned based on the rules generated by the Dynamic Rule Generator. The Role Controller assigns the roles to the Mobile Agents. An Adaptive Mobile Agent has two types of roles present in it. They are the Basic role and inherent role that is assigned to the Mobile Agent at the time of its origin. The adaptive role is the additional role which is being acquired by the mobile agent from the role controller. The communication between various components is carried through communication messages [20].

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.

### H. 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 [17].

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 \( IH = \{h_1, h_2, \ldots, h_n\} \) 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. A corrective action performed by a service or mobile agent in response to an internal exception is application specific and may involve dispatching a compensating mobile agent CM to interact with a service \( s_i \) at a remote agent server AGj. A service \( s_i \) signals a set of external exceptions \( E = \{\text{interface, failure}\} \) to a mobile agent when it fails to satisfy the service request. Upon receiving an external exception the mobile agent may retry service \( s_i \), locate a service at an alternative agent server or report back to the home agent server or parent mobile agent [17]. A mobile agent is dispatched a second time if it crashed or reported back to its owner with a failure exception.

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.

Comparative summary of various mobile agent fault tolerant technique proposed by various authors with some their drawbacks are shown in the Table 1 below.
between a failed agent and an agent that is delayed by slow communication links, as to avoid multiple execution of the agent. This approach can be considered without a coordinator for this purpose, its data should be saved on stable storage periodically and also the whole record of witness agents would be lost, for this purpose the issue of fault tolerance is to keep the exactly-once execution property. It is not a stricter failure model, suffers blocking.

### Exception Handling Approach

**Description**: Exception handling is used to achieve fault tolerance.

**Drawbacks**: Overheads occurs, it can blocking and complexity of system increases.

### IV. CONCLUSION AND FUTURE WORK

This paper discusses the fault tolerance techniques in mobile agents presented by various authors. An important issue of fault tolerance is to keep the exactly-once execution and non blocking property. When a failure is detected by agent owner, it launches a new agent, which requires the ability to correctly detect the crash of the agent that is to distinguish between a failed agent and an agent that is delayed by slow communication links, as to avoid multiple execution of the agent.

In future, some of the drawbacks of the above discussed techniques should be removed so as to make them more fault tolerant. Like in the Witness Agents approach failure of the agent owner is not considered, if the agent owner crashes whole record of witness agents would be lost, for this purpose its data should be saved on stable storage periodically and also this approach can be considered without a coordinator for holding the array, this array can be part of agent’s data.

### REFERENCES


