Palestine Polytechnic University
Deanship of Graduate Studies and Scientific Research
Master of informatics

Integrated Replication-Checkpoint Fault Tolerance Approach for Mobile Agents Systems

Submitted by:

Suzanne Youcef Mohamed Sweiti

Thesis submitted in partial fulfillment of requirements of the degree Master of Science in Informatics
December, 2015

The undersigned hereby certify that they have read, examined and recommended to the Deanship of Graduate Studies and Scientific Research at Palestine Polytechnic University the approval of a thesis entitled: **Integrated Replication-Checkpoint Fault Tolerance Approach for Mobile Agents Systems**, submitted by **Suzanne Youcef Mohamed Sweiti** in partial fulfillment of the requirements for the degree of Master in Informatics.

Graduate Advisory Committee:

Dr. Amal Al Dweik (Supervisor), Palestine Polytechnic University.

Signature:_____ Date:_____

Dr. Radwan Tahboub (Internal committee member), Palestine Polytechnic University.

Signature:_____ Date:_____

Dr. Amjad Rattrout (External committee member), The Arab American University.

Signature:_____ Date:_____

Thesis Approved

Dr. Murad Abu Subaih
Dean of Graduate Studies and Scientific Research
Palestine Polytechnic University

Signature:_____ Date:_____

DECLARATION

I declare that the Master Thesis entitled **"Integrated Replication-Checkpoint Fault Tolerance Approach for Mobile Agents Systems"** is my original work, and hereby certify that unless stated, all work contained within this thesis is my own independent research and has not been submitted for the award of any other degree at any institution, except where due acknowledgment is made in the text.

Suzanne Sweiti

Signature:_____ Date:_____

STATEMENT OF

PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for the master degree in Informatics at Palestine Polytechnic University, I agree that the library shall make it available to borrowers under rules of the library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from, reproduction, or publication of this thesis may be granted by my main supervisor, or in his absence, by the Dean of Graduate Studies and Scientific Research when, in the opinion of either, the proposed use of the material is for scholarly purposes.

Any coping or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Suzanne Sweiti

Signature:_____ Date:_____

DEDICATION

To my mother and father who have been a source of encouragement and inspiration to me throughout my life. They always supported me and encouraged me to go for the Master degree. To my dear husband Khaldon, the source of love, sympathy and emotional support. I also dedicate this thesis to my beloved brothers and sister for their persistent support. I also dedicate this work and give spatial thanks to my brother-in-law has helped me in grammatical edition of my writing.

ACKNOWLEDGEMENT

As I write the last words of this thesis, I give thanks to God for his protection and giving me the ability to do the work. I greatly appreciate the thesis's supervisor Dr. Amal Al Dweik for her sound advice, careful guidance, continual support, encouragement and time she spent with me to accomplish the thesis successfully. I also wish to thank the members of my committee, Dr. Radwan Tahboub and Dr. Amjad Rattrout for their suggestion, comments and additional guidance were invaluable to the completion of this work.

I would like thank my family for continued support, encouragement and patience from the first step till the end. Their extreme generosity will be always remembered.

To each of the above, I extend my deepest appreciation.

الملخص

الوكلاء المتنقلة عبارة عن كيانات برمجية تنفيذية قادرة على الانتقال من خادم إلى خادم آخر في الشبكات الغير المتجانسة نيابة عن مستخدمي الشبكة. أنظمة الوكيل معرضة للفشل الناتج من الاتصالات السيئة، تلف وتعطل خادم الوكيل، الهجمات الأمنية و الإزدحام في الشبكة. اذا حدثت احدى هذه الحالات أثناء التنفيذ، يتعرض الوكلاء المتنقلة للضياع أو لتلف كامل أو جزئي. لهذا السبب، وجب على نموذج تكنولوجية الوكيل المتنقل أن تتناول المصداقية لمواجهة حالات الفشل. التعامل مع الخطأ يمنع الفقدان الجزئي أو الكامل للوكيل، فهو يضمن وصول الوكيل إلى وجهته بشكل صحيح. لذا فإن تحسين الإستمرارية (البقاء على قيد الحياة) للوكلاء المتنقلة في وجود الأعطال في الشبكة الغير الموثوقة مسألة هامة لضمان التنفيذ المستمر للوكلاء المتنقلة. تم حتى الآن اقتراح العديد من مناهج التعامل مع الخطأ المعتمدة على الوكيل المتنقل. وعلى الرغم من هذه الجهود، بقي هذا المجال يعاني من عقبات في شكل تحديات مستمرة.

هذه الرسالة تعرض منهج جديد في التعامل مع الخطأ للكشف عن فشل الوكيل و استعادة الخدمات في انظمة الوكيل المتنقل. هذا المنهج الجديد تحت عنوان منهج دمج النسخ المتماثل ـ نقطة الاستئناف للتعامل مع الخطأ "IRCFT". نهجنا هو تطوير لكل من المنهج FTIRC الذي يعتمد على حفظ نتائج العملية على الخادم الرئيسي بعد إكمال المهمة على الخادمات الثلاثة الأولى اثناء الرحلة، و ايضا تطوير لمنهج PFTM ، فكرة هذا النهج هو حفظ نقطة الاستئناف وتسجيل المعلومات على كل خادم وتطبيق الوكيل الشاهد لرصد إذا كان الوكيل الفعلي على قيد الحياة أو تم

إنهاؤه. منهجنا يستخدم التعاون بين مجموعة من الوكلاء لتحقيق التعامل مع الخطأ. يعتمد IRCFT على النسخ المتماثل للوكيل، و اضافة لهذا فالمنهج يعتمد ايضا على نقطة الاستئناف، و هذا بغيه الحد من إستئناف المعالجة من طرف الوكيل في حالة فشل كل من الوكيل الفعلي (*worker agent*) وكافة النسخ المتماثلة المحدثة للوكلاء. يتعامل IRCFT مع كلٍ منْ، فشل الوكيل و كذلك فشل الخادم، فهو قادر على اكتشاف و معالجة معظم سيناريوهات الفشل في نظام الوكيل المتنقل.

بهدف تقييم أداء IRCFT قمنا بتطبيقه على نظام الوكيل Aglet وتقييمه باستخدام معاملات مثل: اجمالي وقت الجولة total round trip time و عامل استمرارية عَمَلْ الوكيل المتنقل survivability (البقاء على قيد الحياة).

لدعم عملنا، قارنا نتائجنا مع النتائج المقترحة في كل من IRCFT و PFTM فكانت نتائجنا متفوقة مقارنةً مع الأُخريات. فالعمل يظهر تحسنًا ملحوظًا في الأداء و التقنية المستخدمة واعدة في تحقيق المصداقية في نظام الوكيل المتنقل.

Abstract

Mobile agents are executing software entities that are capable of migrating from server to server in heterogeneous networks on behalf of network users. Agent systems are subject to failures that result from bad communication, breakdown of agent server, security attacks and congestion in networks. If any of such event happen, mobile agents suffer loss or damage totally or partially while execution is being carried out. For this reason, reliability must be addressed by the mobile agent technology paradigm in order to challenge such failures. Fault tolerance prevents a partial or complete loss of the agent. This ensures that the agent arrives at its destination correctly.

Improving the survivability of mobile agents in presence of failures in unreliable network is an important issue in order to guarantee continuous execution of mobile agents. Many mobile agent-based fault tolerant approaches have been proposed so far. In spite of these efforts, the field suffers from setbacks in the form of persistent challenges.

This thesis presents a novel fault tolerance approach, Integrated Replication-Checkpoint Fault Tolerance approach (IRCFT), to detect agent failures as well as to recover services in mobile agent systems. Our approach is a development of that in FTIRC approach, where the agent put its computation results on the home server after completing its task on first three servers in its itinerary and PFTM approach , the idea of this approach is saving the

checkpointing and logging information on each server and applying a witness agent to monitor if the actual agent is alive or terminated. IRCFT approach uses cooperating agents to achieve fault tolerance. It makes use of replicated agent. We have also added checkpointing, so as to limit the rollback of agent in case of failure of both the worker agent and all the updating replicas. IRCFT approach handles server and agent failure, it is capable of detecting and recovering most failure scenarios in mobile agent systems.

In order to evaluate the performance of IRCFT approach we have implemented it on the Aglet mobile agent systems and evaluated it in terms of parameters such as total round trip time and agent survivability.

To support our work, we compare our results with that proposed in FTIRC approach and PFTM approach; where our results were superior. The work shows that there is a distinguishable improvement in the performance and that it is a promising technique in achieving the reliability in mobile agent systems.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

$\boldsymbol{WA}_i$	The worker agent that is residing or traveling to S_i
$\boldsymbol{MA}_i$	The monitor agent that residing or traveling to S_i
$\boldsymbol{RcA}_i$	The replica agent that residing or traveling to S_i
$\boldsymbol{n}$	Total number of servers in the itinerary of the worker agent
$\boldsymbol{S}_i$	Server i
$\boldsymbol{log}_{arrive}$	The log message logged by the worker agent at server i when the worker agent arrives at server i
$\boldsymbol{log}_{leave}$	The log message logged by the worker agent at server i when the worker agent leaves the server i
$\boldsymbol{msg}_{ack}$	The message sending from the replicas in the previous servers to inform the active replica that are updated their status
$\boldsymbol{msg}_{alive}$	The message sending from the replica agent at server i to the monitor agent in server i to detect its status when a failure occur in the server i
$\boldsymbol{msg}_{arrive}$	The message sending from the monitor agent at server i to the monitor agent in server *i-1* when the worker agent arrives at server i

$\boldsymbol{msg}_{checkpoint}$	The message sending from the replica agent at server i to the monitor agent at server i when the worker agent sends the checkpoint to the replica agent
$\boldsymbol{msg}_{failure}$	The message sending from the monitor agent at server i to the monitor agent at server *i-1* when a failure occur in the server i
$\boldsymbol{msg\_failure}_{arrive}$	The message sending from the monitor agent at server *i-1* to the monitor agent at server i when the timeout period T_{arrive} is reached
$\boldsymbol{msg\_failure}_{leave}$	The message sending from the monitor agent at server *i-1* to the monitor agent at server i when the timeout period T_{leave} is reached
$\boldsymbol{msg}_{leave}$	The message sending from the monitor agent at server i to the monitor agent at server *i-1* when the worker agent leaves the server i
$\boldsymbol{msg}_{LogArrive}$	The message sending from the worker agent at server i to the monitor agent at server i when the worker agent arrives at the server i
$\boldsymbol{msg}_{LogLeave}$	The message sending from the worker agent at server i to the monitor agent at server i when the worker agent leaves the server i
$\boldsymbol{msg}_{newLink}$	The message sending from the monitor agent at server i to the Manager agent when the timeout of waiting for $msg_{heartbeat}$ from the monitor agent at the server *i+1* is reached without receiving the message

$\boldsymbol{msg}_{heartbeat}$	The periodic message sending from the monitor agent at server i to the monitor agent at server *i-1* to inform it that is alive
$\boldsymbol{msg}_{kill}$	The message sending from the manager agent to the monitor agents and replica agents after doing the checkpoit in the *home server*
$\boldsymbol{period}_{heartbeat}$	Period of the heartbeat message
$\boldsymbol{T}_{alive}$	Timeout of waiting for msg_{alive}
$\boldsymbol{T}_{arrive}$	Timeout of waiting for msg_{arrive}
$\boldsymbol{T}_{checkpoint}$	Timeout of waiting for $msg_{checkpoint}$
$\boldsymbol{T}_{heartbeat}$	Timeout of waiting for $msg_{heartbeat}$
$\boldsymbol{T}_{leave}$	Timeout of waiting for msg_{leave}
$\boldsymbol{T}_{LogArrival}$	Timeout of waiting for $msg_{LogArrival}$
$\boldsymbol{T}_{LogLeave}$	Timeout of waiting for $msg_{LogLeave}$

Chapter 1

Introduction

In both the academic and industrial applications, mobile agents are very important in the recent trends of distributed computing. Mobile agents are autonomous software processes able to move from one host to another in a network to access services provided there and to communicate with other mobile agents. The aim of this technological model is to shift computation towards the data rather than data to the computations [43]. Their general flexibility makes them more preferable for building distributed applications than other technological models such as peer-to-peer, client-server, etc [42].

1.1 Problem statement

Any component (hardware or software) in a network may fail (malicious or not), thus preventing the system from consistently and correctly performing with a required function under stated conditions for a specified period of time. Like any other software systems, mobile agents are not isolated from operating in unusual situations. They are also subject to fault as they consist of autonomous components in distributed dynamic environments [18]. Mobile agents might come across usual errors which emerge especially during

migration request failure, security penetration or communication exceptions [49]. They may suffer from congestion in the network, or it may be waiting and executing in a busy server. These kinds of uncertainties raise problems to the reliable agent systems design. The agent owner cannot tell whether the agent is terminated or the execution is delayed. This may lead to undesirable situations:

- The agent owner consider that the agent has been terminated, but in fact it is not. In this case, the owner launches another agent, and this cause multiple executions of the same piece of agent code.
- The agent owner waits for the agent to finish its itinerary, but the agent is actually lost due to server or agent failures.

Therefore, mobile agents have to be made reliable using through fault tolerance techniques to withstand adverse environmental situations. Fault tolerance mechanism guarantees the reliability of the system by enabling the system to resist faults and restart service despite the failure. It aims to remove the uncertainties during the execution of agents, where it should ensure that the agent can eventually reach its destination, or notify the agent owner of a potential problem.

Improving the survivability of mobile agents in the presence of agent server failures with unreliable networks is a challenging issue.

The aim of this thesis is to investigate a new fault tolerant approach for mobile agent systems. We proposed a new fault-tolerant execution model based on a combination of the updating replication agent and the checkpointing approach by using of mobile agents. The new fault tolerance approach for mobile agents systems is a development of that implemented in FTIRC approach [23] and PFTM approach [32].

1.1.1 Fault tolerance approach in mobile agents for information retrieval applications using check points

Han et al. (FTIRC) [23], have proposed a fault tolerance approach where the agent put its computation results on the home server after completing its task on first three servers in its itinerary as shown in figure 1.1. The 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, the host immediately sends the replicated copy of the original agent to the immediate check point before the faulty server.

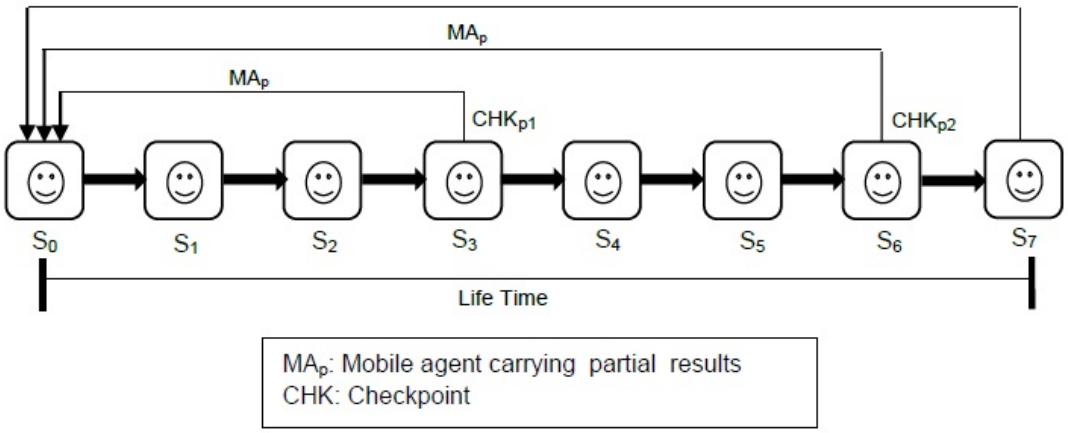


Figure 1.1: FTIRC framework [23]

1.1.2 A progressive fault tolerant mechanism in mobile agent systems

Lyu et al. (PFTM) [32], present an approach of deploying cooperating agents to detect failures as well as recover services in a mobile agent systems. The idea of this approach is saving the checkpointing and logging information on each server and applying a witness agent to monitor if the actual agent is alive or terminated. This approach can handle server failures, agent failures, and

failures in message passing. The overall design of the server architecture is shown in figure 1.2. When the actual agent fails, a new agent will be created by the prior witness agent to recover the lost actual agent by utilizing the checkpointed data stored in the server.

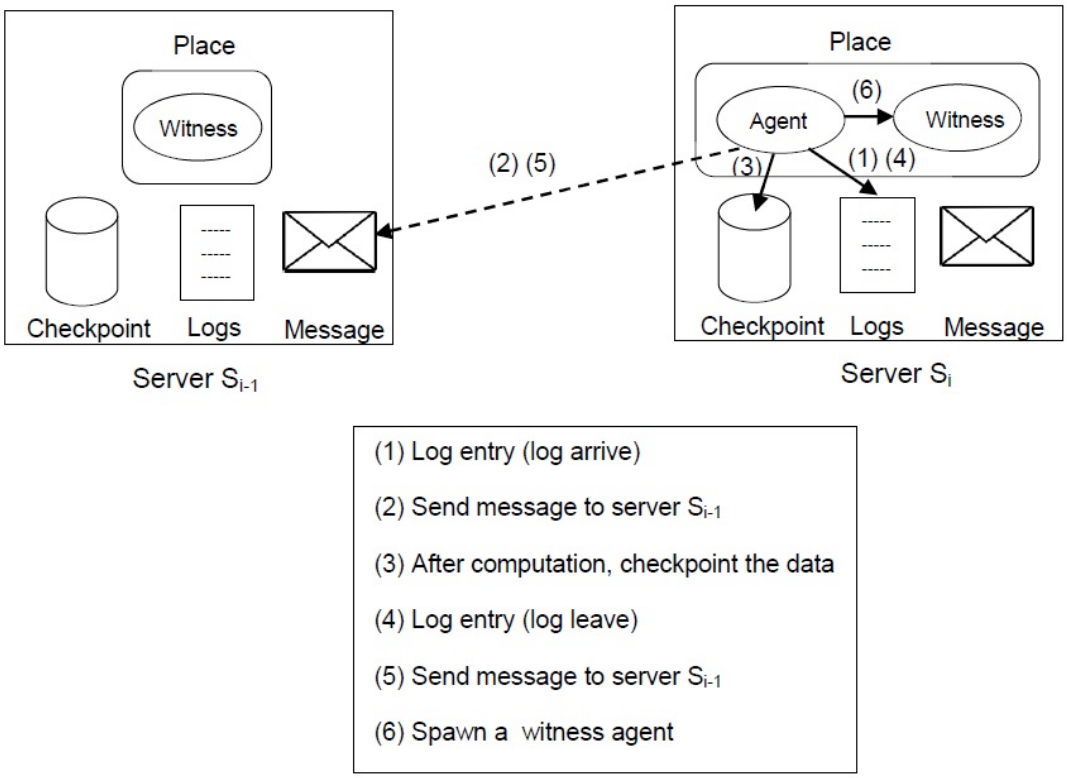


Figure 1.2: PFTM framework [32]

1.2 Contributions

The main contributions of this thesis are summarized as follows:

- Survey of the Mobile Agent Execution. A considerable research on mobile agent paradigms, features, characteristics, applications and platform are introduced in the thesis. The most interesting point is that this survey introduces both the technical and the application oriented features and specifications of the mobile agents.

- Survey of the different fault tolerance techniques for mobile agent in networks. Classification for these techniques according to certain performance criteria like: fault tolerance technique, fault type, coordination, performance analysis, discovery mechanisms, fault tolerance scheme and exactly-once.

- Proposing a fault-tolerant approach for mobile agent systems, by using cooperative agents. This approach is named as Integrated Replication-Checkpoint Fault Tolerance Approach (IRCFT) which is a development of that implemented in FTIRC approach [23] and PFTM approach [32]. Unlike PFTM approach [32] which saves the checkpointing on each server, IRCFT approach uses the replica agent. It updates all the replicas after each computation, it uses also the checkpoint on the home server after completing its task on first four servers in its itinerary but FTIRC approach [23] uses the checkpoint after three servers. IRCFT minimizes the monitor agents in itinerary by terminating them after the checkpoint in the home server; because existence of all of monitor agent is not necessary on the initial servers [32]. IRCFT approach ensures non-blocking agent and the exactly-once execution property.

- Developing the reliability evaluation experiments of the algorithm proposed for IRCFT approach by implementing it using Aglet mobile agent system.

- Compare the results of IRCFT approach with the two other approaches.

1.3 Thesis Organization

This thesis is organized in the following way:

Chapter 1 is an introduction of the thesis. It gives a brief description of FTIRC approach [23] and PFTM approach [32]. It gives the motivation for this thesis, the research problems. We then summarized the contributions and the organization of this thesis.

Chapter 2 gives background on mobile agents. It discusses the various mobile agents platforms, applications and features. It provides an analysis of building a fault-tolerant mechanism for mobile agent systems. It also discusses Aglet platform for mobile agents that will be used in this work.

Chapter 3 provides a summary of some previous related works.

Chapter 4 is the core of the IRCFT approach. It specifies the assumptions for the suggested execution paradigm. It focuses on the details of the agent failure detection and recovery mechanism. It describes the mobile agent system architecture that supports the proposed mechanism, and outlines the protocol of the mechanism. A detailed discussion on different failure scenarios is described.

Chapter 5 presented the evaluation of the IRCFT approach by stating the experiments, graphs and analysis of the results.

Chapter 6 concluded the most important results and achievements for the proposed system. Finally, we presented some directions of future research.

Chapter 2

Background

In this chapter, the theoretical background needed to understand the remaining parts of the thesis is presented. This chapter is organized as follows. In the first section, the important issues regarding the mobile agents were discussed, such as: the mobile agent definition, characteristics, advantages, applications and platforms. In the second section, we describe the fault tolerance of mobile agents, study the different failure types that occur in a mobile agent system, specify the techniques and the basic properties of fault tolerance and categorize the fault tolerance of mobile agent. The final section covers Aglet platform for mobile agents that will be used as the implementation environment for testing IRCFT approach.

2.1 Mobile agents

Mobile agents came into use about two decades ago. It is a new way for distributed computations structuring [26]. One of the major problems that caused the introduction of mobile agents was the problem of performance with RPC [8] through weak communication links. This specific problem was augmented as a result of the emergence of ad-hoc communication infrastruc-

tures in industrial fields. In such cases, in addition to the increasing internet communication, rendered communication-intensive distributed computations with RPC difficult to cope with. The traditional difference between mobile agent approaches and RPC communication is manifested in figure 2.1 .

The client in figure 2.1 (b) communicates messages with a server which is a part of distributed computation. A request message is sent by the client and a reply message is sent by the server. There are two major problems regarding performance in this approach. The first one, when the network communication between the client and the server is characterized by long message latency, the interaction between the server and the client probably becomes important to the complete execution time of the computation. Second, if message bandwidth in network communications between the server and the client is low, a reply with a large-size message may cause lengthening the execution time of the computation.

In figure 2.1 (a), the mobile agent is issued from the client to the server. The mobile agent executes the parts of the client which need an interaction with the server. All the interactions that follow between the client and the server are done by local procedure calls. Hence, not dependent on the performance of network latency or the network bandwidth at hand.

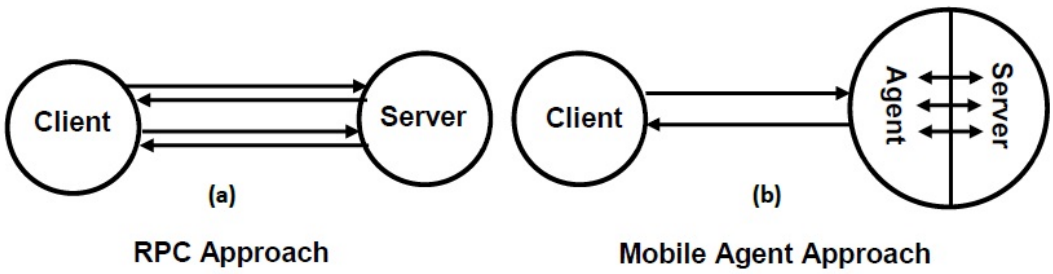


Figure 2.1: Client-Server vs. Mobile Agent Paradigm

The program executed by mobile agents consists of code, data and execution state. These are emerged into particular intelligence and the capability to migrate in an autonomous way across the network that stand for users in different tasks and satisfying the request of its owners [42], [44].

A mobile agent has to be able to communicate with other agents and host systems while the process of migration and execution are underway on several heterogeneous machines or hosts connected to the network. As a result, the platforms usually provide a reliable protocol for communication. Mobile agents are being created in one environment of execution, its code and state can be transferred to another execution environment in the network where execution resumes. In addition, they have the potential to sense their environment and do a series of activities to achieve the goal of the mission.

The concept of mobile agents was initiated to collect momentum in the areas of research and industry. It started with Telescript, which is an agent platform where migrating active objects were initiated as suitable model for maintaining applications on public network [51]. Following the implementation of Telescript, different projects at the institutes of research and in industry field began to emerge. There are popular platforms that include Aglets, Mole, Tacoma, Voyager, Grasshopper, James etc [42], [43], [51].

2.1.1 Mobile agents characteristics

Mobile agents show particular properties that permits efficient deployment of services and applications in a manner that is more flexible, dynamic and customizable in distributed systems. Because of these characteristics, they are more popular for building mobile applications than the classical client-server paradigm, etc [42]. These characteristics include:

- Mobile agents naturally operate in heterogeneous environments [44]. They usually operate in wide-areas and diverse networks where either the reliability or the security assumptions of the computers and the network connections respectively are not a factor.

- Mobile agents act autonomously on behalf of the user [38]. They have the potential to precisely commence their migration on its own and execute its owner's request in contrast with other traditional software objects which are centrally controlled by the basic operating system or middleware. They independently migrate from one host and can efficiently make a decision of where to migrate with the aim of resuming execution [37].

- Mobile agents reveal a multi-hop ability [37]. This means they have the ability to migrate with their code, data, and execution state more than one time by restarting the execution in another server in the network in the wake of finishing their tasks in the server which was visited first.

- Mobile agents greatly depend on the basic communications protocol through interactions and message changes in order to successfully perform a certain task in the agent system [44]. Mobile agents can communicate synchronously or asynchronously during migration with other agents or the host systems. The others are by the way of local interprocess communication mechanism and network-based mechanism.

2.1.2 Advantages of mobile agents over traditional distributed techniques

In this section, some of the merits given by the mobile agents which make them preferable for the creation of distributed systems and especially mobile

agent applications than other models. These include:

- Mobile agents conserve bandwidth especially in applications where there is a great amount of data that have to be processed on diverse hosts by moving the code to the data instead of moving data to the code [13]. By doing this, traffic over the network is reduced greatly, so the bandwidth of the network is saved. Besides, this also aims at increasing the locality of execution. Therefore, data can be executed in its locality.

- Mobile agents have the ability to show asynchronous and autonomous interaction as they can be transferred to carry out a particular task with the intelligence merged on them and they can decide by itself how to interact with at the time of execution [38].

- Mobile agents provide extensive flexibility in date operations which are disconnected particularly in environments with irregular connection like the Internet [22]. In contrast with client-sever application, it has the ability to execute the requested operation on a particular host and wait for the resumption of connection before they move on to the next host. This may decrease the connection costs especially when it comes to wireless networks.

- Mobile agents have the ability to improve latency of network with better response time compared with client-server applications because the intermediate outcome is not sent back and forth between server and client. This might have resulted in delay that is caused by congestion of network [22]. In this case, only the final result of the computation will be transmitted to the server.

- Mobile agents are strong and tolerant with faults since thy are able to

dynamically reply to bad conditions and events in a distributed system [30].

- Mobile agents offer support for heterogeneous environments by the way of mobile frameworks which separate the operating system and the executing host. That is to say, mobile agents are usually independent of the transport-layer and computer-layer and they only rely on the execution environment [31].

- Mobile agents have better scalability especially with respect to their flexible in dynamic deployment [22]. As regards client-server development, modifying the client may require redeployment of client installation. In the systems of mobile agents, the agent modified can be injected once and wherever it is required to install, it installs itself.

2.1.3 Mobile agent applications

Mobile agents have been approached greatly in many application fields such as e-commerce and m-commerce [38], [19], network management and monitoring [11], distributed information retrieval [19], telecommunications [38], [19], remote device control and configuration [29], Internet etc [42], [27].

2.1.4 Agent platforms

The agent platforms supply executing environment for mobile agents. Several academic and commercial systems present agent-based computing platforms out of which to mention some are the Agent TCL, ARA, Mole, Tacoma, Voyager, Aglets, Concordia, Voyager, NOMAD, JAMES, JADE, Naptel, MadKit, FIPA-OS, Grasshopper etc. [42], [27], [20]. These platforms are different from their features, architecture and implementations. However, they more

or less offer common facilities for the support of agent mobility, inter-agent conversation, programming languages and various forms of security etc. [13]. Most of the agent platforms nowadays are using interpreted languages like java; others are based on compiled languages such as C.

Table 2.1 presents a qualitative comparison of the current agent platforms that are used in recognized mobile agents fault tolerance implementations. It is performed on the basis of following parameters:

Nature of product

Whether a platform is open source or commercially available influences end users. If the platform is freely available some compromises can be made with features provided as compared to the one which comprise some cost.

Programming language

Programming language is the implementation language adopted by the mobile agent system for application developers to program a mobile agent.

Mobility type

Mobile agents could be implemented with either or both of strong and weak mobility schemes [39]. In general, mobile agents can be transmitted with or without their execution state [25]. In strong mobility, the full process state (execution state, code and data) along with the execution thread is captured, moved and restored at the destination host. Whereas in the case of weak mobility, the code is migrated and linked to a new execution thread at the destination host [10]. Strong mobility offers better flexibility, but most agent platforms support only weak mobility due to its complexity and in the inadequate support to deal with execution context migration by the present

Java Virtual Machine (JVM) based platforms [10], [50].

Reliable communication

Reliable communication is an essential condition to effective fault detection and recovery especially among agents or collaborating agent. With reliable communication, it is possible to route messages to its mailbox during migration either using synchronous or asynchronous mechanism. Currently all agents platforms support some form of communication, but the platforms that support asynchronous communication provide better communication performance over synchronous communication [34].

Table 2.1: Qualitative comparison of mobile agents platforms

Platform	**Nature of product**	**Programming language**	**Mobility type**	**Reliable Communication**
Aglet [1]	Free, Open source	Java	Weak	Asynchronous, Synchronous, Proxy
Concordia [57]	Not free	Java	Weak	Asynchronous
FIPA-OS [2]	Free, Open source	Java	Weak	Asynchronous
Grasshopper [15]	Not anymore (aban-doned)	Java	Weak	Synchronous, Asynchronous, Multicast, Dynamic method invocation
JADE [3]	Free, open source	Java	Weak	Asynchronous
JAMES [51]	Not free	Java	Weak	Asynchronous
MadKit [4], [16]	Free	Java	Weak	Asynchronous
MOLE [9]	Not free	Java	Weak	Asynchronous, Synchronous, Sessions
Naplet [5]	Free, open source	Java	Weak	Asynchronous
Tacoma [28]	Not free	C/C++, ML, Perl, python	Weak	Asynchronous, Synchronous
Voyager [6]	Not free	Java, C#, C++	Weak	Asynchronous, Synchronous, Multicast, Proxy
Anchor [7], [52]	Available in BSD license	Java	Weak	Asynchronous
SPRINGS [24]	Free	Java	Weak	Asynchronous, Synchronous,Proxy
Tracy [12]	Free	Java	Weak	Asynchronous

2.2 Fault tolerance of mobile agents

In the light of the increased demand for better and more dependable system performance, both software and hardware components face the threat of faults which undermine the system reliability. The faults change the normal execution state into error state. This in turn causes system failure. To keep the systems from these threats, mechanisms of reliability have been designed with different kinds of techniques to tolerate these faults while the system continues to provide a satisfactory level of service.

Fault tolerance makes the system more reliable. It makes the system able to bear the faults and continue working despite failure. The goal is to allow the system to work flexibly in spite of the faults which continue to exist in the system after development [42], [44]. The effective design of the mechanisms of fault tolerance must balance between portability, transparency, reliability, scalability and low overhead. In many application fields, fault tolerance has been used. For example, electronic banking, nuclear safety, military and nautical systems, commerce, medical services, aircraft and air traffic control, space mission, automated manufacturing and so on [42].

2.2.1 Mobile agents fault tolerance techniques

The two general approaches which give mechanisms for fault tolerance in mobile agent systems are agent-centric and system-centric [39].

Agent-centric approach

In agent-centric approach, the strategy of fault tolerance is built inside the mobile agents. In this case the developer has to do a big task, keeping consistency of the data and the processes. [21]

System-centric approach

Furthermore, in system-centric approach the fault tolerance strategy is integrated into the agent platform. In this case, there may be a need to modify the agent platform in order to have capabilities for fault tolerance [22].

2.2.2 Mobile agents fault types

When fault occurs in a mobile agents systems (MAS), interactions between agents may cause the fault to spread through the system randomly. Mobile agent can suffer from different breakpoints during its execution in the system. These failures should not prevent the mobile agent from continuing its itinerary to achieve its goals. There are various types of failure that may affect the mobile agent and surrounding environment [58]. Irrespective of being permanent, intermittent or transient, the failures that affect the execution of mobile agents are categorized into agent failure, communication and crash failure [14]. For instance, some implementations use the term node failure while others state it as host failure, where both refer to crash failure. Figure 2.2 shows the classes of failures.

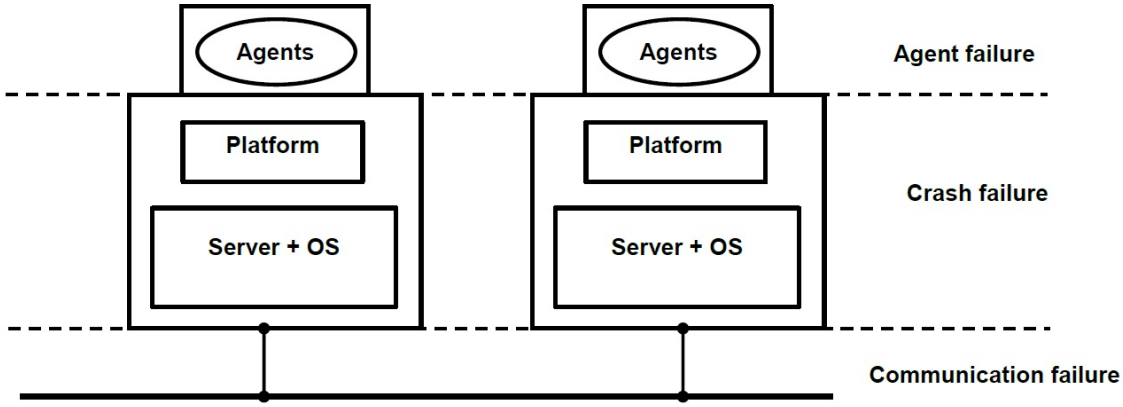


Figure 2.2: Classes of failures

Communication failure

It is a kind of failure that can affect the work of the agent in terms of mobility and communication. Link failure directly affects the agent's transfer and messages; it causes duplication or damage to the message [58], [17].

Crash failure

It is a kind of failure that occurs when the mobile agent hosting place fails such as server failure. This results from hardware failure or operating system failure or agent platform failure, so this stops communication between the agent, other agents and agent home server [14]. This kind of failure leads to agent system failure either partially for the agent's locations because of the agent locations equipment components breakdown, for instance, damage to the communication units or total server crash as a result of failure in the system software or an attack on security [14].

Agent/Agent software failure

It is a kind of failure that causes cessation of agent execution because of unexpected circumstances, a fault in the mobile agent or a software fault. The software faults are primarily because of wrong computation, incorrect inputs, exception and prevention of access to service as a result of heavy load and situation such as deadlock or live-lock characterized by locking [14]. Agent failure occurs either during transmission or while execution is in progress.

2.2.3 Fault tolerance requirements

Two basic properties a mobile agents fault tolerance execution must adhere to in order to be reliable and achieve their design goals. These are the

non-blocking and exactly-once execution properties [22], [44]. The following subsections discus these two requirements in more details.

Non-blocking

Sequel to agent execution, the characteristic of non-blocking makes a stipulation that moving away from any infrastructural component (e.g. place, communication link, machine, etc) from its particular services ought not to hamper the execution of the agent. For example, if a failure in one component may ruin the agent's execution when execution on a particular place, this may impede the progress of the execution and during that, execution can't go on and becomes blocked. The property of non-blocking execution imposes the inclusion of fault tolerance procedures to ensure the continuation of execution despite the failure resulting from loss of agents [44]. Progress in execution happens only when failed part recovers with the help of the existing recovery mechanisms.

Exactly-once

The exactly-once property stipulates that manifold executions of the agent are prohibited. That is to say, the agent code has to be subject only to one execution that exactly once [22]. The vital importance of this property is considered as resulting from particular operations bristled with adverse effects especially in applications which are transaction-based. For instance, banking systems where agent is used for updating accounts.

2.2.4 Fault tolerance schemes

The fault tolerance schemes that deal with failures and recovery of system were classified into three categories: checkpoint-based and replication-

based and a hybrid of both schemes sometimes exist [22], [44], [58]. Each scheme has its own strengths and weaknesses that are applied for its suitability/unsuitability in a given application area.

Replication schemes

Replication-based schemes present one of the most widely used techniques to attain fault tolerance in distributed systems. It can grantee an agents service reliability, survivability and increased availability. In Replication scheme, groups of agents specified as replicas of leader agent exist in multiple places. The replica elects to work as the leader agent to continue computation in the case of the leaders failure [44]. The replication scheme is used to ensure the non-blocking property, but have undesirable effects of compromising the exactly-once property, which is deemed unfavourable for transaction based applications [22], [38]. The Distributed transaction, the consensus protocol or the leader election protocol can be employed to maintain the exactly-once execution property [22]. Replication approach is expensive in terms of performances cost because of higher cost involved in preserving replicas characterized by consumption of enormous execution time such as the time required to replicate or migrate or to implement the consensus between replicas [37] as shown in figure 2.3 [41].

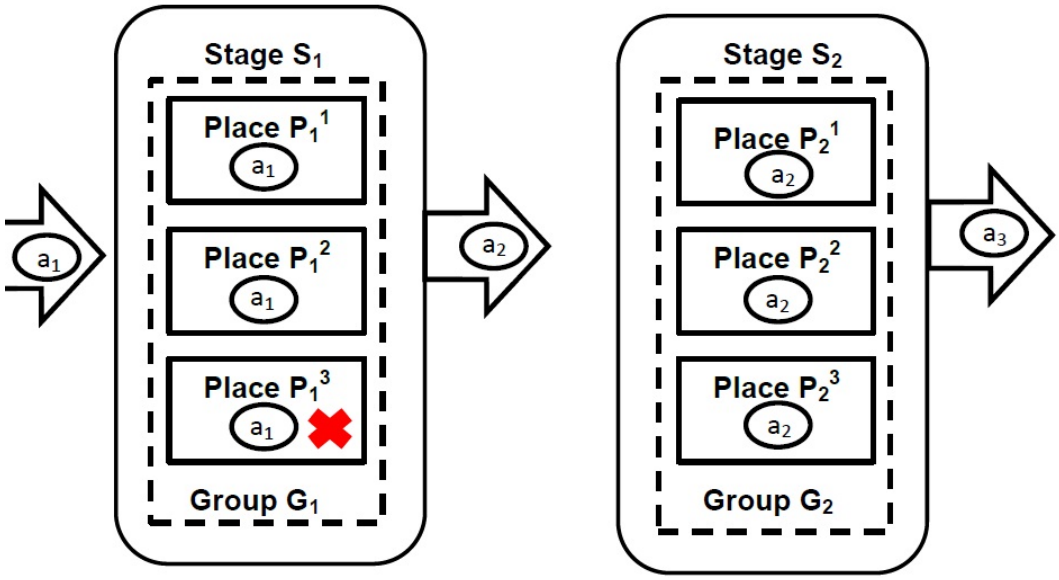


Figure 2.3: Agent replication

Checkpointing schemes

Checkpointing is used to preserve mobile agents availability in case of agent server crashes. It combines two strategies: checkpoint and rollback recovery. Checkpointing approaches focus on saving an agents intermediate state to a non-volatile memory at certain time intervals during execution. The saved state is called checkpoint and retrieving the previous checkpoint upon recovery is called rollback recovery [36]. Instead of restarting from the beginning, the checkpointed information assists the agent to resume computation from saved state. Comparing it to the replication schemes, checkpointing scheme have lower computation overhead. Checkpointing may block an agent execution after a crash failure, because the checkpoint may not be ready until the host recovers [53]. Moreover, the checkpoint scheme bears the possibility of violating the non-blocking property of mobile agents execution [40], [38] as shown in figure 2.4.

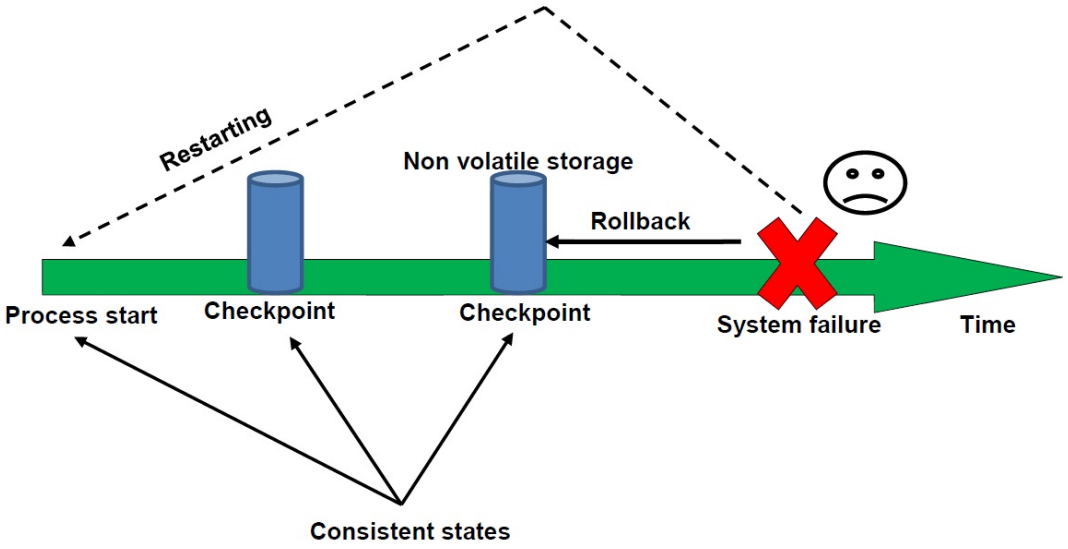


Figure 2.4: Checkpointing

2.3 Aglet

Aglets [1] is one of the best known and most widespread mobile agent systems today. The word aglet means lightweight agent in much the same way that applet means lightweight application. The term aglet is a combined work of agent and appletAglet is developed by research team lead by Danny B. Lange at the IBM Tokyo Research Laboratory in Japan in early 1995. It is an open source freely available platform, with latest version Aglet 2.5 alpha. Aglets are Java objects that can move from one host on the internet to another. An aglet can execute on one host, can stop its execution, dispatches itself to other host and resume its execution on new remote host. On moving from one system to another aglet carries its code and data with it, but not thread state while migrating from one host to other (weak migration). The Aglets Software Development Kit (ASDK) is an implementation of an Aglet API. The ASDK involves Aglet API packages, documentation, sample aglets, and the Tahiti Server. Tahiti allows the users to receive, manage, and send aglets to other host that are running Tahiti.

2.3.1 Basic elements

Here we will see the basic elements of aglet system briefly. The aglet object model defines some abstraction and the behavior which is used to take full advantage of this agent technology. The abstractions which are used are: aglet, proxy, context, message, and identifier.

Aglet

An aglet is a mobile Java object that visits aglet-enabled hosts in a computer network. An aglet that executes on one host can halt execution, dispatch to a remote host, and resume execution there. It is autonomous because it runs in its own thread of execution after arriving at a host, and reactive, because it can respond to incoming messages.

Proxy

A proxy is a representative of an aglet. It provides a handle that is used to access the aglet. The proxy serves to protect the aglet from direct access to its public methods. It also provides location transparency for the aglet. This mean that an aglet and it proxies can be separated so that a local proxy hides the address of the aglet.

Context

A context is an aglets workplace. It provides the execution environment at the remote site. The context is a stationary object that provides a means for maintaining and managing active aglets in a uniform execution environment where the host system is secured against malicious aglets .ie. the Tahiti server provides context to aglets created in it.

Message

A message is an object exchanged between aglets. It communicate by message passing and not by method invocation. Messaging between aglets involves sending, receiving, and handling messages synchronously and asynchronously.

Identifier

An identifier is bound to each aglet. The identifier is unique, immutable over the life time of the aglet and independent of the context it is executing in.

2.3.2 Aglet's life cycle

Basic operations that govern the life cycle of aglets are : creation, cloning, dispatching, retraction, activation/deactivation and disposal. Figure 2.5 illustrates these events in the life cycle of an aglet. The following list summarizes the fundamental operation of an aglet:

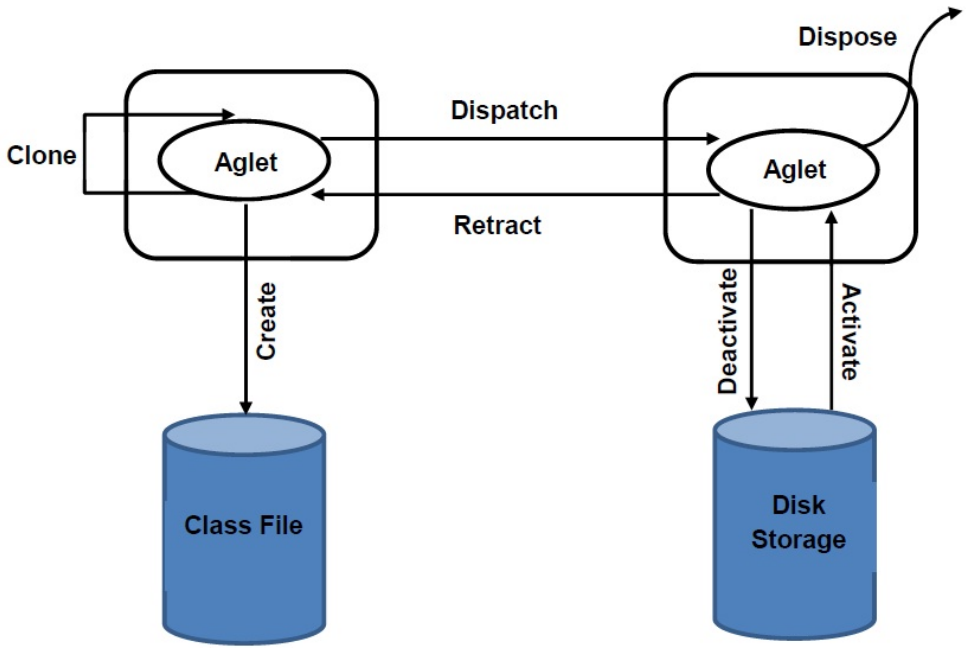


Figure 2.5: Life-cycle of Aglets

Creation

An aglet is created within a context. The new aglet is assigned an identifier, inserted into the context, and initialized.

Cloning

The cloning of an aglet produces an almost similar copy of the original aglet in the same context. The cloned aglet has different identifier.

Dispatching

Dispatching an aglet from one context to another will remove it from its current context and forwarded it into the destination context, where it will restart execution (execution threads do not migrate). The dispatching pushed the aglet into its new context.

Retraction

The Aglet identifier must be known to be able to perform a retraction. The retraction of an aglet will remove it from its current context and reintroduced to its home context.

Activation and deactivation

The deactivation of an aglet is the ability to temporarily halt its execution from its current context and store its state in secondary storage. The aglet is not removed from the aglet context. It sleeps for at least the specified dormancy period. Whereas, the activation of an aglet will restore it in a context. When a deactivated aglets sleep cycle expires. It is activated and its execution begins.

Disposal

The disposal of an aglet will terminate its current execution and remove it from its current context.

2.3.3 Interaction between aglet and host

Through an AgletContext object, an aglet can interact with its environment (its aglet host). An aglet can obtain a handle to its context by invoking getAgletContext(), a method it inherits from base class Aglet.

2.3.4 Interaction between aglets

Interaction with an aglet takes place only via its proxy. To interact aglet with each other, they do not normally invoke each other's methods directly, they go through AgletProxy objects, which serve as aglet representatives. The context might use several of the proxys methods to control the aglet such as clone(), dispatch(), dispose(), and deactivate(). The method sendMessage() is used to send asynchronous messages to the aglet via its proxy.

Chapter 3

Survey and Literature Review

This chapter introduces a summary of some important contributions related to our work and includes a performance evaluation of these techniques by considering some parameters such as fault type, discovery mechanism, fault tolerance scheme, exactly-once, etc.

3.1 Background review

Reviewing previous literature shows that a number of research work has been done in the fault tolerance of mobile agent system. In this section, some existing fault tolerance approaches are briefly discussed.

In [46], the authors have proposed an approach for providing efficient fault tolerance in mobile agent systems to overcome certain failures. Here, antecedence graphs and message logs are used to maintain fault tolerance information of mobile agents. The parallel checkpointing approach is used which considers the antecedence graphs. Before an interval of state, the information of the events already occurred is stored in the form of antecedence graphs. These graphs are directed acyclic graphs that record the dependency information. Whenever any unexpected situation or failure occurs,

these graphs are regenerated for recovery. This approach improves the message overhead, recovery time and execution time as well. The previous antecedence graphs are deleted after the final checkpointing has been done which in turn reduces the graph size. The advantage of this technique is that the identifying, execution and recovery time is improved and the message overhead can be reduced. The disadvantage of this technique is that there is an overhead of taking message log and antecedence graph of each message.

In [23], agent failures is recovered by checkpoint. The agent put its computation results on the home server after completing its task on first three servers in its itinerary. The approach makes use of checkpointing, partial results and the address of last host visited is saved prior before the agent visits the next host in the itinerary. If agent stops its execution due to any fault on the server and unable to move in its itinerary, a message is send to home server, which then sends the replicated copy of the original agent to the immediate checkpoint before the faulty server. The authors measure the performance of the approach. The analysis of this technique show a good result by improving the round trip time of the agent, since after occurrence of fault, the replicated agent need not rollback to the first server as it starts moving from the checkpoint immediately before the faulty server. This approach may violate the exactly once property.

The authors of [45] have proposed an approach to sort out the agent crash problem. Here, checkpoint approach and a clone of original agent are used. The clone is used in an itinerary to keep track of the actual agent. So, if a failure occurs in the mobile agent system, the clone is used to recovery the actual agent. Moreover, if the clone also fails then checkpoint approach can be used. The performance improves the total trip time. However, it may sometimes to violate the exactly once property of mobile agents.

The designing of adaptive mobile agents in [33] aims at accepting additional roles when working inside a special environment that is called context-aware environment that carries out the job of sharing and assigning the roles of the mobile agents existing in the environment. It provides the rules based on the conditions. Mobile agents obtain roles based on the instructions that the environment provide. The adaptive Mobile Agents have to cooperate with each other as well as with the environment to acquire roles. The objective of these roles is to give access to a resource. This formula of limiting access or granting it to a resource is known as Role Based Access Control (RBAC) that performs the major role in data security management. The integration of the various components is done through communication messages. The merit of the technique is that the mobile agents are already inside the system. So, there is no need for any kind of external communication. This means that the time to make and send a new mobile agent is saved and time for response becomes shorter.

In [48], authors introduce that the server and agent failures are detected and recovered by the cooperation of agents with each other. In order to detect and recover the failed agent in 2-Dimensional Mesh Network, another types of agent are used, namely the witness agent, to monitor whether the actual agent is alive or dead. It prevents a partial or complete loss of mobile agent. A communication between both types of agents is done by sending direct and indirect messages. In this approach the use of 2-D mesh network, dependencies among witness agent get reduced as compare to linear network and the drawback is 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.

In [35], the idea of agent tracking technique is described. It provides an

efficient system so that excellent management and maintenance can be performed in the networks of the systems of mobile agents. The life cycle of the agent is controlled by the so-called Multi Agent System (MAS). The authors have also described the Extensible Java-based Agent Framework (XJAF). Certain challenges for the tree-like architecture will be made for this framework. As a result, the work was intended to lessen particular challenges and to make fault tolerance do better. If a connected graph is completely followed, a reliable network can be built. This network will comprise registration of MAS and detection if the MAS breaks down. Furthermore, the agent tracking system is laid. This technique is claimed to be beneficial in Maintenance and efficiency which are attained through robust tracking techniques. The drawback of this approach is that it doesn't comprise replication and techniques for timely fault detection. These properties promote the robustness and flexibility in the system.

In [59], the fault tolerant mechanism has been laid with relation to the applications that deal with transactions. The protocol proposed is based on the behavior of Mobile Agent, Watch Agent as well as Transaction Manager. When commit at destination protocol is applied, the property of exactly-once execution is not violated. The advantage is that fault tolerance technique in transactional support is available. The fault tolerance's shortcoming is that Transaction Manager is not considered. Therefore, a non-blocking atomic commitment protocol might be included in the Transaction Manager to make global transaction valid. This is done by mobile agent.

In [47], an integrated mechanism has been proposed using SMAPS (Secure Mobile Agent Platform System) which aims to prevent agent blocking in certain cases where agent is identified by malicious host. When executing transactions, the partial result retrieval is taken into account. This is then

used to follow the location of the mobile agents during the process at any time. The criteria considered are : fault time, message size and communication overhead. The proposed mechanism has been applied to SMAPS. As for the originator, the process of tracking can be achieved through acknowledgments. Some partial results are sent back after the passage of some fault tolerant time and the number of hosts defined beforehand are visited. Thus attacks are prevented and so protection and reliability are provided. As a result, the major objective is to make the mobile agents appropriate for real time applications because it reduces the possibility of total agent loss. This technique is helpful in improving the reliability and performance through parameters like communication overhead, size of the message and fault tolerance time. This technique's shortcoming is that is suitable merely for time sensitive applications.

In [32], the authors address a fault tolerance approach of deploying cooperating agents to detect server and agent failure as well as to recover services in mobile agent systems using unreliable network. The idea of this approach is saving the checkpointing and logging information on each host and applying a witness agent to monitor if the worker is alive or terminated. This model employs three types of agents. They are actual agent which performs programs for its owner, witness agent which monitors the actual agent and the witness agent after itself, probe which is sent for recovery the actual agent or the witness agent on the side of the witness agent. A communication between both types of agents is done by sending direct and indirect messages. The agent server provide three types of stable storage for logs, checkpoints and messages. At each time, after the worker leaves a host, the witness will spawn a new witness agent on it and migrate to the server where the actual agent has just visited. Therefore, a chain of witness agents will

be created along the itinerary. The older witness agents are responsible for monitoring the witness agent that is just one server closer to the actual agent in its itinerary.

$w_{i-2} \rightarrow w_{i-1} \rightarrow w_i \rightarrow \alpha$ represent the monitoring relationship among witness agents, where w is witness agent and α represent the worker. When a worker fails, a new worker will be created by the prior witness agent to carry out the task. When w_i fails, w_{i-1} is responsible to recreate it. Thus, the chain of witness agents can be maintained. The authors claim that because no more than k servers can fail simultaneously, k witness agents are sufficient to guarantee the availability of w where k is the number of witness agents. The system is susceptible to blocking. If the host carrying the last witness agent is crashed and it cannot be recreated, then k becomes zero immediately. At this moment, the worker is not protected by any witness agent. The improvement in agent survivability is achieved by spending more time and space resource's, more witness agents and probes are necessary, so system complexity increases.

3.2 Comparative analysis of fault tolerance approaches

A comparative analysis of techniques as shown in table 3.1 for nine (9) researches is performed on the basis of some parameters like agent centric, system centric, fault types, coordination, performance analysis, discovery mechanism, fault tolerance scheme, type of recovery and exactly-once.

Fault tolerance techniques

Fault tolerance parameter show what technique is used. These are marked by agent-centric or system-centric.

- **Agent-centric:** The fault tolerance is built inside the mobile agents.
- **System-centric:** The fault tolerance is achieved using the mobile agent.

Fault types

Fault type parameter describes that which type of fault is being considered. The failure may be agent failure, crash failure or communication failure.

Coordination

The coordination parameter show the mode of coordination i.e. how the communication is to be carried out.

Performance analysis

Performance analysis parameter describes whether the performance analysis has been considered in the work or not.

Discovery mechanism

Discovery parameter show that what mechanism is used to detect the agent failure.

Fault tolerance scheme

The fault tolerance parameter describes that which scheme of fault tolerance is being considered. The fault tolerance projects classified into three

categories: checkpoint-based, replication-based and hybrid schemes.

Exactly-once

Exactly-once execution parameter describes whether the exactly-once property has been achieved in the work or not. These is marked by yes or no.

Table 3.1: Comparative analysis of fault tolerance approaches in mobile agents

Paper	**[46]**	**[23]**	**[45]**	**[33]**	**[48]**	**[35]**	**[59]**	**[47]**	**[32]**
Fault tolerance techniques	Agent-centric	Agent-centric	Agent-centric	System-centric	System-centric	Agent-centric	Agent-centric	System-centric	Agent-centric
Fault type	Agent	Agent	Agent	Agent	Agent, Crash	Agent	Agent, Crash, Communication	Agent	Agent, Crash, Communication
Coordination	Direct	Indirect	Indirect	Direct	Direct, Indirect	Indirect	Direct	Direct	Direct, Indirect
Performance analysis	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes
Discovery mechanism	parallel check-point	Timeout	Clone and Timeout	Heartbeat signal	Witness agent	Connection agent	Witness agent	Acknowledge	Witness agent
Fault tolerance scheme	Checkpoint	Checkpoint	Checkpoint	Replication	Checkpoint	Checkpoint	Checkpoint	Checkpoint	Checkpoint
Exactly-once	Yes	No	No	Yes	No	No	No	No	Yes

Chapter 4

IRCFT Approach Design and Implementation

In the first section we introduce the IRCFT assumptions for the suggested execution approach of the IRCFT. We introduce the approach architecture in the second section. In section 3, we describe the approach which involves the cooperation between different kinds of agents. Agent interaction protocols of IRCFT approach are describe in section 4. Different failures and recovery scenarios are discussed in section 5. The characteristics of the presented approach are discussed in section 6. Finally, a comparison between IRCFT approach, FTIRC [23] and PFTM [32] is done.

Some publication was extracted from the thesis: [56], [54] and finally [55]. The IRCFT approach is a development of that implemented in FTIRC [23] and PFTM [32]. We address a fault tolerance approach of deploying cooperating agents. The basic idea used in IRCFT approach is to tolerate faults using the concept of checkpoint and replica agent.

4.1 Approach assumptions

The agents in IRCFT approach communicate at different locations by exchanging messages through unreliable communication channels. Therefore, the system is assumed to use unreliable network connection. i.e., there is treatment of the loss in messages due to communication failures. Some other specific assumptions in the system may be summarized in the following points:

1. Agents in the system can be generated from every server on network. Thus, each server on network provides the mobile agents an execution environment to accomplish its tasks and objectives.
2. The home server is always available and free of failure.
3. All agent servers are correct and trustworthy.
4. No failure in log entries and the mobile agent can be recorded in the permanent storage.
5. Software faults in mobile agents or in mobile agent platforms may occur.
6. No Byzantine failure.
7. When a server failure occurs in S_i, all the agents inside S_i will be terminated.

4.2 Approach architecture

In our approach, we distinguish four types of agents, one type is performing the required computation for the user,named as the *worker agent*. Worker agent plays an active role in this approach. Another type is to detect the

status of the worker agent, named as the *monitor agent*; to monitor whether the worker agent is alive or dead. It is also responsible to monitor the status of other monitors. In our approach we use two monitors to detect the status of the worker agent: the first located in the same server where the worker agent resides and the second located in the last server. The third agent is the *manager agent* Which resides in the home server and is responsible to send the address of the next server to the worker agent. The Last agent's type is the *replica* agent to recover the worker agent in case of failure.

Monitor agent and worker agent communicate by using direct and indirect messages. The direct message is a peer-to-peer message between the monitor agent resides in the server that the worker agent just previously visited and the monitor agent that resides in the same location with the worker agent. There are cases that the worker agent cannot send a direct message to a monitor agent who is in the same location with it. There can be several reasons, e.g., the monitor agent is on the way to the target server.

Monitor agent sets a timer with a certain time-out value for each server S_i. We also need to log the actions performed by the worker agent. The purpose of log entries and message is to guarantee that the actual agent has finished up to a certain point of its execution. The information logged by the agent is vital for failure detection. If a server failure occurs between a log entry and its corresponding message, it can determine when and where the worker agent failed.

When a failure happens, we have to recover the lost agent due to the failure. If we allow the agent to start computation from the beginning, the exactly-once property will be violated. Therefore, We use the replica or the checkpoint to recover the worker agent. The checkpoint serve to save partial results after completing the execution on the first four servers of the itinerary,

where it is used to recover the worker agent in case that the replicas and the worker agent are down.

4.3 Approach design

During an agents trip, the agent will visit a number of servers. The sequence of servers visited composes the agents itinerary. The servers on the itinerary are denoted by S_0,S_1,S_2,...,S_n, where S_i is the i^{th} visited server. Specially, S_0 is the home server on which the agent is generated.

At first, the manager agent creates in the home server the *worker agent* and the *monitor agent* where they migrate to the next server.

Assume that, currently there are n network servers on the execution itinerary, and the worker agent has just arrived at server S_{i+1} and does not reach the fourth server yet, the worker agent and monitor agent are at the third server and the elder monitor agent reside in the server S_i. We labeled the worker agent as *WA*, monitor agent as *MA* and the replica agent as *RcA*. Figure 4.1 illustrates the workflow of IRCFT approach.

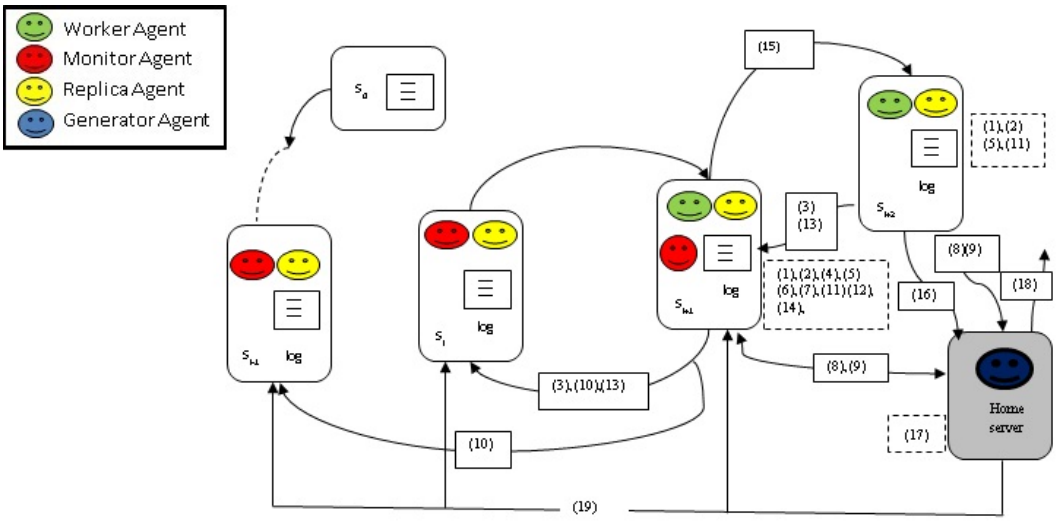


Figure 4.1: Workflow of IRCFT approach

The workflow is as follows:

1. After *WA* has arrived at S_{i+1}, it immediately registers a log_{arrive} on the permanent storage in S_{i+1}. The purpose of this log entry is to provide an evidence that *WA* has successfully landed on this server.

2. Afterwards, WA_{i+1} informs MA_{i+1} that it has arrived at S_{i+1} safely by sending a $msg_{LogArrival}$ message to MA_{i+1}.

3. MA_{i+1} informs MA_i that WA_{i+1} has arrived at S_{i+1} safely by sending a msg_{arrive} message to MA_i.

4. WA_{i+1} creates a replica RcA_{i+1}. This replica is used when the agent die.

5. Next, WA_{i+1} accomplishes the task appointed by the owner on S_{i+1}.

6. After WA_{i+1} has finished the task, it takes a checkpoint in timeout period (T_{cp}) sends it to replica. RcA_{i+1} update its status using the checkpoint.

7. WA_{i+1} sends message $msg_{checkpoint}$ to MA_{i+1}. The reason of this message is to inform the MA_{i+1} that RcA_{i+1} is updated its status.

8. WA_{i+1} sends its current address to the *manager agent* and waits for a response for address of the next server. If not response it re-sends the request message.

9. *Manager agent* sends address of next server if the current address is not in the list of the previous visited address.

10. RcA_{i+1} sends an update message msg_{update} to the previous replicas; RcA_i and RcA_{i-1}; and waits for reception of acknowledge message

(msg_{ack}) from the other replicas to be sure that all replicas are updated. This is a necessary condition for system stability in rollback recovery.

11. WA_{i+1} registers a log_{leave} in S_{i+1}. This log entry expresses that WA has completes its computation and is ready to travel to the next server S_{i+2}.

12. In the next step, WA_{i+1} sends to MA_{i+1} a message, $msg_{LogLeave}$, in order to inform MA_{i+1} that WA_{i+1} is ready to leave S_{i+1}.

13. MA_{i+1} sends message msg_{leave} to MA_i.

14. When MA_{i+1} receive $msg_{LogArrival}$ and $msg_{LogLeave}$ from WA_{i+1}, it spawns a new monitor in S_{i+1}.

15. WA_{i+1} and the *new monitor* leave S_{i+1}, and migrates to S_{i+2}. Note that the new monitor agent know where to go, i.e. S_{i+2}, because $msg_{LogLeave}$ contain information about the location of S_{i+2}.

After completing its execution on the first three servers of the itinerary the *worker agent* moves to S_{i+2} and repeats the following stapes 1-3, 5, 8-9 and 11-13 motioned above.
When WA_{i+2} sends the $log_{LogLeave}$ to MA_{i+2}, then step 16 is done.

16. WA_{i+2} moves back to the *home server*, checkpoint the data and saves the values computed from the previous four servers.

17. After saving the value and adding checkpoints, the *manager agent* creates a new monitor.

18. The WA and the *new monitor* move to the next server in the itinerary that is S_{i+3}.

19. Finally, the *manager agent* sends msg_{kill} to the $RcAs$ and MAs residing in the previous servers since they are no longer needed.

The process is repeated for every four servers in the itinerary until WA reaches the last destination in its itinerary and it returns back to the home server. Figure 4.2 and 4.3 illustrate the flow chart of the approach.

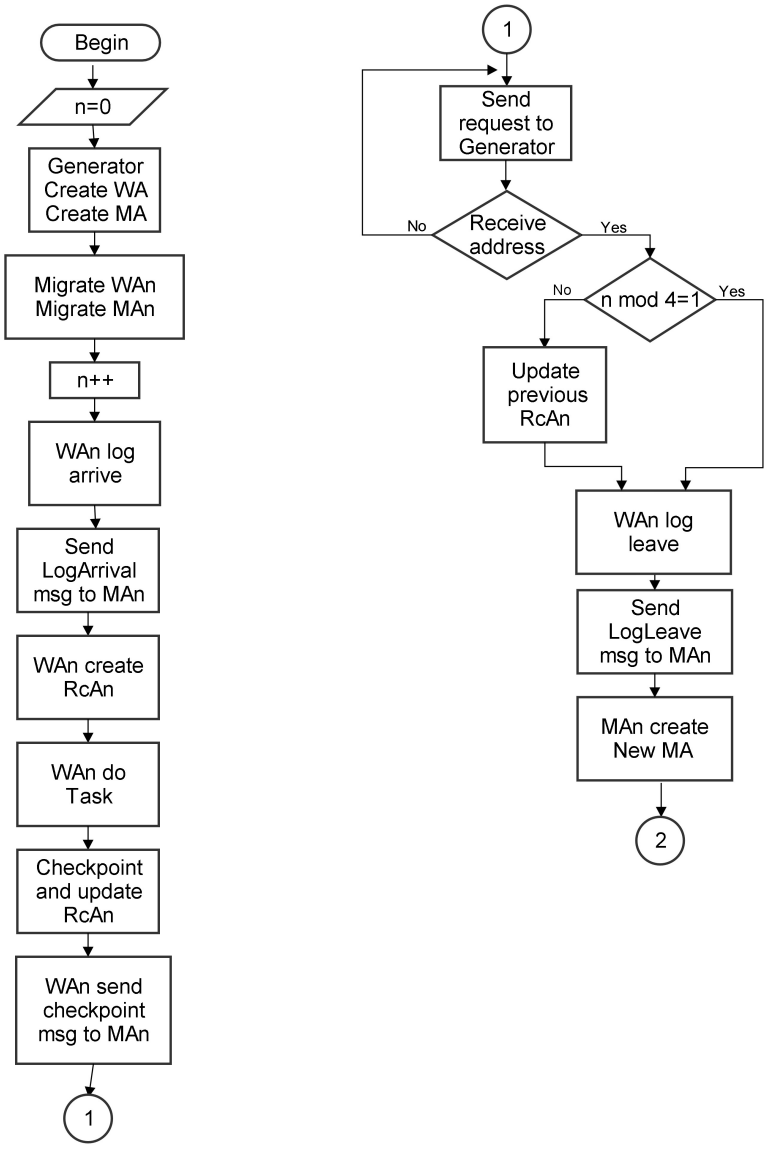


Figure 4.2: Flowchart of IRCFT approach -1-

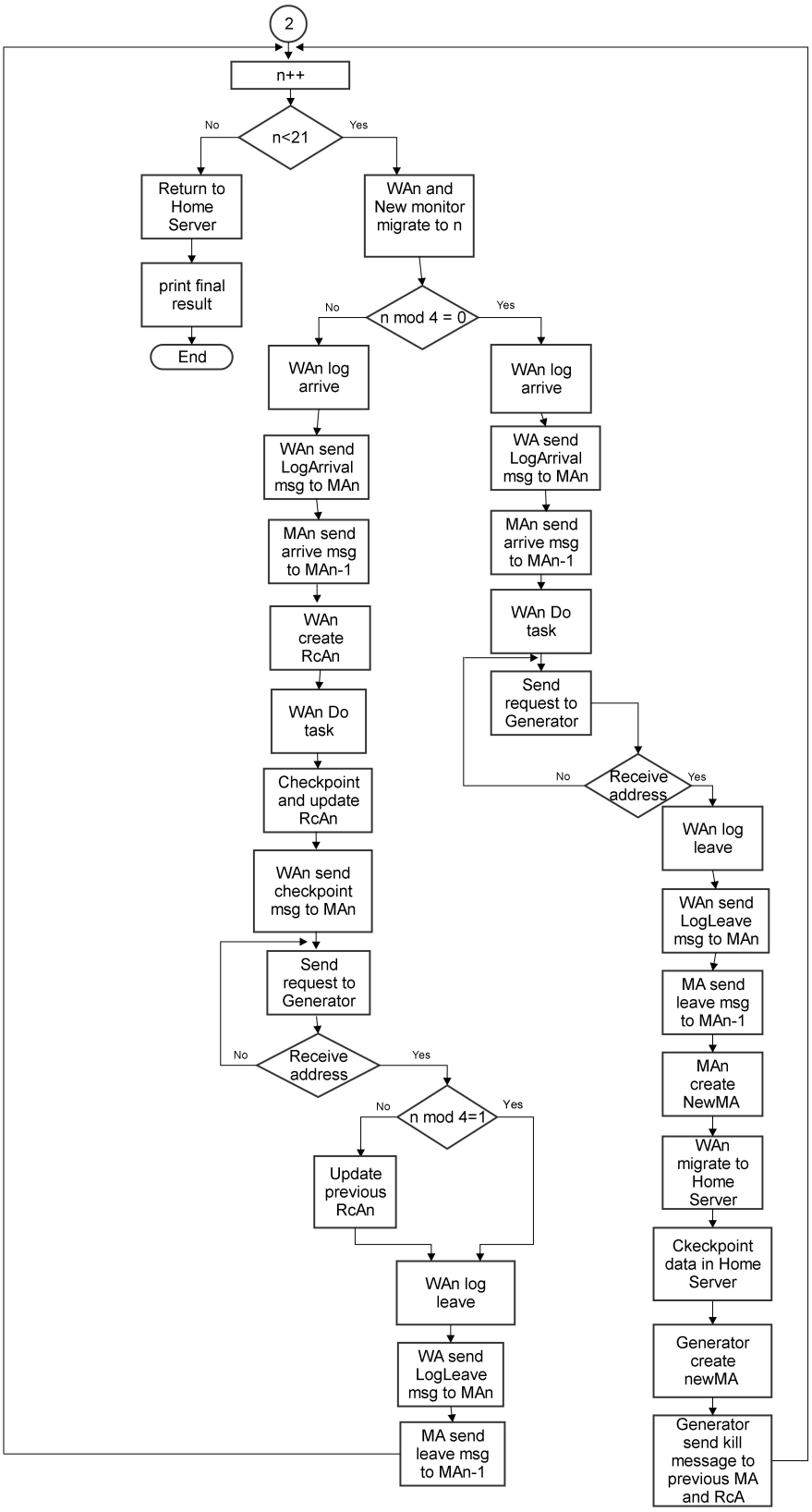


Figure 4.3: Flowchart of IRCFT approach -2-

A chain of monitors is built in the itinerary $MA_{i-1}\rightarrow$ $MA_i\rightarrow$ $MA_{i+1}\rightarrow$ $MA_{i+2}\rightarrow$ WA_{i+2}. Where MA_{i+2} and MA_{i+1} monitor the liveness of the WA_{i+2}. The others responsible to check if the next monitor is alive or not by sending among them a periodic message. MA_i sends a heartbeat message $msg_{heartbeat}$ to monitor MA_{i-1} periodicaly. Similarly, the monitor MA_{i+1} sends a $msg_{heartbeat}$ to the last monitor MA_i.

The life cycle of worker, monitor, manager and replica algorithm are shown respectively in algorithm 1, 3, 2 and 4.

Algorithm 1 Worker algorithm

```
procedure STAGE( )
    if CheckPosition() == true then
        InServer4 ()
    else
        BeforeServer4 ()
    end if
end procedure

procedure BEFORESERVER4( )
    do log arrive
    send LogArrival message to the monitor
    spawn replica
    do task
    send update message to replica
    send checkpoint message to the monitor
    if Receive ( ) ≠ Failure then
        do log leave
        send LogLeave message to the monitor
        migrate to the next server
    else
        terminate
    end if
end procedure

procedure INSERVER4( )
    do log arrive
    send LogArrival message to the monitor
    do task
```

```
    if Receive () ≠ Failure then
        do log leave
        send LogLeave message to the monitor
        migrate to the home server
        DoCheckpoint ( )
        migrate to the next server
    else
        terminate
    end if
end procedure

procedure RECEIVE( )
    resend:
    send current address   ▷ request the next address form manager agent
    wait for reply
    if timeout then
        Go to resend
    else
        if receive next address then
            return next address
        else
            return Failure
        end if
    end if

procedure CHECKPOSITION(n): Boolean
    if n mod 4 =0  then
        return true
    else
        return false
    end if
end procedure
procedure DOCHECKPOINT( )
    save partial result
    save address of the previous server
end procedure
```

Algorithm 2 Manager algorithm

```
procedure ManagerTask( )
    Receive address from the worker agent    ▷ request of the next address
    if Worker.CheckPosition(n) is true then
        if searach(address) is true then
            send Failure to the worker agent
        else
            send next address to the worker agent
        end if
    else
        spawn new monitor
        send kill to replica and monitors in the previous server
    end if
end procedure
procedure Search(address): Boolean
    while is not the end of the address file do
        if address exist in the file then
            return true
        else
            return false
        end if
    end while
end procedure
```

Algorithm 3 Monitor algorithm

```
procedure msgHandlerMonitor( )
    if msg is valid then
        switch msg do
            case LogArrival
                update TimeMonitorReceive
                send arrive to LastMonitor
            end case
            case checkpoint
                update TimeMonitorReceive
            end case
            case LogLeave
                update TimeMonitorReceive
                send leave to LastMonitor
            end case
            case arrive
                update TimeMonitorReceive
            end case
```

```
            case leave
                update TimeMonitorReceive
            end case
            case heartbeat
                update TimeLastMonitorReceive
            end case
            case IDNextMonitorandAdressNextHost
                send NewLink
            end case
            case adressNextHost
                send NewLinkWorker
            end case
            case NewLink
                update LastMonitorReceive
                send heartbeat
            end case
            case NewLinkWorker
                update LastMonitorReceive
                send arrive or leave
            end case
            case failure
                send failure to replica
            end case
            case error
                MonitorReceive
            end case
            case failurearrive
                send failurearrive to replica
            end case
            case failureleave
                send failureleave to replica
            end case
            case kill
                terminate agent
            end case
            case alive
                send IamAlive
            end case
        end switch
    end if
end procedure
```

Algorithm 4 Replica algorithm

```
procedure msgHandlerReplica( )
    if msg is valid then
        switch msg do
            case failure
                migrate to NextServer
                recover Worker
            end case
            case failurearrive
                migrate to NextServer
                verify logarriveEntry
                if logarriveEntry not Null then
                    send arrive to LastMonitor
                    send alive to Monitor
                    if msg is IamAlive then
                        return to the previous server
                    else
                        recreate Monitor
                        return to the previous server
                    end if
                else
                    send alive to Monitor
                    if msg is IamAlive then
                        recover Worker
                    else
                        recreate Monitor
                        recover Worker
                    end if
                end if
            end case
            case failureleave
                migrate to NextServer
                verify logleaveEntry
                if logleaveEntry not Null then
                    send leave to LastMonitor
                    send alive to Monitor
                    if msg is IamAlive then
                        return to the previous server
                    else
                        recreate Monitor
                        return to the previous server
                    end if
```

```
                else
                    send alive to Monitor
                    if msg is IamAlive then
                        recover Worker
                    else
                        recreate Monitor
                        recover Worker
                    end if
                end if
            end case
            case update
                update status
            end case
            case kill
                terminate agent
            end case
        end switch
    end if
end procedure
```

4.4 Agent interaction protocols of IRCFT approach

An agent interaction protocol (AIP) describes a communication pattern as an allowed sequence of messages between agents and the constraints on the content of those messages.

4.4.1 Sequence interaction among agents

Sequence diagrams are used in Agent UML to represent interaction protocols. The following figures represent the sequence diagram of IRCFT approach.

At first, the manager agent creates the *worker agent* and the *monitor agent* in the home server, where they migrate to the next server. Figure 4.4 shows the sequence diagram when the worker agent and the monitor agent are created in the home server.

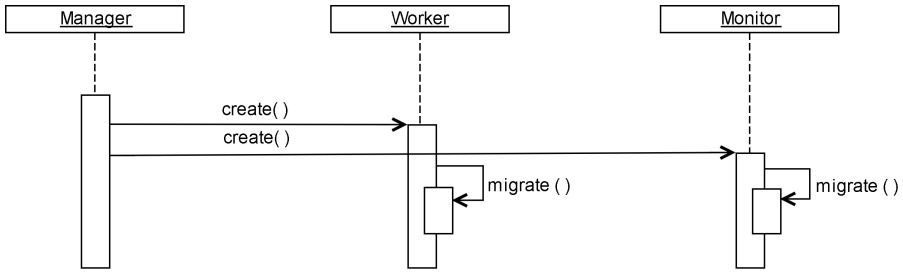


Figure 4.4: Sequence diagram 1 [Initially]

Once the worker agent and the monitor agent arrived to the next server, the worker agent immediately registers its arrive, and sends a message to the monitor agent to inform him that has arrived safely. Then the worker agent creates a *replica agent*, where it used when the worker agent die. After the worker agent completes its execution at that server, it updates the replica agent. It requests the manager agent to send him the address of the next server. In case of loss of the message due to an unreliable network, the worker agent re-transmits the request message. When the manager agent receives the message, it verifies the address of the server where the worker agent resides. If the address of the worker agent is in the list of the previous visited address, the manager agent terminates the worker agent, else it sends him the address of the next server. Finally the worker agent registers the log leave and sends a message to the monitor agent to inform him that is ready to leave the server. When the monitor receives the log leave message, it spawns a new monitor, then the worker agent and this new monitor migrate to the next server. Figure 4.5 shows the sequence diagram when the worker agent migrates to the first server of the itinerary or to the next server after the fourth server.

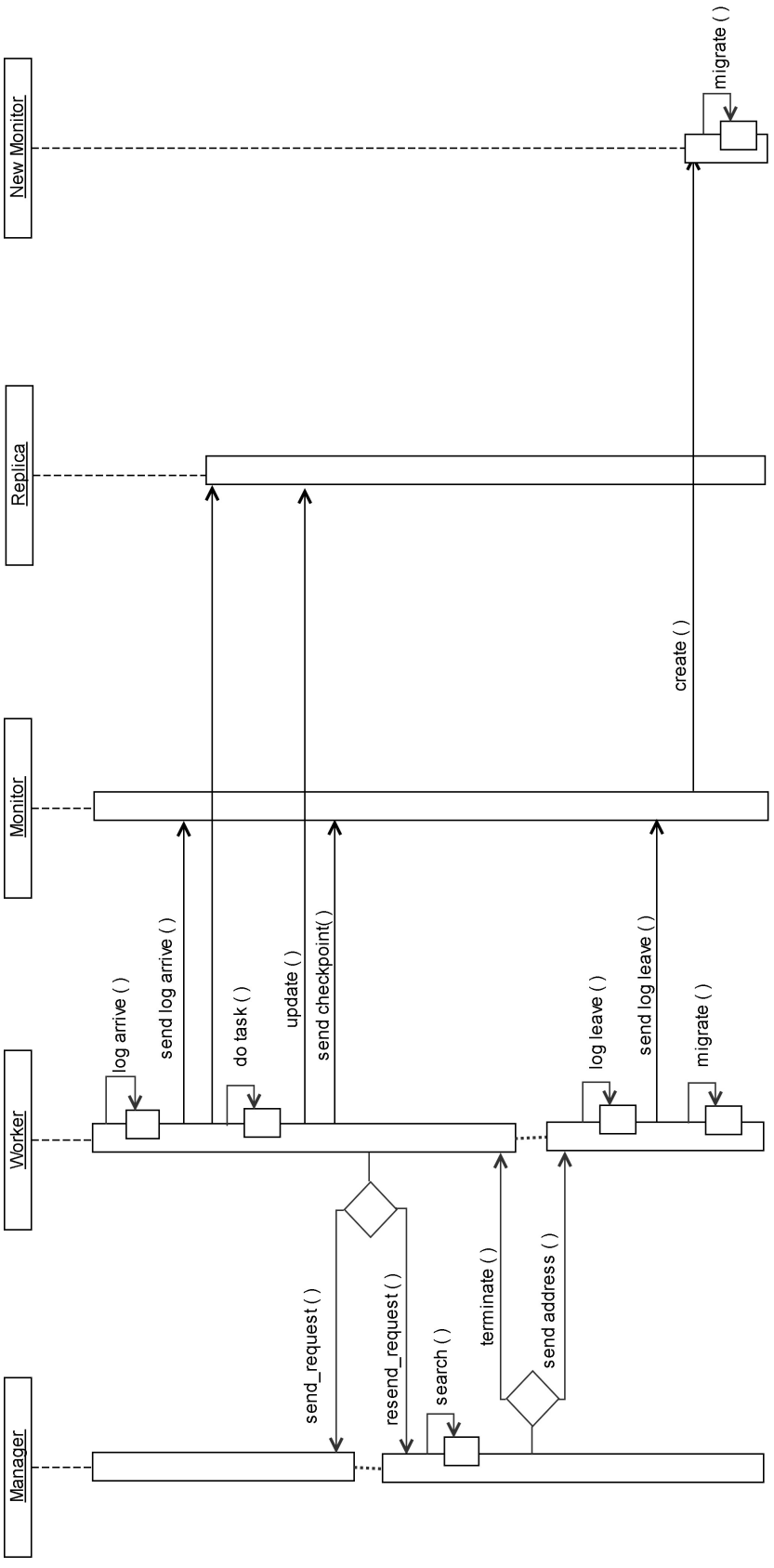


Figure 4.5: Sequence diagram 2 [In the first server]

When the worker agent does not reach the fourth server, it registers its arrive, and sends a message to the monitor agent to inform him that has arrived safely. Then, the monitor agent sends the log arrive message to the previous monitor. The worker agent creates a replica agent. After the worker agent completes its execution at that server, it updates the replica agent. The worker agent requests the manager agent to send him the address of the next server. In case of loss of the message due to an unreliable network, it re-transmits the request message. When the manager agent receives the message, it verifies the address of the server where the worker agent resides. If the address of the worker agent is in the list of the previous visited address, the manager agent terminates the worker agent, else it sends him the address of the next server. Then, the replica updates the previous replica (Prev Replica). Finally the worker agent registers the log leave and sends a message to the monitor agent to inform him that is ready to leave the server. The monitor agent sends the log leave messages to the previous monitor.

When the monitor receives the log leave message, it spawns a new monitor, then the worker agent and this new monitor migrate to the next server.

When the worker agent and the new monitor migrate to the next server the monitor agent sends a heartbeat message periodically to the previous monitor (Prev Monitor). Figure 4.6 shows the sequence diagram when the worker agent executes the task on the first three servers of the itinerary.

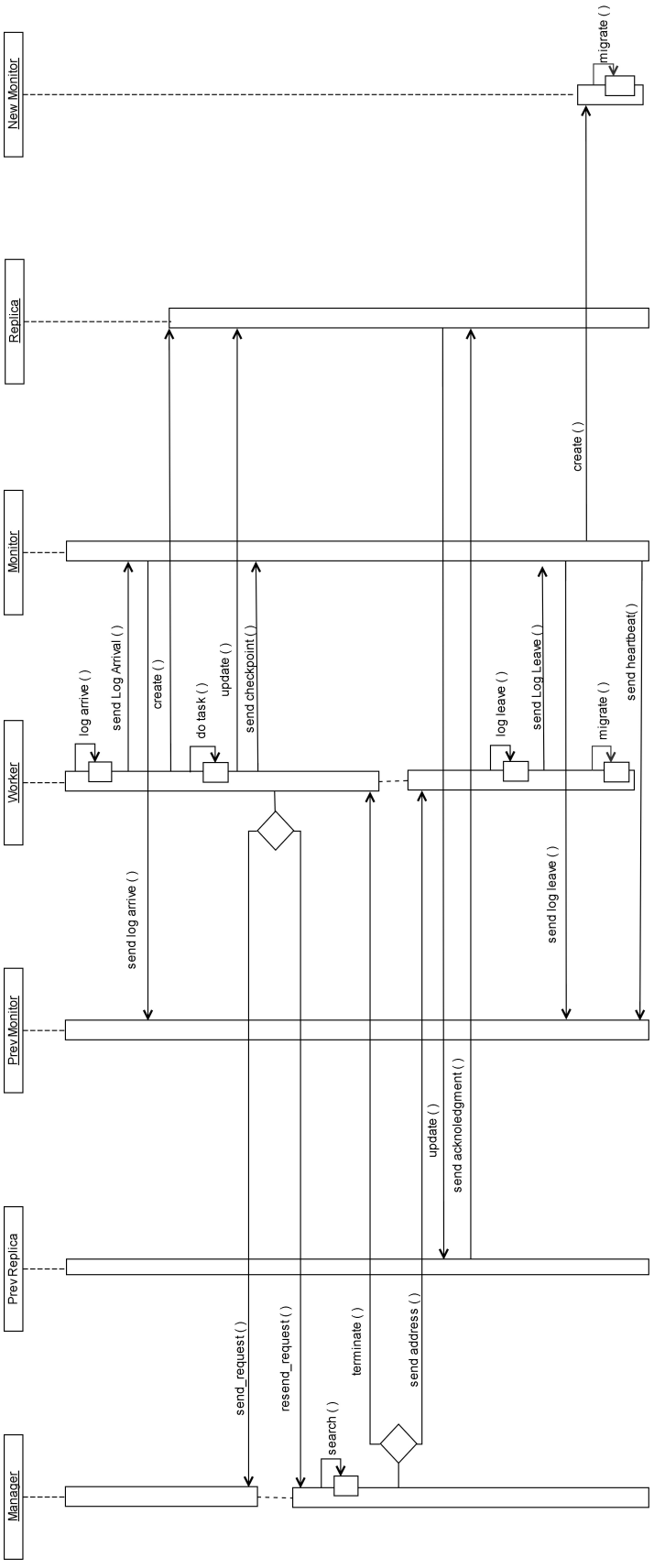


Figure 4.6: Sequence diagram 3 [Before the fourth server]

If the worker agent completes it execution on the first three servers of the itinerary, the worker agent moves to the next server which is the fourth server. In this case it will not create a replica but after registering the log leave and sending a message to the monitor agent, it moves back to the home server, checkpoints the data and saves the values computed from the previous four servers. The manager agent creates a new monitor, the worker agent and this new monitor migrates to the next server. Then, the manger agent terminates both the monitors agent and replicas agent reside in the previous servers since they are no longer needed. Figure 4.7 shows the sequence diagram when worker agent moves and reaches the fourth server respectively.

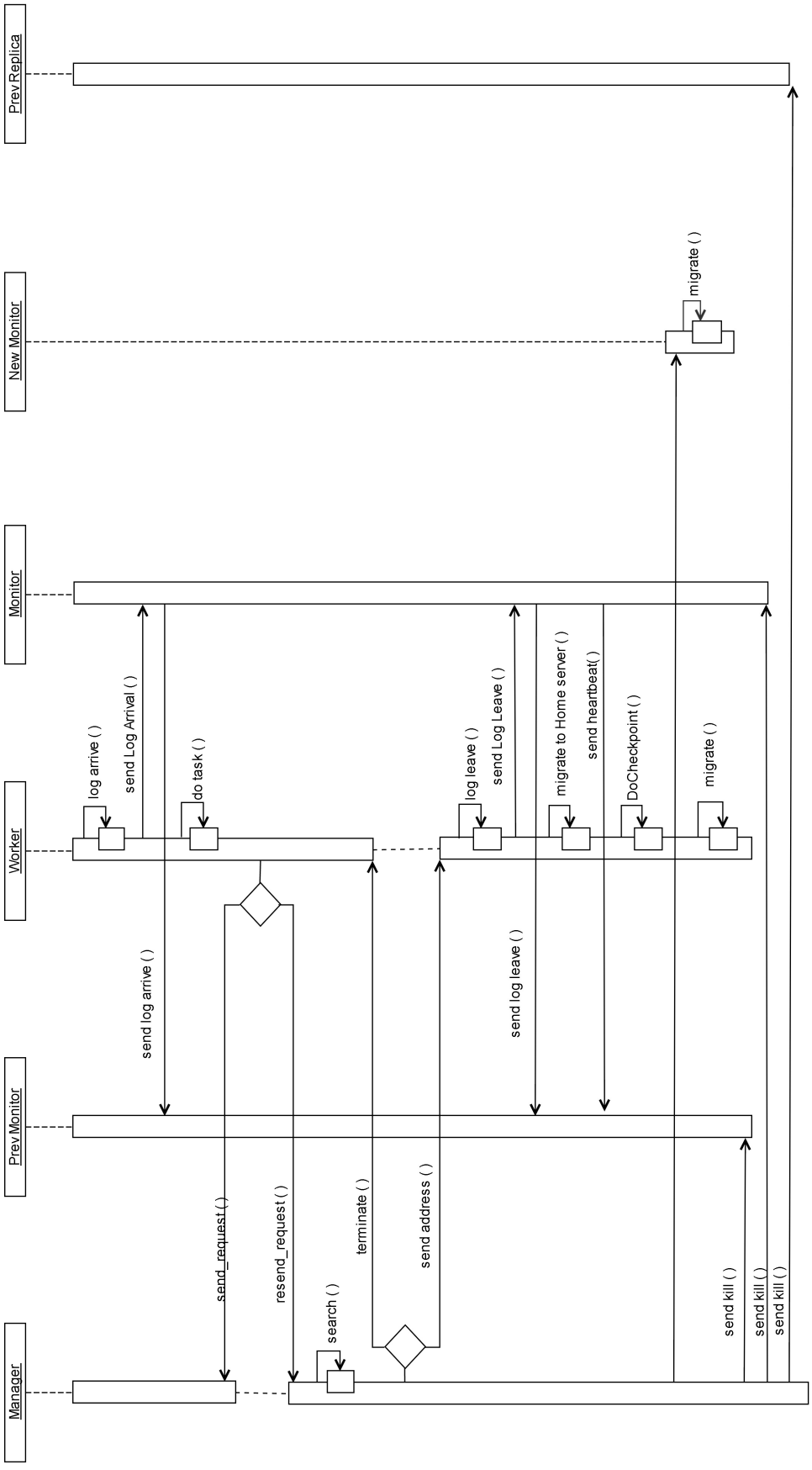


Figure 4.7: Sequence diagram 4 [In the fourth server]

4.5 Failure detection and recovery scenarios

In this section, we discuss different scenarios with the presence of failure. We describe the actions of the different agents in this approach in order to detect and recover the loss of the worker agent and also the monitor agent.

The current server is S_{i+1}. Figure 4.8 shows the different scenarios of failure in the first three servers.

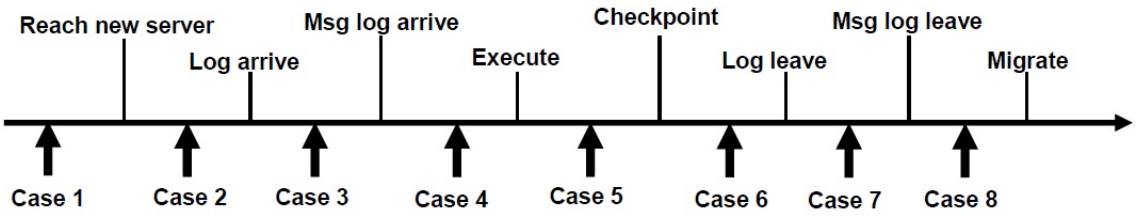


Figure 4.8: Different scenarios of failure in the first three servers

We describe several scenarios:

1. The monitor MA_{i+1} fails to receive $msg_{LogArrival}$;
2. The monitor MA_{i+1} fails to receive $msg_{checkpoint}$;
3. The monitor MA_{i+1} fails to receive $msg_{LogLeave}$;
4. The monitor MA_i fails to receive msg_{arrive} or msg_{leave};
5. WA_{i+1} and RcA_i fail;
6. Monitor failure;
7. Bad servers; and
8. Failure in the fourth server.

Safe case

In the safe case, MA_i successfully receive msg_{arrive} and msg_{leave} from MA_{i+1} and the WA_{i+1} successfully completes execution on S_{i+1} and moves safely to the next server S_{i+2}.

The monitor MA_{i+1} fails to receive $msg_{LogArrival}$

- **Case 1, 2, 3 and 8.** The reason can be:

  1. WA_{i+1} is terminated when it is ready to leave S_i;
  2. WA_{i+1} is terminated when it has just arrived at S_{i+1}; without logging; or
  3. WA_{i+1} is terminated when it has just arrived at S_{i+1} with logging and before sending $msg_{LogArrival}$ to MA_{i+1}.

In this cases MA_{i+1} waits for the message $msg_{LogArrival}$ for timeout period ($T_{LogArrival}$). If the timeout is reached, it sends to MA_i failure message $msg_{failure}$. Then MA_i sends $msg_{failure}$ to RcA_i. RcA_i travel to S_{i+1} to recover the failure. After the recovery is completed, the recovered worker agent can start performing it computation.

The monitor MA_{i+1} fails to receive $msg_{checkpoint}$

- **Case 4, 5 and 6**. The reason can be:

  1. WA_{i+1} is terminated when MA_{i+1} has just sends msg_{arrive} to MA_i;
  2. WA_{i+1} is finished its execution; or
  3. WA_{i+1} is finished the checkpoint to update RcA_{i+1}.

In this case MA_{i+1} waits for the $msg_{checkpoint}$ for timeout period ($T_{checkpoint}$). If the timeout period is reached, it sends to M_i a failure message $msg_{failure}$. Then MA_i send $msg_{failure}$ to RcA_i and travel to S_{i+1} to recover the failure.

The monitor MA_{i+1} fails to receive $msg_{LogLeave}$

- **Case 7**. The reason can be:

  1. WA_{i+1} is terminated when it has just sends $msg_{checkpoint}$ to MA_{i+1}; or
  2. WA_{i+1} is terminated when it has just logged the leave entry.

In this case MA_{i+1} waits for the $msg_{LogLeave}$ for timeout period ($T_{LogLeave}$). If the timeout is reached it sends to MA_i error message msg_{error} to inform its that recovering will be through the replica locates in the server S_{i+1}. The error message will be sent also to RcA_{i+1}. Then RcA_{i+1} recover the failure.

The monitor MA_i fails to receive msg_{arrive} or msg_{leave}

The reasons can be:

1. msg_{arrive} or msg_{leave} is lost due to an unreliable network;
2. msg_{arrive} or msg_{leave} is arrived after the timeout period of MA_i; or
3. WA_{i+1} and MA_{i+1} are terminated due a crash failure.

If the failure is because of the first two reasons, the WA_{i+1} is still alive in S_{i+1}. In this case MA_i waits for the message for timeout period (T_{arrive} or T_{leave}). If the timeout is reached MA_i sends $msg\_failure_{arrive}$ to RcA_i to verify the log_{arrive}; or $msg\_failure_{leave}$ to verify log_{leave}. Then RcA_i travels to S_{i+1} to search for log_{arrive} or log_{leave} in S_{i+1}. If found, then RcA_{i+1} re-transmits

msg_{arrive} or msg_{leave} to MA_i, then RcA_{i+1} sends a message to MA_{i+1} to verify if it is lost or no. If RcA_{i+1} fails to receive the response, it creates a new monitor and returns to S_i.

If MA_i fails to receive msg_{arrive} because of the loss of MA_{i+1} and WA_{i+1}, it sends $msg\_failure_{arrive}$ to the RcA_i and travel to S_{i+1} to search for log_{arrive}. Upon arriving at S_{i+1}, it searches the log file for the entry log_{arrive}. If the log entry is not found, RcA_{i+1} sends a message to MA_{i+1} to verify if it is died or no. If RcA_{i+1} fails to receive the response, it creates a new monitor, recover the WA_{i+1} and re-transmits the message msg_{arrive} to MA_i.

When MA_i fails to receive msg_{leave} because of the loss of MA_{i+1} and WA_{i+1}, it sends $msg\_failure_{leave}$ to the RcA_i and travel to S_{i+1} to search for log_{leave}. Once the RcA_i reached S_{i+1}, it searches the log file for the entry log_{leave}. If it is not found, RcA_{i+1} sends a message to MA_{i+1} and waits for a response. If it fails to receive the response, it creates a new monitor and recover the worker agent. If the replica receives the update before the failure then it continues the execution, otherwise it repeats the task.

WA_{i+1} and RcA_i fail

In this case MA_{i+1} sends $msg_{failure}$ to MA_i. MA_i detects that RcA_i is terminated then, it re-sends the failure message to MA_{i-1} in the server S_{i-1}. Then, RcA_{i-1} can migrate and recover the failure in S_{i+1}. Figure 4.9, illustrates a recovery procedure of worker agent and replica agent failure.

Monitor failure

In the life cycle of a monitor, the monitor periodically sends a heartbeat message to the last monitor. It is performed considering the following chain:

$MA_{i-2} \rightarrow MA_{i-1} \rightarrow MA_i \rightarrow MA_{i+1} \rightarrow WA_{i+1}$.

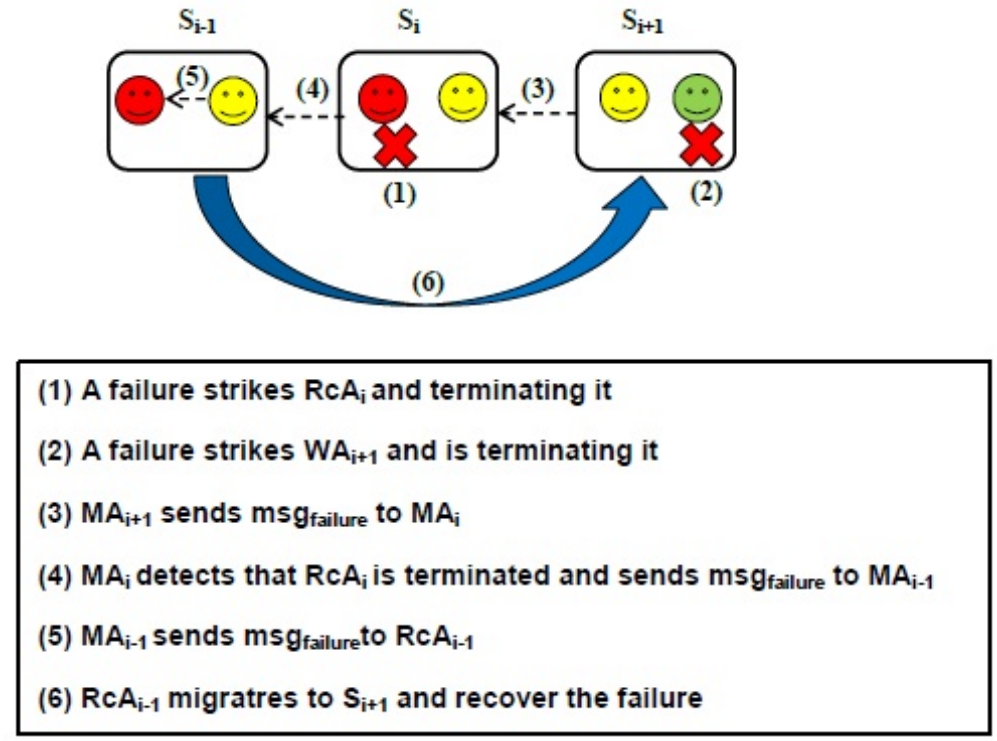


Figure 4.9: WA_{i+1} and RcA_i failure scenario

If MA_{i-2} does not receive the heartbeat message;$msg_{heartbeat}$ from MA_{i-1} after several attempts, then it suspects that MA_{i-1} is down. Table 4.1 shows the parameters used in IRCFT's evaluation. The monitor MA_{i-2} will request the identity and the server address of the next monitor from the *manager agent*. Once MA_{i-2} receive the identity and the address of the next server, it will exclude MA_{i-1} from the chain by linking with monitor MA_i. If MA_i receives $msg_{NewLink}$ from a MA_{i-2}, it will consider it as the last monitor and reply with $msg_{heartbeat}$.

If MA_{i-1} does not receive $msg_{heartbeat}$ from MA_i. Then it suspects that MA_i is down. In this case, MA_{i-1} will request the identity and the server address of the next monitor from the *manager agent*. The *manager agent* will send the next address without the identifier. Then, MA_{i-1} send $msg_{NewLinkWorker}$ and only MA_{i+1} receive the message and reply with msg_{arrive} or msg_{leave}(Depending

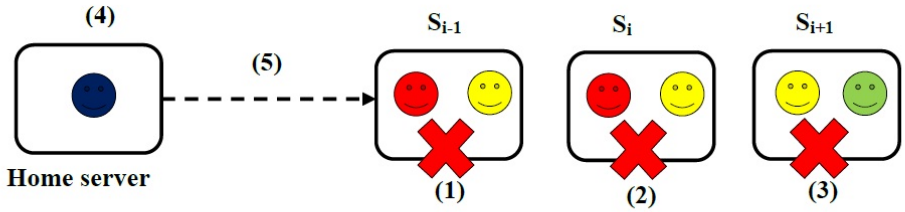
Table 4.1: Parameters used in IRCFT approach's evaluation

Parameters	Value
Heartbeat period	$period_{heartbeat}$
Tolerance of missing heartbeat	3

on where the worker arrived in the execution).

Bad servers

When bad servers increases in the itinerary, both the worker agent, the monitors and replicas are down. In this situation, the checkpoint in the home server recovers the failure. After a period of time, a fault message is sent to *manager agent*. It creates a new monitor and a replicated copy of the agent to starts its execution from the immediate checkpoint saved in the home server. Figure 4.10 illustrate a recovery procedure of bad servers.



(1) A failure strikes S_{i-1} and MA_{i-1} and RcA_{i-1} are terminated

(2) A failure strikes S_i and MA_i and RcA_i are terminated

(3) A failure strikes S_{i+1} and WA_{i+1} and MA_{i+1} are terminated

(4) Manager Agent detects the failure and creates a copy of agent

(5) The copy of agent starts from the immediate checkpoint before the faulty server S_{i-1}

Figure 4.10: Bad server scenario

Failure in the fourth server

When the *WA* migrates to the fourth server and terminates the execution, the *RcAs* and the *MAs* in the previous servers will be killed, then if the *WA* stops its execution due to any fault when it is in the next server. In this situation, a fault message is sent from the *MA* to *manager agent* in the home server and we have no other option than to send the replicated copy of the agent. The data retrieved from the previous serves is already saved to the home server with the checkpoints after every four servers. So, the replicated agent needs not to rollback to the first server in the itinerary. The replicated agent starts its execution from the immediate checkpoint.

4.6 Discussion

Preservation of the exactly-once property, non blocking property and management of monitors are discussed.

4.6.1 Preservation of the exactly-once property

The exactly-once execution property is guaranteed in IRCFT approach. The replica agents are not executing while the original executing agent is active, and it is impossible for more than one *worker agent* to gain the acceptance from the *manager agent* for the same stage, and it never receives the next server. In this case the duplicate agent will be terminated.

4.6.2 Preservation of non-blocking property

The non-blocking feature is guaranteed even in the case of multiple failures by allowing the last checkpoint or the replica of the crashed agent to replace it in order to continue execution even in the case of agent failures. Hence, both

checkpoint and replication is introduced in the suggested approach to solve the blocking problem of the mobile agent execution where the replication and checkpoint masks failures and ensures progress of the mobile agent execution.

4.6.3 Management of monitors failure

In this approach, new link is used to maintain the chains of monitors. The servers with high crash rate are eliminated from the chain, because IRCFT approach update the replica each time after the completion of task. So, in case of failure one of the replica can be used to recover the failure. Instead that done PFTM [32], a new monitor will be created in order to replace the lost monitor agent in the server.

4.7 Main differences between IRCFT and other approaches

IRCFT approach is a development of that in FTIRC approach [23] and PFTM approach [32]. IRCFT approach differs from these approaches in several points, that are summarized as follows:

- IRCFT approach and PFTM approach [32] communicate at different locations by exchanging messages through unreliable communication channels but in FTIRC approach [23] the authors use the reliable network.

- IRCFT approach and FTIRC approach [23] use the checkpoint in the home server; the checkpoint serve to save partial results after completing the execution in a number of servers of the itinerary. In IRCFT approach, the checkpoint usage was after completing its execution on

the first four servers, but in FTIRC approach [23] uses it after three servers.

- In FTIRC approach [23] and PFTM approach [32] the mobile agent knows its itinerary. In IRCFT approach the *worker agent* requests the address of next server from the *manager agent*. For this reason, the exactly-once feature is guaranteed. In FTIRC approach [23], the exactly-once property is violated due to network congestion. The agent owner consider that the agent has been terminated and launched another agent, and this cause multiple executions of the agent.

- IRCFT approach minimizes the monitor agents and replicas by terminating them by the *manager agent* after the checkpoint in the home server; because existence of all of monitor agent is not necessary on the initial servers [32].

- PFTM approach [32] recovers the lost monitor. This is achieved by preserving the monitoring dependency: the recovery of MA_{i-1} can be performed by MA_{i-2}. MA_{i-1} will be created in order to replace the lost monitor agent in the server S_{i-1}. In IRCFT approach recovery of monitor is ignored.

- PFTM approach [32] stores the checkpoint data in a stable storage, it is used in case of failure. IRCFT approach uses the checkpoint to update the *replica agent* located in the server and the other previous replicas.

- In case of failure in FTIRC approach [23], the checkpoint in the home server is used to recover the failure and the replicated copy of the original agent will be sent to the immediate checkpoint before the fault.

But in IRCFT approach, the update replica is used to recover the failure, and it uses the checkpoint in the home server to recover the failure in case where all replicas are terminated and there is no way to recover the failure using the update replica.

Chapter 5

Experiments and Results

In this chapter we present the experimental results of applying IRCFT approach. Several experiments were performed to evaluate IRCFT approach in terms of parameters such as round-trip time and agent survivability.

5.1 Experimental settings

The proposed approach has been implemented using AGLETS-2.0.2. Results are evaluated by various experiments and represented below in the form of graph charts, where faults may occur on random servers.

Reliability in this thesis is measured by the success rate of worker agent in completing their scheduled round-trip travels. Total trip time is also measured, where it is the time required by mobile agent to complete its itinerary and return to the home server.

We carry out the experiment by using different itineraries with various lengths. We assume that the home server is error-free, the other servers are error-prone.

Our experiment is carried out by implementation with up to 20 servers and each experiment was performed 30 times. Average values are used to

express the result. The normal execution time of the agent on each server is assumed to be 1 sec or 1000 ms.

In the experiments, the normal round trip time and survivability of agent in the suggested approach, IRCFT, is compared to that without fault tolerance mechanism (WFTM) and to the approaches FTIRC [23] and PFTM [32].

5.1.1 Effect of round trip time when there is no failures

The normal round trip time of agent without fault tolerance is compared to the round trip time of IRCFT approach and that suggested in FTIRC approach [23] and PFTM approach [32]. The results is shown in table 5.1. In our experiments we have considered an itinerary consisting of 4, 7, 9, 13,16 and 20 servers.

Table 5.1: Effect of round trip time when there is no failure

Server	**4**	**7**	**9**	**13**	**16**	**20**
WFTM	5585	9536	12170	17438	21389	25340
FTIRC	5922	10210	213181	318786	23074	28679
PFTM	6525	11106	14160	20268	24849	30957
IRCFT	7662	12838	16429	23554	29037	36162

Figure 5.1 shows that the round trip time in IRCFT is higher than the round trip time of the mechanism used in FTIRC approach [23] and PFTM approach [32]. This is because of the processing required for implementing fault tolerance; log, sending messages between agents, update the replicas agent and adding checkpoint and the last server visited by the worker agent after every four servers in the itinerary; that adds to the overheads and leads to the increase in the round trip time of the itinerary.

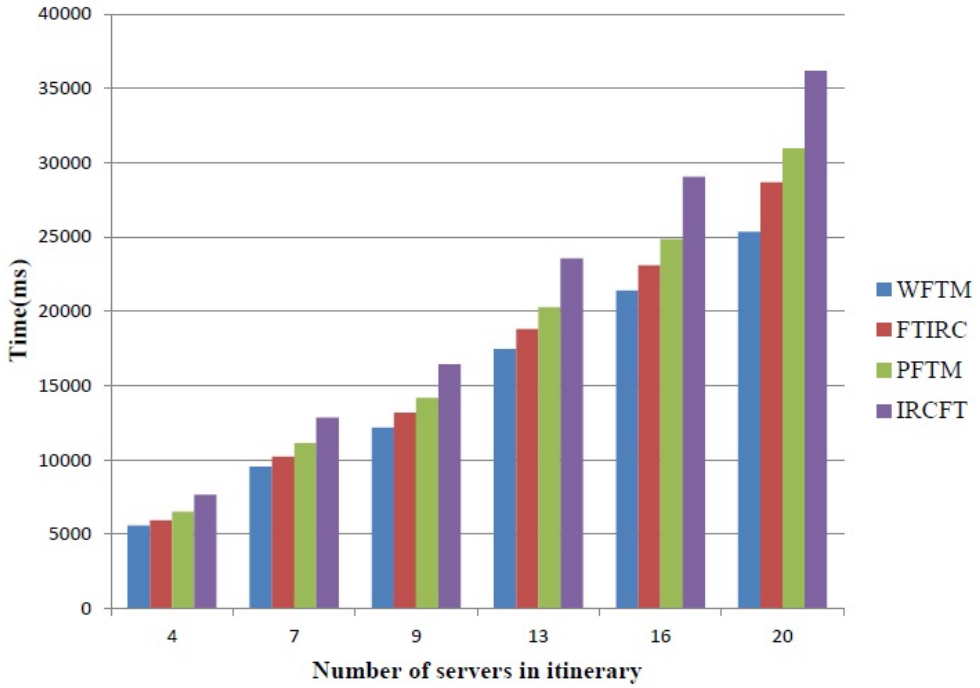


Figure 5.1: Round trip time without considering failures

5.1.2 Effect of round trip time by considering several agent failures

Table 5.2 shows the comparison results of the times of total trips needed to complete its itinerary when fault happens to various servers. In the experiments we have considered an itinerary consisting of 3, 7, 11, 15 and 20 servers.

Table 5.2: Effect of round trip time with considering several fault

Failure in the server	**3**	**3,7**	**3,7,11**	**3,7,11,15**	**3,7,11,15,20**
WFTM	8002	22589	42444	67467	99392
FTIRC	8681	18679	28677	39899	50214
PFTM	6258	13943	22195	29880	39409
IRCFT before checkpoint	6768	15265	23762	32259	42725
IRCFT after checkpoint	5591	12911	20231	27551	38017

An agent was manually killed to test the approach, this happens by killing the agent thread on servers particular. The agent is killed after completing its associated task.

In IRCFT approach, we have two cases: in the first case, the agent dies after finishing work and before the process of checkpoint and update of the replica located in the server. In this case, the monitor agent waits for the $msg_{checkpoint}$ for timeout period ($T_{checkpoint}$). If it is timeout, it sends to the previous MA a failure message $msg_{failure}$. Then MA sends a $msg_{failure}$ to RcA and travels to the fault server to recover the failure.
The second case, the worker agent terminate after sending a checkpoint message to the monitor agent. In this case, MA waits for the $msg_{LogLeave}$ for timeout period ($T_{LogLeave}$). If it is timeout, it sends to the previous monitor an error message msg_{error}. Then, MA send a the error message (msg_{error}) to the replica located in the server to recover the failure.

According to the figure 5.2, we noticed that the total trip time of agent in IRCFT approach is less when fault occurs after sending a message of checkpoint compared to the total trip time of agent after ending a task without updating the replica. This is because the replica does not need to migrate or do the task once again.

The time of the normal round trip without fault tolerance to complete faults happens on any server is greater than the round trip time in FTIRC approach [23], PFTM approach [32] and our approach IRCFT. This happens because when there is failure on any server, the replicated agents starts again from the beginning.

The total trip time of agent in IRCFT for the two cases of the approach is less than time in FTIRC approach [23]. This happens because in process FTIRC approach [23] the home server detects the fault after receiving the

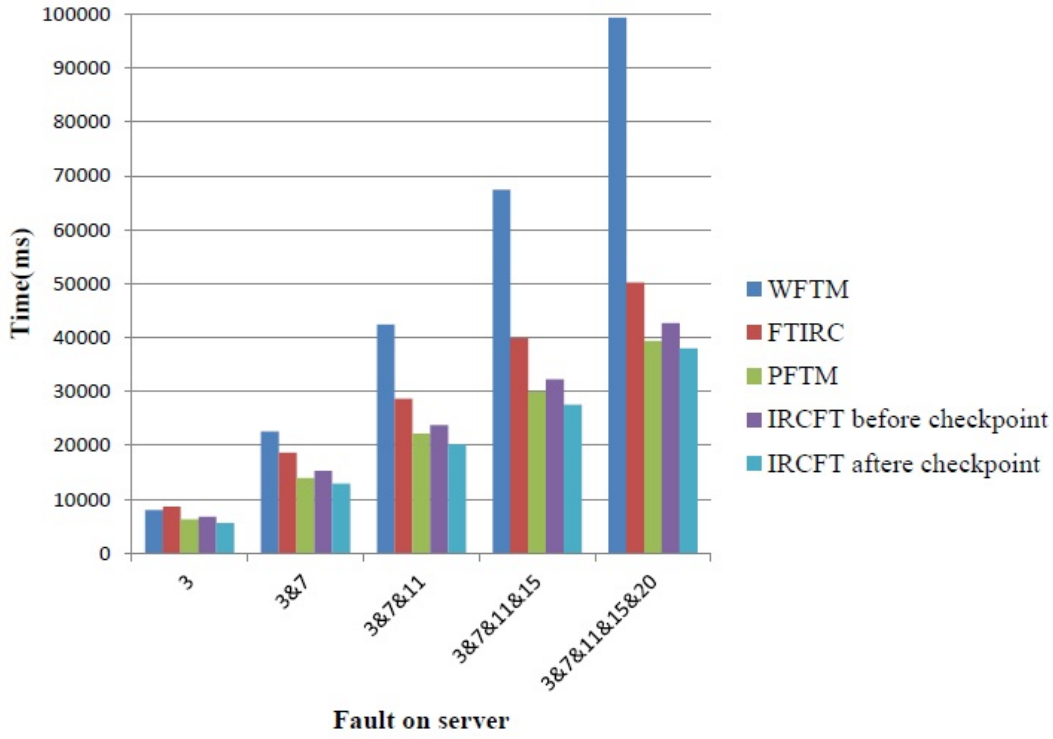


Figure 5.2: Round trip time with considering several fault

fault message and require to start its trip from the checkpoint before the faulty server. The total trip time of agent in our approach in case recovering after checkpoint is less than the round trip time in PFTM approach [32], too.

5.1.3 Effect of round trip time by considering several agent faults and blocking of the previous server

Table 5.3 shows the comparison result of the total trips needed to complete its itinerary when fault happens to several servers and with blocking the previous server every time i.e. if an agent failure occurs in the server number three, another failure block the previous server (server number two) for 200 ms. In the experiments we have considered an itinerary consisting of 3, 7, 11, 15 and 20 servers.

Table 5.3: Effect of round trip time when there is no failure

Failure in the server	**3**	**3,7**	**3,7,11**	**3,7,11,15**	**3,7,11,15,20**
WFTM	8002	22589	42444	67467	99392
FTIRC	8681	18679	28677	39899	50214
PFTM	8332	18091	27850	37609	49212
IRCFT before checkpoint	6918	15565	24212	32859	43288
IRCFT after checkpoint	5591	12911	20231	27551	38017

We noticed in the figure 5.3 that total trip time of agent in IRCFT for the two cases of the approach is less than time in FTIRC approach [23] and PFTM approach [32]. This happens because new link is used in case of IRCFT before checkpoint to maintain the chains of monitors where we can eliminate the servers with high crash rate from the chain (in case the fault in S_{11}, the server S_{10} will be blocked for 200 ms). In this case MA_9 suspects that MA_{10} is down, and by using the new link it detects the failure in S_{11} and sends the RcA_9 to recover the failure. This method reduces the waiting time of recovering the server and creating a new monitor to recover the failure as in PFTM approach [32].

Figure 5.3 shows that round-trip time of agent is less in case of IRCFT after checkpoint, this happens because the replica located in the failed server recover the worker agent and does not need to do the task again.

5.1.4 Agent survivability

We define the metric, agent survivability, as the successful ratio of worker agents in completing their scheduled round-trip time travels in a network of agent servers. The survivability result of each approach is shown in table 5.4.

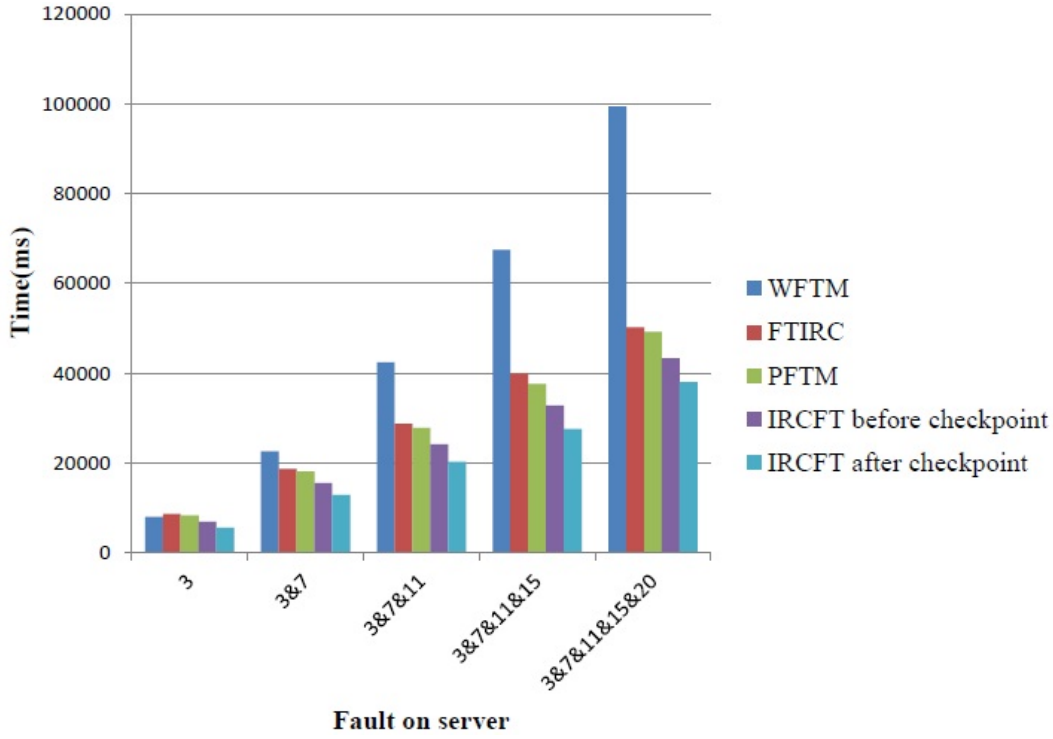


Figure 5.3: Round trip time with considering several faults and block the previous server for 200 ms

In the experiments we have considered an itinerary consisting of 1, 4, 12, 16 and 20 servers.

Table 5.4: Effect of round trip time when there is no failure

Number of servers	**1**	**4**	**8**	**12**	**16**	**20**
WFTM	0.85 %	0.75 %	0.50 %	0.45 %	0.22 %	0.1 %
FTIRC	1 %	0.93 %	0.86 %	0.76 %	0.70 %	0.67 %
PFTM	1 %	1 %	0.93 %	0.9 %	0.86	0.83%
IRCFT	1 %	1 %	0.96 %	0.93 %	0.90 %	0.87%

Figure 5.4 shows the performance of the system when the number of visited hosts increases. As shown, there is a great difference of the worker survivability between "with fault-tolerant mechanism" and "without fault-tolerant mechanism". If there is no fault-tolerant mechanism, the worker survivability decreases when the number of visited servers increases. On

other hand, the proposed fault tolerance mechanism is able to protect the worker agent from failure even when the number of servers increases.

The result indicates that the agent survivability in IRCFT approach is further improved (0.87%) with respect to that in FTIRC approach [23] (0.67%) and PFTM approach [32] (0.83%).

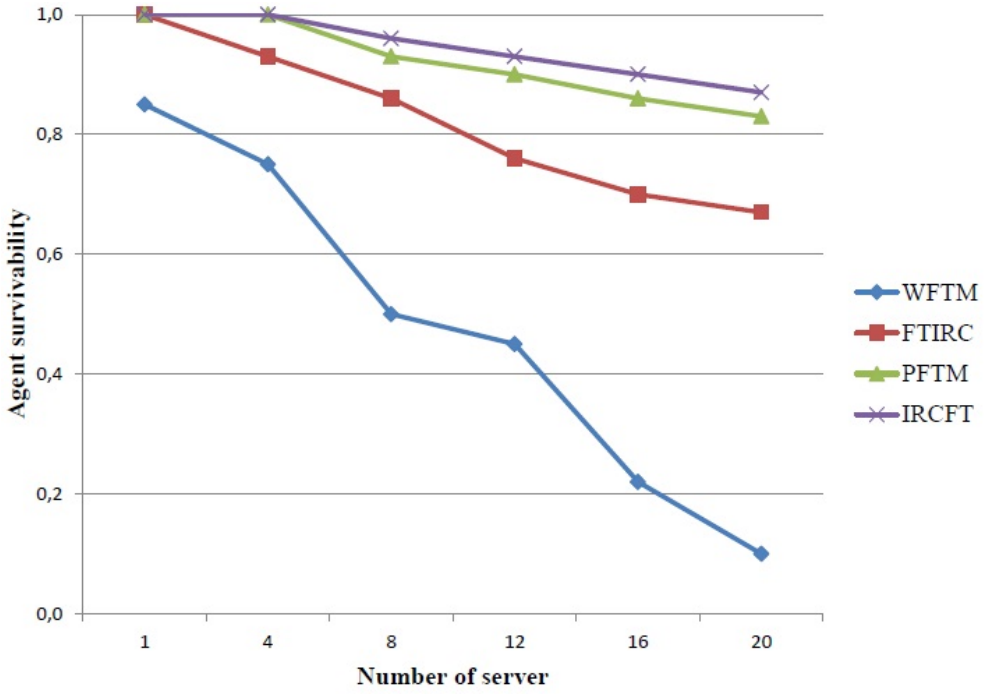


Figure 5.4: Agent survivability against the number of visited hosts

Chapter 6

Conclusion and Future Work

In this chapter, we conclude the work performed in this thesis by summarizing our contributions. In second section, we will suggest some future research directions that can provide the others with new steps.

6.1 Conclusion

Using the mobile agent in the distributed applications causes a sets of issues concerning its standards, applications and performance. The reliability of the mobile agent is the most important performance measurement. IRCFT approach is developed to solve the mobile agent failures and to achieve high level of reliability even in case of failures.

The results obtained from the work can be summarized into three points:

1. Developing a mobile agent approach, IRCFT, which is fault tolerant mechanism to increase the network reliability even in case of failures.

2. Analyzing the performance parameters of IRCFT approach. The performance factors are measured against some defined performance metrics.

3. Comparing the results obtained to those of FTIRC approach [23] and PFTM approach [32].

The novel fault tolerance approach named as Integrated Replication-Checkpoint Fault Tolerance Approach "IRCFT" detects agent failures and recovers services in mobile agent systems.
IRCFT approach is a development of that in FTIRC approach [23] and PFTM approach [32]. Four types of agents are involved in IRCFT approach, which are the *Worker agent*, the *monitor agent*, the *manager agent* and the *replica agent*. It uses the replication agent and also added checkpointing, so as to limit the rollback of agent in case of failure.

IRCFT approach uses the checkpoint in the home server; the checkpoint serve to save partial results after completing the execution in a number of servers of the itinerary. The checkpoint usage was after completing its execution on the first four servers. IRCFT approach updates the *replica agent* located in the server and the other previous replicas.

IRCFT approach can handle server and agent failure, it is capable of detecting and recovering most failure scenarios in mobile agent system. In case of failure in IRCFT approach, the update replica is used to recover the failure, and it uses the checkpoint in the home server to recover the failure in case where all replicas are terminated and there is no way to recover the failure using the update replica.

In order to evaluate the performance of IRCFT approach we have implemented it on the Aglet mobile agent system and evaluated in terms of parameters such as total round trip time and agent survivability.
Implementation of the proposed approach shows good results by improving the total trip time and the reliability (0.87%) in case of failure, this results by using the update replica or the process of checkpoint in the home server

after visiting four servers.

The mechanism of IRCFT approach makes sure that two significant properties of the mobile agent execution are satisfied even if there is failure. These two properties are exactly-once and non-blocking feature.

The non-blocking feature is guaranteed even in the case of multiple failures by allowing the last checkpoint or the replica of the crashed agent to replace it in order to continue execution. Hence, both checkpoint and replication is introduced in the suggested approach to solve the blocking problem of the mobile agent execution where the replication and checkpoint masks failures and ensures progress of the mobile agent execution.

The exactly-once property is guaranteed also in IRCFT approach. The agent replicas are not executing while the original executing agent is active, and it is impossible for more than one worker to gain the acceptance from the manager agent for the same stage, and it never receives the next server. In this case the duplicate agent will be terminated.

However, the improvement in agent survivability and round trip time is achieved by increasing the complexity of the algorithms and specially the tasks of the monitor agent and replica agent. Therefore, how to achieve the expected improvement with less complexity of the algorithm is a trade-off issue and should be investigated. We can reduce this complexity in IRCFT approach by reducing the tasks of monitors. Monitors will be only receive the messages from the worker agent instead of sending messages between them. They will receive two messages: one is msg_{arrive} and the other is msg_{leave} from the worker agent. This messages will be utilized in recovering the lost worker agent. After receiving these two messages, the monitor agent waits for the direct heartbeat message that used to detect the lost monitor agent. Monitors will only send a message to the replica in case of failure of

the worker agent, and to the manager agent in case of failure in the former server. So monitor will be more passive than the worker agent.

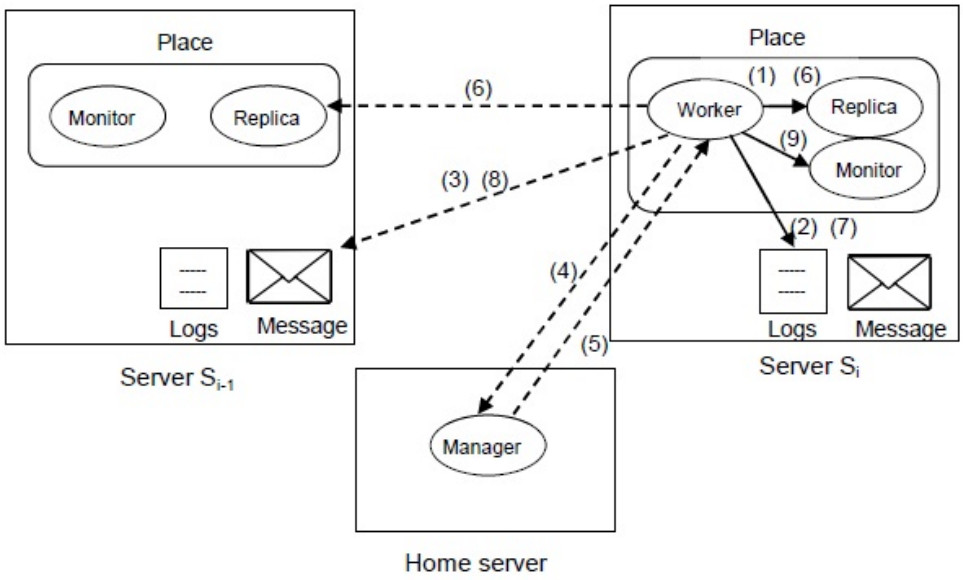
We add a new task to the worker agent, this task is the creation of a new monitor before leaving the server. If the worker agent is in the first three server, it will be create a replica agent on arrival to the server and will update this replica agent and the other replicas agent locate in the previous servers after the computation.

To reserve the bandwidth, the worker agent must sends it current address and requests a commitment from the manager agent and waits for the reply instead of requesting the address of the next server. In this case the migration path of the worker agent will be dynamic. Figure 6.1 and 6.2 show the improvement to IRCFT approach, where figure 6.1 shows the new tasks of IRCFT approach when the worker agent is in the first three server and figure 6.2 when the worker agent is in the fourth server.

6.2 Future work

In the future, the IRCFT approach will be applied to real-world mobile agent systems. Our approach can tackle the stopping failure happened in the fault detection and recovery procedures. We can further extend the mechanism to handle the byzantine failure.

The topology in IRCFT approach is linear and the mobile agent travels through servers consequential. We can change topology from linear to non-linear and study the results.



(1) Spawn a replica agent

(2) Log entry (log arrive)

(3) Send message to server S_{i-1}

(4) After computation, requests a commitment from the manager agent

(5) Receive the address from the manager agent

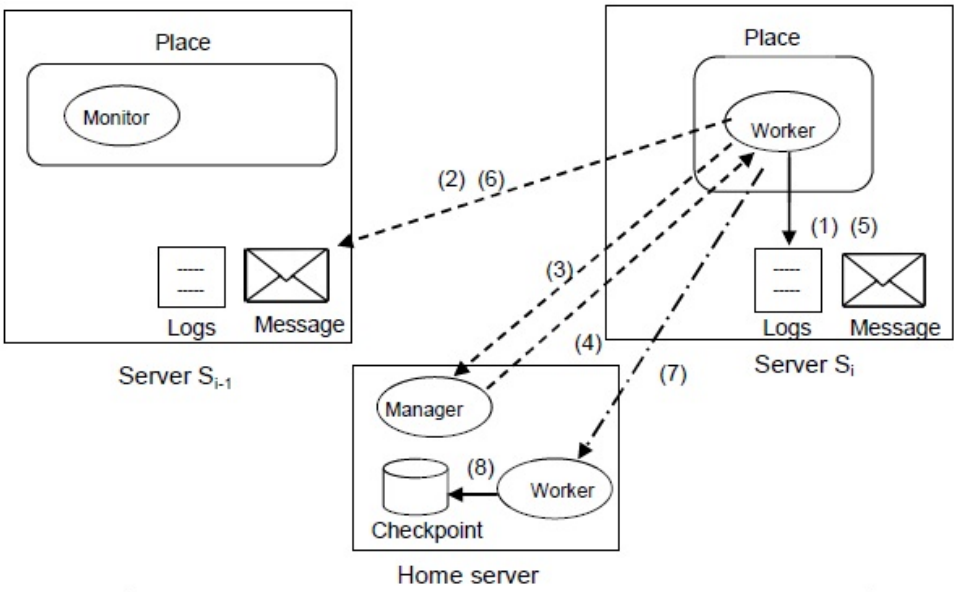
(6) Update the replica and the previous replica

(7) Log entry (log leave)

(8) Send message to server S_{i-1}

(9) Spawn a monitor agent

Figure 6.1: New IRCFT approach -1-



(1) Log entry (log arrive)

(2) Send message to server S_{i-1}

(3) After computation, requests a commit from the manager agent

(4) Receive the commit from the manager agent

(5) Log entry (log leave)

(6) Send message to server S_{i-1}

(7) Migrate to the home server

(8) Checkpoint the data

Figure 6.2: New IRCFT approach -2-

Bibliography

[1] Aglet. http://aglets.sourceforge.net/, [Accessed December 2, 2015].

[2] Fipa-os tutorial. http://fipa-os.sourceforge.net/tutorials.htm, [Accessed December 2, 2015].

[3] Jade - java agent development framework. http://jade.tilab.com/, [Accessed December 2, 2015].

[4] Madkit. http://www.madkit.org, [Accessed December 2, 2015].

[5] Naplet: Mobile agents for network-centric pervasive applications. http://www.ece.eng.wayne.edu/~czxu/software/naplet.html, [Accessed December 2, 2015].

[6] Voyager: Middleware for enterprise mobile applications. http://www.recursionsw.com/voyager-intro/, [Accessed December 2, 2015].

[7] S. Aarti, J. Dimple, and A. K. Sharma. Agent development toolkits. *International Journal of Advancements in Technology*, 2, 2011.

[8] D.B. Andrew and J.N. Bruce. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, 1984.

[9] F. Baumann, J. Hohl, K. Rothermel, M. Strasser, and W. Theilmann. Security and reliability in concordia. *MOLE: A Mobile Agent System, John Wiley & Sons, SoftwarePractice & Experience*, 32, 2002.

[10] L. Bettini and R. D. Nicola. Translating strong mobility into weak mobility. *Springer, In Proceedings of the 5th International Conference on Mobile Agents*, 2240:182–197, 2001.

[11] R. Boutaba and J. Xiao. State-of-the-art of mobile agent computing: Security, fault tolerance, and transaction support. *Springer, IFIP The International Federation for Information Processing*, 92, 2002.

[12] P. Braun, J. Eismann, C. Erfurth, and W.R. Rossak. Tracy - a prototype of an architected middleware to support mobile agents. *In Proceedings of the 81th IEEE Conference and Workshop on the Engineering of Computer Based Systems (ECBS), Washington D.C., U.S.A.*, 2001.

[13] P. Braun and W. Rossak. Mobile agents: Basic concepts, mobility models, and the tracy toolkit. *Morgan Kaufmann*, 2005.

[14] J. Briot, Z. Guessoum, S. Charpentier, S. Aknine, O. Marin, and P. Sens. Dynamic adaptation of replication strategies for reliable agents. *In Proceedings of 2nd Symposium on Adaptive Agents and Multi-Agent Systems*, pages 10–19, 2002.

[15] C. Bumer and M. Thomas. Grasshopper - a mobile agent platform for active telecommunication. *In Proceedings IATA of the Third International Workshop on Intelligent Agents for Telecommunication Applications*, 1999.

[16] C. Bumer and M. Thomas. Madkit: A generic multi-agent platform. *ACM*, 2000.

[17] J. Chen, H. Shi, C. Chen, Z. Hong, and P. Zhong. An efficient forward and backward fault-tolerant mobile agent system. *Eighth International Conference on Intelligent Systems Design and Applications*, pages 61–66, 2008.

[18] W. Dake, C.P. Leguizamo, and K. Mori. Mobile agent fault tolerance in autonomous decentralized database systems. *IEEE, In Proceedings of the Autonomous Decentralized System on The 2nd International Workshop*, 2002.

[19] M. Eid, H. Artail, A. Kayssi, and A. Chehab. Trends in mobile agent applications. *Journal of Research and Practice in Information Technology*, 37(4):347–353, 2005.

[20] E. N. Elnozahy, L. Alvisi, Y. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM, ACM Computing Surveys*, 34(3):375–408, 2002.

[21] A. Fedoruk and R. Deters. Improving fault-tolerance by replicating agents. *ACM, In Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pages 737–744, 2002.

[22] R. S. Gray, G. Kotz, D. Cybenko, and D. Rus. Mobile agents: Motivations and state-of-the-art systems. *Dartmouth College, Technical Report: TR2000-365*, 2000.

[23] R. Han and R. Kaur. Fault tolerance approach in mobile agents for information retrieval applications using check points. *International Journal of Computer Science & Communication Networks*, 2(3):347–353, 2012.

[24] S. Ilarri, R. Trillo, and E. Mena. Springs: A scalable platform for highly mobile agents in distributed computing environments. *in Fourth International WoWMoM 2006 Workshop on Mobile Distributed Computing (MDC06), Niagara FallsBuffalo, New York, USA. IEEE Computer Society*, 2006.

[25] T. Illmann, F. Kargl, M. Weber, and T. Krger. Migration of mobile agents in java: Problems, classication and solutions. *In MAMA 2000: Proc. of the International ICSC Symp. on MultiAgents and Mobile Agents in Virtual Organizations and E-Commerce, Wollogong, Australia*, 2000.

[26] W. Jim. The foundation for the electronic marketplace. *General Magic Inc*, 1994.

[27] G. Jin, B. Ahn, and K. D. Lee. A fault-tolerant protocol for mobile agent. *Springer, In Proceedings of International Conference on Computational Science and Its Applications*, pages 993–1001, 2004.

[28] D. Johansen, R. V. Renesse, F. B. Schneider, N. P. Sudmann, and K. Jacobsen. A tacoma retrospective. *In Proceeding IATA of the Third International Workshop on Intelligent Agents for Telecommunication Applications*, 32(6), 2002.

[29] O. Kachirski and R. Guha. Intrusion detection using mobile agents in wireless ad hoc networks. *IEEE, In Proceedings of the IEEE Workshop on Knowledge Media Networking*, pages 153–158, 2002.

[30] D. Kotz and R. S. Gray. Mobile agents and the future of the internet. *ACM, Communications of the ACM*, 33(3):88–89, 1999.

[31] D. B. Lange and M. Oshima. Seven good reasons for mobile agents. *ACM, Communications of the ACM*, 42(3):88–89, 1999.

[32] M.R. Lyu and T.Y. Wong. A progressive fault tolerant mechanism in mobile agent systems. *Proc. 7th World Multiconference on Systemics, Cybernetics and Informatics*, 10:299–306, 2003.

[33] P. Marikkannu, J.J. Adri Jovin, and T. Purusothaman. Fault-tolerant adaptive mobile agent system using dynamic role based access control. *International Journal of Computer Application*, 20(2), April 2011.

[34] S. Mishra and P. Xie. Interagent communication and synchronization support in the daagent mobile agent-based computing system. *IEEE, IEEE Transactions on Parallel and Distributed Systems*, 14(3), 2003.

[35] D. Mitrovic, Z. Budimac, M. Ivanovic, and M. Vidakovic. Improving fault tolerance of distributed multi-agent systems with mobile network-management agents. *In Proc. of the International Multiconference on Computer Science and Information Technology*, 2010.

[36] T. Osman, W. Wagealla, and A. Bargiela. An approach to rollback recovery of collaborating mobile agents. *IEEE, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 34(1):48–57, 2004.

[37] K. Park. A fault-tolerant mobile agent model in replicated secure services. *Springer, In Proceedings of International Conference Computational Science and Its Applications*, 3043:500–509, 2004.

[38] T. Park, I. Byun, H. Kim, and H. Y. Yeom. The performance of checkpointing and replication schemes for fault tolerant mobile agent systems. *IEEE Computer Society, In Proceedings of 21th IEEE Symposium on Reliable Distributed Systems*, 2002.

[39] B. Peter, E. Christian, R. Wilhelm, and S. Friedrich. An introduction to the tracy mobile agent system. *Jena University, Computer Science Department, Germany*, 2000.

[40] S. Pleisch. State-of-the-art of mobile agent computing: Security, fault tolerance, and transaction support. *Research Report, IBM Research, Z. R. Lab. Switzerland*, 1999.

[41] S. Pleisch and A. Schiper. Fatomas-a fault-tolerant mobile agent system based on the agent-dependent approach. *IEEE, In Proceedings of the 2001 International Conference on Dependable Systems and Networks*, pages 215–224, 2001.

[42] L. L. Pullum. Software fault tolerance techniques and implementation. *Artech House*, 2001.

[43] W. Qu and H. Shen. Analysis of mobile agents fault-tolerant behavior. *IEEE/WIC/ACM, Proceedings of International Conference on Intelligent Agent Technology*, pages 377–380, 2004.

[44] W. Qu, H. Shen, and X Defago. A survey of mobile agent-based fault-tolerant technology. *IEEE, In Proceedings of the Sixth International Conference on Parallel and Distributed Computing Applications and Technologies*, pages 446–450, 2005.

[45] Rahul and K. Ramandeep. Novel dynamic shadow approach for fault tolerance in mobile agent systems. *In Proceedings of 6th International Conference on Signal Processing Communication Systems, Publication IEEE Conference*, 2012.

[46] S. Rajwinder and D. Mayank. Antecedence graph approach to checkpointing for fault tolerance in mobile agent systems. *In Proceedings of IEEE Transactions On Computers*, 62(2), 2013.

[47] K. Ramandeep, K.C. Rama, and S. Rajwinder. Integrated mechanism to prevent agent blocking in secure mobile agent platform system. *In Proceedings of International Conference on Advances in Computer Engineering*, 2010.

[48] A. Rostami, H. Rashidi, and M. S. zahraie. Fault tolerance mobile agent system using witness agent in 2- dimensional mesh network. *In Proceedings of International Journal of Computer Science Issues*, 7(5), 2010.

[49] G. Serugendo and A. Romanovsky. esigning fault-tolerant mobile system. *Springer, International Workshop on Scientific Engineering for Distributed Java Applications*, pages 185–201, 2003.

[50] L. M. Silva, V. Batista, and J. G. Silva. Fault-tolerant execution of mobile agents. *IEEE, In Proceedings International Conference on Dependable Systems and Networks*, pages 135–143, 2000.

[51] L. M. Silva, G. Soares, P. Martins, V. Batista, and L. Santos. The performance of mobile agent platforms. *IEEE, In Proceedings of First International Symposium on Third International Symposium on Mobile Agents*, pages 270–271, 1999.

[52] M. Srilekha, E. Abdeliah, and J. William. Anchor toolkit: A secure mobile agent system. *In Proceedings of Mobile Agents 99 Conference*, 1999.

[53] S. Summiya, K. Ijaz, U. Manzoor, and A. A. Shahid. A fault tolerant infrastructure for mobile agents. *IEEE, In Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce*, pages 235–235, 2006.

[54] S. Suzanne and D. Amal. Integrated replication-checkpoint fault tolerance approach of mobile agents "ircft". *ACIT, In Proceedings of the 16th International Conference on Information Technology*, December,2015.

[55] S. Suzanne and D. Amal. Integrated replication-checkpoint fault tolerance approach of mobile agents "ircft". *IAJIT, The International Arab Journal of Information Technology*, 13(1A), February,2016.

[56] S. Suzanne and D. Amal. Ircft implementation and results. *PICCIT, In Proceedings of the 4th Palestinian International Conference on Computer and Information Technology*, October, 2015.

[57] T. Walsh, N. Paciorek, and D. Wong. Security and reliability in concordia. *IEEE, In Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences*, 7, 1998.

[58] H. K. Yeom, H. Y. T. Park, and H. Park. The cost of checkpointing, logging and recovery for the mobile agent systems. *In Proceedings of Pacific Rim International Symposium on Dependable Computing*, pages 45–48, 2002.

[59] L. Zeghache and N. Badache. Optimistic replication approach for transactional mobile agent fault tolerance. *In Proceedings of 11th CIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2010.