FAULT TOLERENACE OF MOBILE AGENTS: CENTRALIZED APPROACH

ENG. AMAL MOH'D AL-DWEIK
Palestine Polytechnic University
Cairo University
amal.dweik@gmail.com

A.PROF. IMANE ALY SAROIT ISMAIL
Cairo University
iasi63@hotmail.com

ABSTRACT

Faults tolerance of Mobile Agents is one of the most promising researches which deal with the tolerating of faults in distributed systems and internetworking environments. JADE, Java Agent Development Environment platform was used to implement and test the system. JADE uses a very preliminary method in tolerating faults which just tolerating faults of the main container, the basic placeholder for agents in the platform. In this paper, we will state the implementation of the centralized approach which was described mathematically in [1]. This paper concentrates on the modeling of the centralized approach. The implementation of this approach and the experiments were done to test it. Results were analyzed after implementing the approach and compared with the built in tool already used in JADE. The paper stated that this approach could improve the performance and reliability of JADE when it is used with the built in tool implemented by the JADE board.

Keywords: *Distributed systems, Mobile agent, Fault tolerance, Replication.*

1. INTRODUCTION

Mobile Agent is a type of software agent, with the feature of *autonomy*, *social ability*, *learning*, and most important, *mobility*. When the term *mobile agent* is used, it refers to a process that can transport its state from one environment to another, with its data intact, and still being able to perform appropriately in the new environment [2]. The conceptual diagram of the mobile agent is shown in figure1. Multi-agent systems are often, in fact, distributed cooperative applications on the Internet. As a result, open multi-agent systems need to cope with the characteristics of the Internet, e.g., dynamic availability of computational resources, latency, and diversity of services [3]. Agents may migrate at any point in their execution, fully preserving their state, and may exchange messages with other agents within the same platform or with other agents in other platforms. By ensuring the continuity of computations in spite of failure occurrences, fault-tolerant mechanisms are essential in large-scale environments [3].

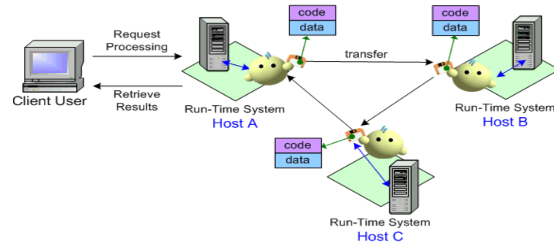


Figure 1: Mobile Agent: Conceptual Diagram [2]

Various approaches to fault tolerance support exist. They have different strengths and weaknesses, and address different environments. Because of this variety, it is often difficult for the application programmer to choose the approach best suited to an application [1]. This paper is organized as follows: after this introduction, a brief theoretical background for the fault tolerance of mobile agents is introduced in section 2. JADE and its features were mentioned in section 3. Section 4 presents the system model and its specification. The experiments and results are introduced in section 5. The conclusion and future work is mentioned at the end of the paper in section 6.

2. THEORITICAL BACKGROUND.

When moving through a large and unreliable network such as the Internet, mobile agents may crash, or even the placeholders for those agents may crash. Generally, in any distributed system or network, a source needs to process information in different n hosts. The source sends out a MA to visit these n hosts either by knowing these hosts or dynamically associate this MA to these hosts. After the MA visits its list of hosts and gets the results, then it either brings the results back or sends them to any other host as it is specified. During the processing on n hosts, the host can crash. The agent itself could suffer from crash as well and the same is for the platform. [1]

3. JADE

JADE, Java Agent Development Environment, is an agent platform that implements the basic services and infrastructure of a distributed multi-agent application such as: agent life-cycle and agent mobility, white & yellow-page services, peer-to-peer message transport & parsing, agent security, scheduling of multiple agent tasks, set of graphical tools to support monitoring, logging, and

debugging, and it enables interoperability through FIPA compliance [5]. JADE supports "hard mobility", i.e. mobility of status and code. The status of an agent can stop its execution on the local container, move to a remote container (likely on a different host) and restart its execution there from the exact point where it was interrupted. While if the code of the moving agent is not available on the destination container it is automatically retrieved on demand. In order to be able to move, an agent must be Serializable. Mobility can be either: self-initiated through the doMove() method of the Agent class, or forced by the AMS (following a request from another agent [5].

3.1 JADE Communication Model

The communication model is based on asynchronous message passing. Each agent has a sort of mailbox where messages for that agent are inserted. When a message is put in the mailbox the agent is notified. However it will be up to him to decide if and when to read the message and how to react to it [4].

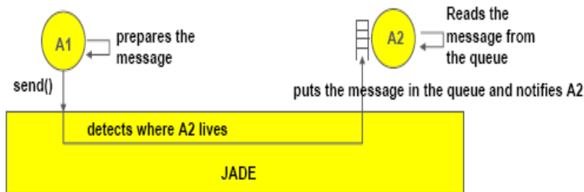


Figure 2: communication model

3.2 JADE Architecture

A JADE-based application is composed of a collection of active components called Agents. Each agent has a unique name and is a peer since he can communicate in bidirectional way with all other agents. Each agent lives in a container (that provides its run time) and can migrate within the platform. One container plays the role of main (where AMS, DF live). The main-container can be replicated via replication service as shown in figure 3.

3.3 JADE Fault Tolerance Mechanism

To keep JADE fully operational even in the event of a failure of the Main-Container, support for Main Container replication has been introduced. Using this support, it is possible to start any number of Main Container nodes (a "*master*" main container actually holding the AMS and a number of "*backup*" main containers), which will arrange themselves in a logical ring so that whenever one of them fails; the others will notice and act accordingly [6]. Ordinary containers will then be able to connect to the platform through any of the active Main Container nodes; the different copies will evolve together using cross-notification. As the following figure shows, without Main Container replication, JADE platform has a star topology. End enabling Main Container replication turns the topology into a ring of stars [6].

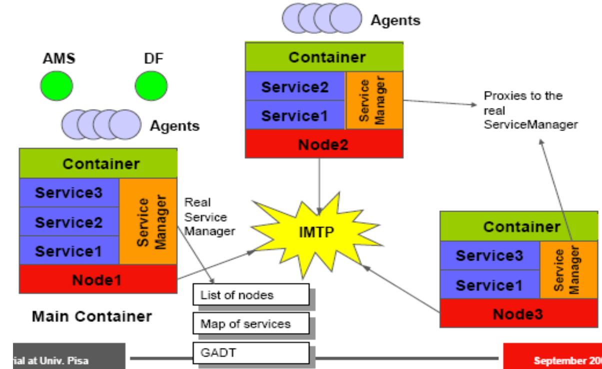
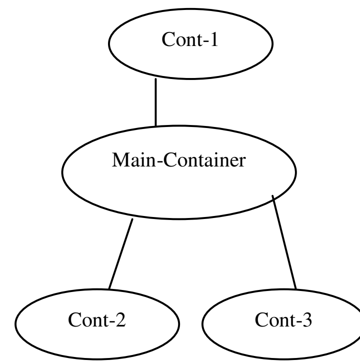
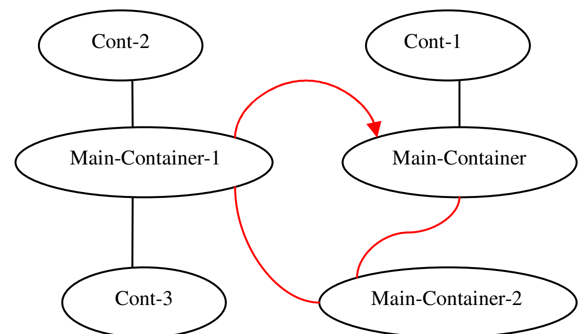


Figure 3: JADE Architecture Model



(a)



(b)

Figure 4 (a, b): Jade fault tolerance tools

Figure 4 also shows that peripheral containers can be arbitrarily spread among the available Main Container nodes. Any single peripheral container is connected to exactly one node and in absence of failures it is completely unaware of all the other copies. When a Main Container node fails, there will generally be some orphaned peripheral containers Main Container nodes present in the platform [6]. JADE supports two policies in distributing the Main Container node list to peripheral containers. *A first option* is to activate the Address-Notification service on all Main Container nodes and on the peripheral

containers. This service will detect additions and removals to the Main Container nodes ring and update the address lists of all peripheral containers involved. *A second option* is to pass the address list to a starting peripheral container with -smaddrs command line argument. This approach avoids generating notification traffic towards peripheral containers but assumes a fixed list of Main Container nodes, which is known beforehand [6].4. SYSTEM MODEL AND SPECIFICATION Rather than trying to anticipate all potential kinds of faults, our approach makes use of the technique used in the Jade and adds its own adds-on tool to improve performance and reliability as a basic means of recovery from failure accidents.

4.1 System Definition

The system objective is to build an adds-on tool on the top of JADE platform to increase reliability and achieve tolerating of faults in the case of agent and container crash. Herein in this paper we will talk about the agent crash. The container crash will not be handled in this paper. We have three main proposed approaches: the windowing approach, the winnowing-shot approach, and the centralized approach [1]. In addition to the built-in tool exist in JADE we've implemented and tested the centralized approach. The system definition, assumptions, and constraints are all discussed in our previous paper outlined in [1]. We'll quickly review the assumptions and constraints, and then go thoroughly to the diagrams and the algorithm proposed.

4.1.1 Assumptions

Here are the basic assumptions that are used throughout the entire system model.

$\forall$ JS, Fail (AP_i) ➔ Fail(JS)

$\forall$ JS and $MA_j^k \in C_i^j$ and MA_1^{co} = Clone (MA_j^k, Loc) then P (Loose (MA_l^{co})) = 0.

Where: "co" refers to the agent replica, AP_i: agent platform i, JS: JADE System, MA_j^k : MobileAgent k at j container, C_i^j: Container j at I platform.

4.1.2 Constraints

$\forall$ Jade system, JS, $\exists$ AP_i:: (q P_{lc}, s P_r); where s, q $\geq$0.

$\forall$ JS, $\exists$!{ $C_m^l \in AP_i$ and $C_i^j \in AP_m$ }; where m $\neq$i and l $\neq$ j.

No more than one container can have the same name in the same system.

$\forall$ JS, $\exists$! { $MC_m^l \in AP_i$ and $MC_i^j \in AP_m$ } ; where i $\neq$m and {l, j } = 1. So, no more than one main container in the same system could exist whether it is in the same local platform or remote platform.

$\forall$ JS, $\exists$! {$MA_l^m \in C_i^j$ and $MA_l^k \in C_i^l$ }; where j $\neq$l and k $\neq$ m.

With no more than one MA has the same name in the same system.

$\forall$ JS : (MC_i (z MA_1^l) , n C_i^j (m MA_j^k)) ; where n$\geq$0. and z, m $\geq$0 and k $\neq$ 1 .

4.2 Centralized Approach

4.2.1 Model

This approach concentrates on the crash of agents and not for that of the containers. The time the mobile agent is created, a list of services is registered for that MA. As the centralized approach suggests, a copy of this created MA is created at the Main-Container, MC, with all the original agent services registered. This means:

Create(MA_j^k,Scv_k) ➔ *Reg(MA_j^k,Scv_k)* ➔ *Clone(MA_j^k,1).*

In the centralized approach, we can use only one replica for each MA. Any MA that finishes its execution will be forced to update its unique replica existing in the main container. So, when migrating this agent, Mig (MA_j^k, Loc), we will get MA_{Loc}^k (Scv_k) and Update_Agent (MA_j^k); where:

Update_Agent(MA_j^k)=Kill(MA_1^{co})+Clone(MA_j^k,1).

When this MA is crashed for any reason, then the takedown() method is invoked where the centralized code is executed to launch the most up-to-dated version of this crashed agent, Update_Agent(MA_j^k), from the Main-container. This process could be explained to follow these steps: the replica of this MA, suppose it is MA_j^k, is migrated to the last container that it works on, C_i^{j-1}. And a new updated version of this agent is created at the main container as before. So, the model could be in the failure case as follows:

Fail(MA_j^k) = 1? Recover (MA_j^k).
Recover (MA_j^k):
Begin
Mig (MA_1^k, j+1)
Update_Agent(MA_j^k)
Clone (MA_1^k,j)
End

The MA then completes its journey until reaching its final destination or is back to its original source; where it is then be killed and vanished normally.

4.2.2 Diagrams

The diagram of this approach is shown in figure 5.

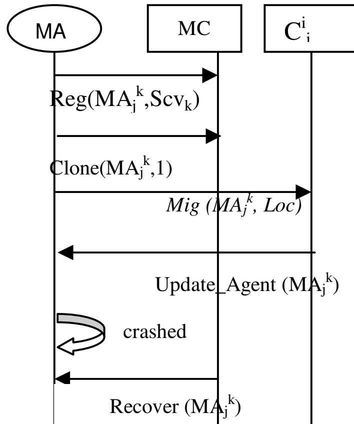


Figure 5: the diagram of the centralized approach

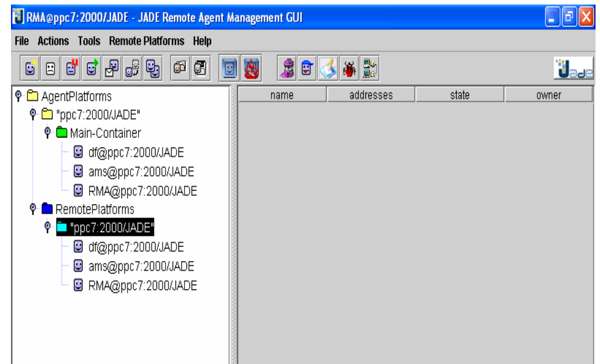
5. EXPERIMENTS AND RESULTS

5.1 Jade fault tolerance tools

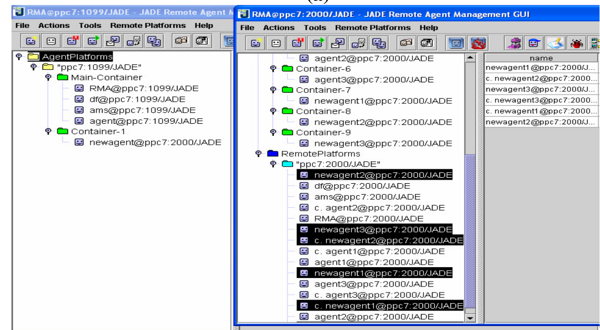
Jade is a single point of failure platform. The only fault tolerance mechanism that is used in jade is by making the main container fault free. This is accomplished by making some kind of replication with another replicated main container “Backup main container” in another host connected to the same platform. This secondary or replicated one is automatically updated periodically by the system. When the main container is crashed, then this will be initiated as a substitution for the main one. This backup container contains all the information needed. Therefore, the system will not suffer from the fault of the main container and it will continue functioning properly. As Jade just makes the platform tolerate faults using the backup for the main container, then if this backup is crashed then there is no such replication and so no data is saved for the agents running or deployed in the platform. The system enables the cascaded backups for the main container. In JADE, there is no proper saving for the whole agent snapshot is properly accomplished; even they mentioned that the persistency adds-on tool could help in that.

5.1.1Results and problems

We have tested the JADE tools and ensure that it accomplishes its objectives as shown in figure5(a,b). As stated in 5.1.1, the JADE tool is just for tolerating the crash of the main container by creating backup containers. To tolerate faults of the backup itself, you have to make backup container for it. So, it needs cascading backups in order to solve the problem of the main container. No fault tolerance procedure or mechanism is done for agents or any other container except the main container.



(a)



(b)

Figure 5 (a and b): JADE backup tool experiment

5.2 Centralized approach

The algorithm is fully implemented and tested according to the suggested approach stated in details at 4.2.

5.2.1 Implementation

Herein, the failure detection service is based on the "alive" status for the running agent. As the agent, or specifically the MA, is running, then its "alive" status is activated. If it is changed, then the takedown () and doDelete() methods are invoked, some built-in exceptions are raised when such situations are occurred. As JADE assumes full and reliable connectivity between containers; we can implement some kind of keep-alive mechanism at the agent level or (even better) at the kernel services level exploiting the ping() remote method of the jade.core.Node class (each container sits on top of a Node). The code is added in the beforeMove(), afterMove(), and takedown() methods in the mobile agent code. The mobile agent code is inherited from the agent code after adding the specific mobility services and ontologies. The entire code is compatible with the FIPA specification and legally could be created and appended to the JADE code since JADE is open source software under the terms of the LGPL (Lesser General Public License Version 2).

5.2.2 Results

After implementing the approach, it becomes possible to tolerate failure of mobile agents. We tested the

whole approach under the constraints that the main container is always on and doesn't fail under any circumstances unless the platform itself is down, which is correct in the specification of JADE simulator. The mobile agent itinerary was dynamic, i.e., no previous knowledge about the mobile agent journey before its creation. This dynamicity adds more flexibility to the system and so we have achieved better reliability and flexibility. As long as the data that mobile agent is using is serializable, then the Mobile agent could deal with it easily and could be persistent and tolerate faults as well as the mobile agent code and status themselves.

The approach itself is tested several times and an abnormal termination for several mobile agents is forced several times to ensure that it is working under the different circumstances. Replicating every agent in an application comprising up to millions of agents is likely to undermine the overall determination of the result, particularly in terms of performance. However, JADE scalability allows hundreds of agents to be created over each container in the platform. So, as long as we are not overwhelming this limit, then it is acceptable. At any given time, some agents can be lost without significant incidence on the rest of the computation whereas some others are crucial to it [3]. We studied such case, and we can conclude that when the mobile agent is killed abnormally or crashed immediately after its launch, then it will be cheaper to reissue this agent instead of applying the centralized approach with the replication and all other steps which will be considered, in this case, as overhead rather than increasing the reliability and performance.

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

As we mentioned in [1], there is no official benchmark about mobility in JADE. FIPA Standard specifications do not include anything about mobility too. Mobile agents seem to be well-suited for network management applications in heterogeneous environments; Internet is the best suited example for such environment.

Replication is the most efficient reliability technique in the presence of failures. However, software replication is a costly solution as it implies the multiplication of resource-consuming components as well as the consistency maintenance between replicas. It might be argued that increased resource consumption is not really a problem in a large-scale environment where resource availability is virtually infinite. For JADE its scalability features implies the use of hundreds of agents in each container.

As the centralized approach implies just one replica for each mobile agent is needed to be fault free, then it seems that the number of replicas are acceptable. Not all mobile agents are needed to be fault free; some mobile agents are needed just to go to an additional container, where it could be better and cheaper to re issue such mobile agent in stead to use the centralized approach. Finally, this centralized approach is simple and needs less resource replication compared to other approaches with the same performance. The main problem is that the replication highly depends on the continuation of the main container. The JADE fault tolerance tool is needed to be used in addition to the centralized approach to achieve better reliability and performance.

6.2 Ongoing and Future Work

We are working on the implementation of the other two approaches: the windowing and the windowing – shot are. Since the three approaches share some common first steps, then this makes it easier to compare and implement. The work still not completed. So, no comparisons or recommendations could be announced right now. But if we just make a quick study and go thoroughly the other two approaches, we will get some advantages for them over this approach, since in the case of windowing and windowing-shot approach we have second level of replication so we can get more reliability. As long as we are not approaching the maximum number of agents within the platform, then no problems in getting such replicas whether they are at the other containers or at the main container. But in the case of scalability or even when JADE containers are overloaded, this centralized approach seems to be better.

7. REFERENCES

[1] Amal Al Dweik and Imane Ismail. "Fault Tolerance of Mobile Agents". ACIT 2005. Amman. 2005.

[2] "Wikipedia, an international Web-based free-content encyclopedia project". URL: http://en.wikipedia.org/wiki/Mobile%5Fagent

[3] Benno Overeinder, Frances Olivier, and Marin Brazier1, "Fault Tolerance in Scalable Agent Support Systems: Integrating DARX in the AgentScape Framework". This research is supported by the NLnet Foundation, http://www.nlnet.nl/, 2002

[4] "JADE: the new kernel and last developments". Giovanni Caire, JADE Board Technical Leader, Pisa 2004

[5] JADE Tutorial for beginners, Part 2 - USING JADE. Fabio Bellifemine, TILAB, The Hague, 12/10/04

[6] JADE ADMINISTRATOR’S GUIDE. Fabio Bellifemine, Giovanni Caire, Tiziana Trucco. Giovanni Rimassa. Copyright (C) 2000 CSELT S.p.A., 2001 TILab S.p.A., 2002 TILab S.p.A.