5th SASTech
Iran |Mashhad
May 12 - 17 | 2011
5th Symposium on Advances in Science & Technology

Computing of Fault Tolerant Mobile Agents in Distributed Systems

H.Hamidi
Department of Electronics Engineering
Islamic Azad University -Doroud Branch
hamidi@iau-doroud.ac.ir
Paper Reference Number: 1812-472
Name of the Presenter: H.Hamidi

Abstract

The reliable execution of a mobile agent is a very important design issue to build a mobile agent system and many fault-tolerant schemes have been proposed. Hence, in this paper, we present evaluation of the performance of the fault-tolerant schemes for the mobile agent environment. Our evaluation focuses on the checkpointing schemes and deals with the cooperating agents. We derive the Fault-Tolerant approach for Mobile Agents (FANTOMAS) design which offers a user transparent fault tolerance that can be activated on request, according to the needs of the task, also discuss how transactional agent with types of commitment constraints can commit. Furthermore this paper proposes a solution for effective agent deployment using dynamic agent domains.

Key words: Checkpointing; Fault Tolerant; Mobile Agent; Network Management; FANTOMAS;

1. INTRODUCTION

The client/server computing paradigm is today's most prominent paradigm in distributed computing. In this computing paradigm, the server is defined as a computational entity that provides some services. The client requests the execution of these services by interacting with the server. Having executed the service, the server delivers the result back to the client. The server therefore provides the knowledge of how to handle the request as well as the required resources. The computing paradigm of mobile code generalizes this concept by performing changes along two orthogonal axes:

1. Where is the know-how of the service located?
2. Who provides the computational resources?

Depending on the choices made on the client and server sides, the following variants of mobile code computing paradigms, illustrated in Table 1, can be identified (Fuggetta, Picco, & Vigna,1998):

In the Remote Evaluation (REV) paradigm, component A sends instructions specifying how to perform a service to component B (represented by *code* in Table 1). These instructions can, for instance, be expressed in Java byte code. Component B then executes the request using its own resources, and returns the result, if any, to A. Java Servers are an example of remote evaluation (Coward, 2001).

	before the invocation		after the invocation	
	machine 1	machine 2	machine 1	machine 2
Client/Server	A	code,resource,B	A	code,resource,B
Remote Evaluation	code,A	resource,B	A	*code*,resource,B
Code on Demand	resource,A	code,B	*code*,resource, A	B
Mobile Agent	code,A	resource, B		*code*,resource,*A*, B

Table 1: Different variants of the mobile code computing paradigm (Fuggetta, Picco, & Vigna, 1998).Code or computational entities transported between machines are indicated by italics. Component A accesses the services provided by component B.

In the Code on Demand (CoD) paradigm, the resources are collocated with component A , But A lacks the knowledge of how to access and process these resources in order to obtain the desired result. Rather, it gets this information from component B (represented by *code* in Table 1). As soon as A has the necessary know-how (i.e., has downloaded the code from B), it can start executing. Java applets5 fall under this variant of the mobile code paradigm. The mobile agent computing paradigm is an extension of the REV paradigm. Whereas the latter focuses primarily on the transfer of code, the mobile agent paradigm involves the mobility of an entire computational entity, along with its code, the state, and potentially the resources required to perform the task. As developer-transparent capturing and transfer of the execution state (i.e., runtime state, program counter, and frame stacks, if applicable) requires global state models as well as functions to externalize and internalize the agent state, only few systems support this *strong mobility* scheme. In particular, Java-based mobile agent platforms are generally unsuitable for this approach, because it is not possible to access an agent's execution stack without modifying the Java Virtual Machine. Most systems thus settle for the *weak mobility* scheme where only the data state is transferred along with the code. Although it does not implicitly transport the execution state of the agent, the developer can explicitly store the execution state of the agent in its member attributes. The values of these member attributes are transported to the next machine. The responsibility for handling the execution state of an agent thereby resides with the developer. In contrary to REV, mobile agents can move to a sequence of machines, i.e., can make multiple hops.

A mobile agent is a software program which migrates from a site to another site to perform tasks assigned by a user. For the mobile agent system to support the agents in various application areas, the issues regarding the reliable agent execution, as well as the compatibility between two different agent systems or the secure agent migration, have been considered. Some of the proposed schemes are either replicating the agents (Hamidi & Mohammadi , 2005) or checkpointing the agents (Park,Byun,Kim,&Yeom,2002;Pleisch&Schiper,2001;). For a single agent environment without considering inter-agent communication, the performance of the replication scheme .In this paper, we focus on the checkpoint-based schemes for the cooperating agents. For the performance analysis, a refined simulator has been developed and based on the simulation results, the performance of the schemes are discussed in the context of the influence of failures and system parameters. The suggestion for the efficient checkpointing is also made. In the area of mobile agents, only little work can be found relating to fault tolerance. Most of them refer to special agent systems or cover only some special aspects relating to mobile agents, e. g. the communication subsystem. Nevertheless, most people working with mobile agents consider fault tolerance to be an important issue (Izatt, Chan,&Brecht,1999; Shiraishi,Enokido,&Takzawa,2003).

2. MOBILE AGENTAND FAULT MODEL

We assume an asynchronous distributed system, i.e., there are no bounds on transmission delays of messages or on relative process speeds. An example of an asynchronous system is the Internet. Processes communicate via message passing over a fully connected network.

A mobile agent executes on a sequence of machines, where a *place* pi ($0 < i < n$) provides the logical execution environment for the agent (Pleisch&Schiper,2001,2003).Logically, a mobile agent executes in a sequence of stage actions (see Figure 1). Each stage action *Sai* consists of potentially multiple

operations op_0, op_1 ,... .Agent ai ($0 < i < n$) at the corresponding stage *Si* represents the agent *a* that has executed the stage actions on places pj ($j < i$) and is about to execute on place *pi*. The execution of ai on place pi results in a new internal state of the agent as well as potentially a new state of the place (if the operations of an agent have side effects). We denote the resulting agent ai+1. Place *pi* forwards ai_+1 to pi+1 (for $i < n$). Any hardware and software component in a computing environment is potentially subject to failures.

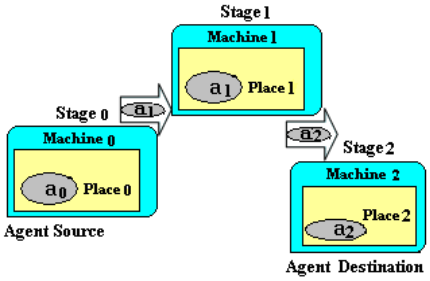

Fig.1: Model of a mobile agent execution with three stages

This paper addresses the following failures: crash of an agent, a place, or a machine. Clearly, the crash of a machine causes any place and any agent running on this machine to crash as well (Figure 2.d). A crashing place causes the crash of any agent on this place, but this generally does not affect the machine (Figure 2.c). Similarly, a place and the machine survive the crash of an agent (Figure 2.b). We do not consider programming errors in the code of the agent or the place as relevant failures in this sense.

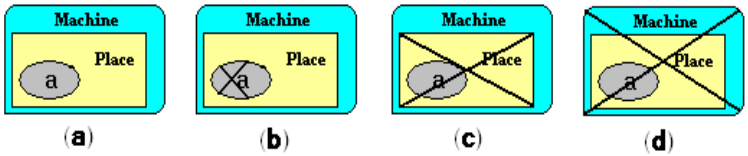

Fig.2. Failure model for mobile agents

3. THE FANTOMAS CONCEPT

From these considerations, we choose independent checkpointing with receiver based logging as base for our fault tolerance approach for mobile agents. Adhering to the agent paradigm and exploiting the already available facilities of the mobile agent respect. The agent environment, an agent is used as the stable storage for the checkpointed state and the message log. For each mobile agent (called *user agent* (UA) in the following), for that fault tolerance is enabled, a *logger agent*(LA) is created. A user agent and its logger agent form an agent pair (Figure 3).The logger agent does not participate actively in the application's computation, and thus needs only a small fraction of the available CPU capacity. It follows the user agent at a certain, non-zero, distance on its migration path through the system. They must never reside on the same node, so that not a single fault destroys both of them. User and logger agent monitor each other, and if a fault is detected by one of them, it can rebuild the other one from its local information. The creation of the agent pair is readily derived from the already existing migration facilities. To create a logger agent, the user agent serializes its state in the same way as for a migration, and sends it to a remote agent place. There, a new agent is created from this data. Different from migration, the new agent does not start the application module that was sent with the state information, and the user agent continues normal execution. Further, the communication unit of the agent is exchanged against a version that first forwards each incoming message to the logger agent before delivering it locally.

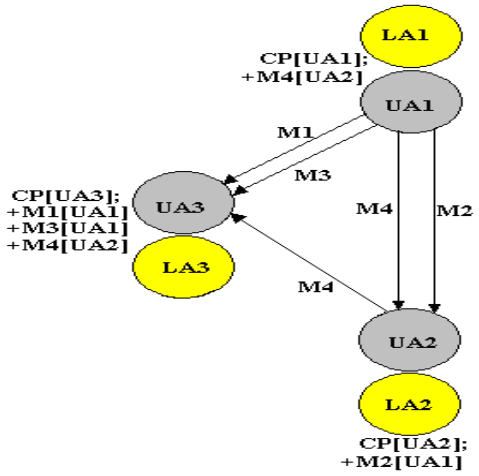

Fig. 3: Example for an application with three user and logger agents with checkpoints (CP) and messages (M) (Petri & Grewe,1999)

4. BLOCKING PROBABILITY

The node needs to collect a majority of votes during processing to be able to commit the transaction of a stage. Therefore, a transaction can only be committed if more than half of the stage nodes are available. This fact can be used to give a (simple) metric for the availability *As* of a stage which is the probability that a majority of stage nodes is available so that an agent can finish a step and proceed with the next step. Let *n* be the number of the nodes of a stage and *p* be the availability of an individual node (i.e. the probability that the node is available). Then the probability that exactly *m* out of these *n* nodes are available can be calculated using the binomial probability function

$$f(n, m) = \binom{n}{m} p^m (1-p)^{(n-m)} \qquad (1)$$

(Hughes, Grawoig, 1971).The availability *As*(*n*, *p*) of a stage *S* can then be calculated by

$$A_s(n, p) = \sum_{i=\left\lceil \frac{n+1}{2} \right\rceil}^{n} \binom{n}{i} p^i (1-p)^{(n-i)} \qquad (2)$$

The blocking probability is defined to be the probability that the agent is blocked in the stage. It is calculated by

$$Bs(n, p) = 1 - As(n, p) \qquad (3)$$

The relative blocking probability *Br*(*n*, *p*) is calculated by

$$Br(n, p) = Bs(n, p) / Bs(1, p) \qquad (4)$$

Where a relative blocking probability of $Br(n, p)=0.4$ means for example that the probability of an agent blocking in a stage with n nodes (node availability p) is only 40% of the probability of an agent blocking on one node with availability p. Table 2 and Figure 4 show the relative blocking probability Br depending on the availability p of a node and the number n of stage nodes. It shows that an odd number of nodes bigger or equal to 3 reduce the relative blocking probability dramatically.

n	p		
	0.75	0.9	0.99
1	100%	100%	100%
2	175%	190%	199%
3	62%	28%	3%
4	105%	52%	6%
5	41%	9%	~0%
6	68%	16%	~0%
7	28%	3%	~0%

Table 2: Relative blocking probability of a stage

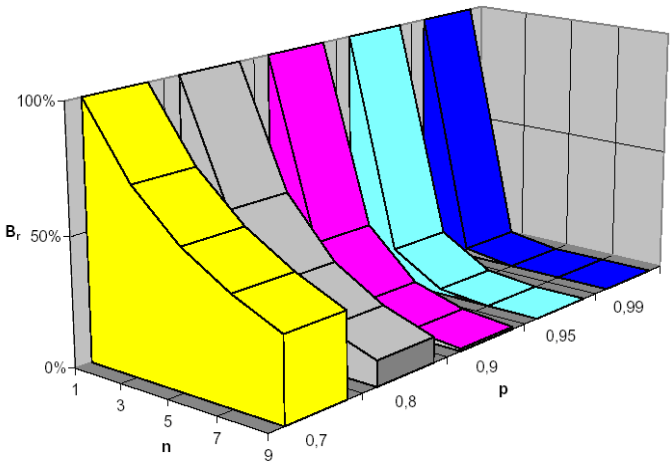

Fig.4: Relative blocking probability

5. EVALUATION AND SIMULATION RESULT

We evaluate transactional agents in terms of access time compared with client-server model. In the evaluation, three object servers S_1 , S_2 and S_3 are realized in Oracle which are in sun workstation (SPARC 900MHz x 2) and a pair of Windows PCs(Pentium4 2.0GHz x 2 and pentium4 1.0 GHz), respectively. The object servers are interconnected with 100base LAN. JDBC classes are already loaded in each object server .An application program A manipulates table objects by issuing select and update to some number of object servers at the highest isolation level, i.e. select for update in Oracle. The application program A is realized in Java on a client computer. In the transactional agent model, the application A is realized in Aglets .The computation of Aglets is composed of moving, class loading, manipulation of objects, creation of clone, and commitment steps. In the client-server model, there are computation steps of program initialization, class loading to client, manipulation of objects, and two-phase commitment.

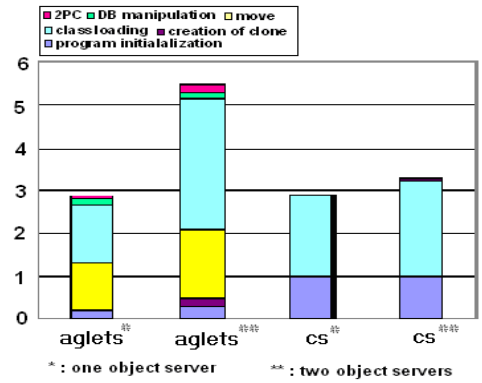

Fig.5: Access time

Figure 5 shows how long it takes to perform each step for two cases, one for manipulating one object server and another for manipulating two object servers, in client-server (cs) and transactional agent (TA) models. In the transactional agent model, Aglets classes are loaded to each object server before an agent is performed**.** Since Java classes are loaded to only client in the client-server model, the time for loading the classes is constant for any number of object servers. As shown in Figure 5, time to manipulate objects in a transactional agent is shorter than the client-server model because there is no communication between agent and object server. The time to load Aglets classes in each object server is about half of the total computation time in the transactional agent. The transactional agent faster manipulates objects than the client-server model. On the other hand, Aglets classes have to be loaded in the transactional agent. It takes about two seconds to load Aglets classes.

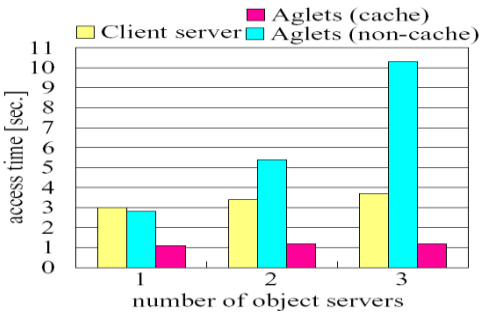

Fig. 6: Client server vs. agent

Figure 6 shows the access time for number of object servers. The *non-cache Aglets* shows that Aglets classes are not loaded when an agent *A* arrives at an object server. Here, the agent can be performed after Aglets classes are loaded. On the other hand, the *cache Aglets* means that an agent manipulates objects in each object server where Aglets classes are already loaded, i.e. the agent comes to the object server after other agents have visited on the object server. As shown in Figure 6, the client-server model is faster than the transactional agent. However, the transactional agent is faster than the client-server model if object servers are frequently manipulated, i.e. cashed Aglets classes are a priori loaded. A simulator was designed to evaluate the algorithm. The system was tested in several simulated network conditions and numerous parameters were introduced to control the behavior of the agents. We also investigated the dynamic functioning of the algorithm. Comparing to the previous case the parameter configuration has a larger effect on the behavior of the system. The most vital parameter was the frequency of the trading process and the pre-defined critical workload values. Fig. 7 shows the

number of agents on the network. The optimal agent population is calculated by dividing the workload on the whole network with the optimal workload of the agent.

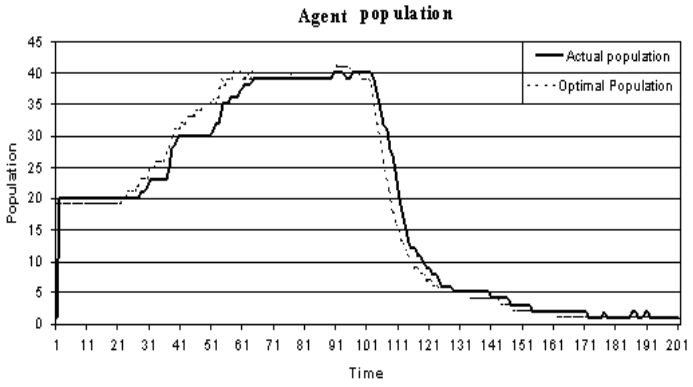

Fig. 7: The size of the agent population under changing network conditions

Simulation results show that choosing correct agent parameters the workload of the agents is within a ten percent environment of the predefined visiting frequency on a stable network. In a simulated network overload the population dynamically grows to meet the increased requirements and smoothly returns back to normal when the congestion is over. To measure the performance of FANTOMAS our test consists of sequentially sending a number of agents that increment the value of the counter at each stage of the execution. Each agent starts at the agent source and returns to the agent source, which allows us to measure its round–trip time. Between two agents, the places are not restarted. Consequently, the first agent needs considerably longer for its execution, as all classes need to be loaded into the cache of the virtual machines. Consecutive agents benefit from already cached classes and thus execute much faster. We do not consider the first agent execution in our measurement results. For a fair comparison, we used the same approach for the single agent case (no replication). Clearly, this is a simplification, as the class files do not need to be transported with the agent. Remote class loading adds additional costs because the classes have to be transported with the agent and then loaded into the virtual machine. However, once the classes are loaded in the class loader, other agents can take advantage of them and do not need to load these classes again.

6. CONCLUSION

This paper discussed a mobile agent model for processing transactions which manipulate object servers. An agent first moves to an object server and then manipulates objects. We showed the evaluation of the mobile agent-based transaction systems for applications. If Aglets classes are a priori loaded, the transactional agents can manipulate object servers faster than the client-server model. Also, we have presented the experimental evaluation of the performance of the fault-tolerant schemes for the mobile agent environment. From the experimental results, the system environment suitable for each of the fault tolerant schemes has been discussed and the stability of the schemes has been also discussed. General possibilities for achieving fault tolerance in such cases were regarded, and their respective advantages and disadvantages for mobile agent environments, and the intended parallel and distributed application scenarios shown. This leads to an approach based on warm standby and receiver side message logging. In the paper dynamically changing agent domains were used to provide flexible, adaptive and robust operation. If Aglets classes are *a priori* loaded, the transactional agents can manipulate object servers faster than the client-server model.

REFERENCES

Coward, D. (2001, August). Java Servlet Specification – Version 2.3, Sun Microsystems.

Fuggetta, A. & Picco, G. P., & Vigna, G.(1998). Understanding code mobility. IEEE Transactions on Software Engineering, 24(5).(pp. 342–361).

Hamidi, H. & Mohammadi,K.(2005, March).Modeling and Evaluation of Fault Tolerant Mobile Agents in Distributed Systems .In Proceeding of the 2th IEEE Conference on Wireless & Optical Communications Networks (WOCN2005)(pp.91-95).

Hamidi, H. & Mohammadi,K.(2006,January-March). Modeling Fault Tolerant and Secure Mobile Agent Execution in Distributed Systems. International Journal of Intelligent Information Technologies (IJIIT 2(1)). pp.21-36.

Hamidi, H. & Vafaei, A**.** (2008, May). Evaluation of Security and Fault-Tolerance in Mobile Agents. In Proceeding of the 5th IEEE Conference on Wireless & Optical Communications Networks (WOCN2008) (pp.1-5).

Hughes, A. & Grawoig, D.(1971).Statistics: A Foundation for Analysis. Addison-Wesley Publishing Company.

Izatt,M., Chan, P., & Brecht, T. (1999, June). Agents: Towards an Environment for Parallel, Distributed and Mobile Java Applications. In *Proceeding of the* 1999 ACM Conference on Java Grande,(pp. 15-24).

Park, T. , Byun, I.,Kim,H., & Yeom, H.Y. (2002). The Performance of Checkpointing and Replication Schemes for Fault Tolerant Mobile Agent Systems. In *Proceeding of the 21th IEEE Symp. on Reliable Distributed Systems*.

Petri, S. & Grewe, C.(1999,September).A Fault-Tolerant Approach for Mobile Agents. In Dependable Computing – EDCC-3, Third European Dependable Computing Conference, Fast Abstracts. Czech Technical University in Prague.

Pleisch ,S. & Schiper, A.(2000).Modeling Fault-Tolerant Mobile Agent Execution as a Sequence of Agree Problems. In Proceeding of the 19th IEEE Symposium on Reliable Distributed Systems(pp. 11-20).

Pleisch ,S. & Schiper, A.(2001,July).FATOMAS - A Fault-Tolerant Mobile Agent System Based on the Agent-Dependent Approach. In Proceeding of the 2001 International Conference on Dependable Systems and networks (pp.215-224).

Pleisch ,S. & Schiper, A.(2003). Fault-Tolerant Mobile Agent Execution. IEEE Transactions on *Computers,52(2).*

Shiraishi,M.,Enokido,T.&Takzawa,M.(2003). Fault-Tolerant Mobile Agents in Distributed Systems. In Proceedings of the Ninth IEEE Workshop on Future Trends of Distributed Computer (FTDCS'03)(pp. 11-20).