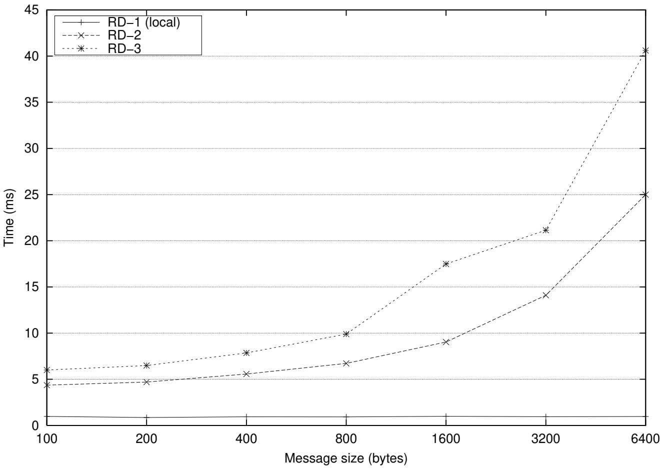
*

***

*   *



RD−1 (local)
RD−2
RD−3
Time (ms)
Message size (bytes)
0
5
10
15
20
25
30
35
40
45
100
200
400
800
1600
3200
6400

***

*

*

*** **

** **

Dynamic and Adaptive Replication for Large-Scale Reliable Multi-Agent Systems

Z. Guessoum,
LIP6, Université Pierre et
Marie Curie (Paris 6)
BP 169, 4 place Jussieu,
F-75252 Paris Cedex 5
Paris, France
Zahia.Guessoum@lip6.fr

J.-P. Briot
LIP6, Université Pierre et
Marie Curie (Paris 6)
BP 169, 4 place Jussieu,
F-75252 Paris Cedex 5
Paris, France
Jean-Pierre.Briot@lip6.fr

S. Charpentier
LIP6, Université Pierre et
Marie Curie (Paris 6)
BP 169, 4 place Jussieu,
F-75252 Paris Cedex 5
Paris, France
charpentier@mangoosta.net

ABSTRACT

In order to make large-scale multi-agent systems reliable, we propose an adaptive application of replication strategies. Critical agents are replicated to avoid failures. As criticality of agents may evolve during the course of computation and problem solving, we need to dynamically and automatically adapt the number of replicas of agents, in order to maximize their reliability and availability based on available resources. We are studying an approach and mechanisms for evaluating the criticality of a given agent (based on application-level semantic information, e.g. messages intention, and also system-level statistical information, e.g., communication load) and for deciding what strategy to apply (e.g., active replication, passive) and how to parameterize it (e.g., number of replicas).

Keywords

Multi-agent, Fault-tolerance, Observation, Organization, Roles.

1. INTRODUCTION

A multi-agent system is a set of autonomous and interactive entities called agents ([1]). Recent real-life applications (e.g., intensive care monitoring, air traffic control and process control) are often distributed at large scale and must run continuously without any interruption. As distributed systems, multi-agent systems are exposed to possibility of failure of their hardware and/or software components. The failure of one component can often evolve into the failure of the whole system. To make these large-scale systems reliable, an obvious solution is the introduction of redundancy: duplication (replication) of the critical components.

Replication mechanisms have been successfully applied for various distributed applications ([8]), e.g. data-bases. But in most cases, replication is decided by the programmer and applied statically, before the application starts. This works fine because the criticality of components (e.g., main servers) may be well identified and remains stable during the application session.

Opposite to that, in the case of dynamic and adaptive multi-agent applications, the criticality of agents may evolve dynamically during the course of computation. Moreover, the available resources are often limited. Thus, simultaneous replication of all the components of a large-scale system is not feasible. Our idea is thus to **automatically** and **dynamically** apply replication mechanisms **where** (to which agents) and **when** it is most needed. In this extended abstract we will introduce our approach to this objective and the software architecture to help at building reliable multi-agent systems.

2. REQUIREMENTS AND TECHNIQUES FOR FAULT-TOLERANCE

2.1 A First and Simple Example

We consider the example of a distributed multi-agent system that helps at scheduling meetings. Each user has a personal assistant agent which manages his calendar. This agent interacts with:

- the user to receive his meeting requests and associated information (a title, a description, possible dates, participants, priority, etc.),
- the other agents of the system to schedule a meeting.

If the assistant agent of one important participant (initiator or prime participant) in a meeting fails (e.g., his machine crashes), this may disorganize the whole process. As the application is very dynamic - new meeting negotiations start and complete dynamically and simultaneously - decision for replication should be done automatically and dynamically.

2.2 Principles of Replication

Replication of data and/or computation is an effective way to achieve fault tolerance in distributed systems. A replicated software component is defined as a software component that possesses a representation on two or more hosts ([7]). There are two main types of replication protocols:

- active replication, in which all replicas process concurrently all input messages,
- passive replication, in which only one of the replicas processes all input messages and periodically transmits its current state to the other replicas in order to maintain consistency.

Active replication strategies provide fast recovery but lead to a high overhead. If the degree of replication is n, the n replicas are activated simultaneously to produce one result.

Passive replication minimizes processor utilization by activating redundant replicas only in case of failures. That is: if the active replica is found to be faulty, a new replica is elected among the set of passive ones and the execution is restarted from the last saved state. This technique requires less CPU resources than the active approach but it needs a checkpoint management which remains expensive in processing time and space.

The choice of the most suitable strategy is directly dependent on the environment context, especially the failure rate, and the application requirements in terms of recovery delay and overhead.

2.3 Limits of Current Replication Techniques

Many toolkits (e.g., [7] and [15]) include replication facilities to build reliable applications. However, most of them are not quite flexible enough to implement adaptive replication mechanism. For example, the chosen strategy cannot be modified in the course of the computation.

Therefore we designed a specific and novel framework for replication, named DarX (see details in Section 4), which allows dynamic replication and dynamic adaptation of the replication policy (e.g., passive to active, changing the number of replicas). Moreover, DarX has been designed to easily integrate various agent architectures.

3. TOWARDS DYNAMIC REPLICATION AND ADAPTIVE CONTROL

Several solutions have been proposed to replicate distributed systems. These solutions are often used by the designer to replicate the system components before run time. The number of replicas and the replication strategy are explicitly and statically defined by the designer before run time. However, these solutions are not suitable to multi-agent system. The solution we propose is mainly characterized by dynamic replication and adaptive control.

3.1 Dynamic Replication

In most multi-agent applications, the environment context is very dynamic. So, the choice of the replication strategy of each component, which relies on a part of this environment, must be determined dynamically and adapted to the environment changes.

Moreover, a multi-agent system component which can be very critical at a moment can loose its criticality later. If we consider the replication cost which is very high, the number of replicas of these components must be therefore dynamically updated.

Thus, the solution we propose allows to dynamically adapt the number of replicas and the replication strategy. This solution is provided by the framework DarX (see Section 4).

3.2 Adaptive Control

DarX provides the needed adaptive mechanisms to replicate agents and to modify the replication strategy. Meanwhile, we cannot always replicate all the agents of the system because the available resources are usually limited. In the given example (Section 2.1), we can consider more than 100 assistant agents and resources that do not allow to duplicate more than 60 agents. The problem therefore is to determine the most critical agents and then the needed number of replicas of these agents.

We distinguish two cases: 1) the agent's criticality is static and 2) the agent's criticality is dynamic. In the first case, multi-agent systems have static organization structures, static behaviors of agents, and a small number of agents. Critical agents can be therefore identified by the designer and can be replicated by the programmer before run time.

In the second case, multi-agent systems may have dynamic organization structures, dynamic behaviors of agents, and a large number of agents. So, the agents criticality cannot be determined before run time. The agent criticality can be therefore based on these dynamic organizational structures. The problem is how to determine dynamically these structures to evaluate the agent criticality? Thus, we propose a new approach for observing the domain agents and evaluating dynamically their criticality. This approach is based on two kinds of information: semantic-level information and system-level information.

4. DARX: A FRAMEWORK FOR DYNAMIC REPLICATION

DarX is a framework to design reliable distributed applications which include a set of distributed communicating entities (agents). Each agent can be replicated an unlimited number of times and with different replication strategies (passive and active). Note that, we are working on the integration of other replication strategies in DarX, including quorum-based strategies [2]. The number of replicas may be adapted dynamically. Also, and this is a novel feature, the replication strategy is reified such as one may dynamically change the replication strategy.

4.1 DarX Architecture

DarX includes group membership management to dynamically add or remove replicas. It also provides atomic and ordered multi-cast for the replication groups' internal communication. For portability and compatibility issues, DarX is implemented in Java.

4.2 Agent Replication

A replication group is an opaque entity underlying every application agent. The number of replicas and the internal strategy of a specific agent are totally hidden to the other application agents. Each replication group has exactly one leader which communicates with the other agents. The leader also checks the liveness of each replica and is responsible for reliable broadcasting. In case of failure of a leader, a new one is automatically elected among the set of remaining replicas.

DarX provides global naming. Each replicated agent has a global name which is independent of the current location

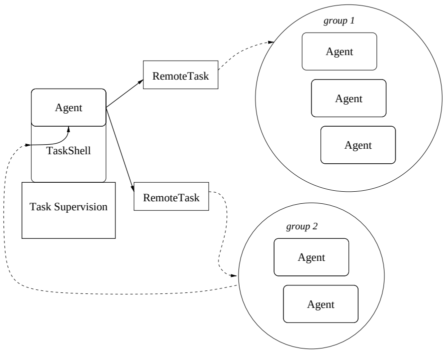


Figure 1: DarX application architecture

of its replicas. The underlying system allows to handle the agent's execution and communication. Each agent is itself wrapped into a TaskShell (Figure 1), which acts as a replication group manager and is responsible for delivering received messages to all the members of the replication group, thus preserving the transparency for the supported application. Input messages are intercepted by the TaskShell, enabling message caching. Hence all messages get to be processed in the same order within a replication group.

An agent can communicate with a remote agent, unregarding whether it is a single agent or a replication group, by using a local proxy implemented by the RemoteTask interface. Each RemoteTask references a distinct remote entity considered as the leader of its replication group. The reliability features are thus brought to agents by an instance of a DarX server (DarxServer) running on every location. Each DarxServer implements the required replication services, backed up by a common global naming/location service.

4.3 Measurements

Our first measurements of DarX are very promising. We evaluated several costs and made comparisons with other systems (see [11]).

In this extended abstract, we just show the cost of sending a message to a replication group using the active replication strategy. Figure 2 presents three configurations with different replication degrees. In the RD-1 configuration, the task is local and not replicated. In the RD-2 (resp. RD-3) configuration, there is one (resp. two) replica(s); the leader being on the sending host and the other replica(s) residing on one (or two) distinct remote host(s).

5. ADAPTIVE CONTROL OF REPLICATION

5.1 Hypothesis and principles

We want some automatic mechanism for generality reasons. But in order to be efficient, we also need some prior input from the designer of the application. This designer can choose among several approaches of replication: static and dynamic.

In the proposed dynamic approach, the agent criticality relies on two kinds of information:

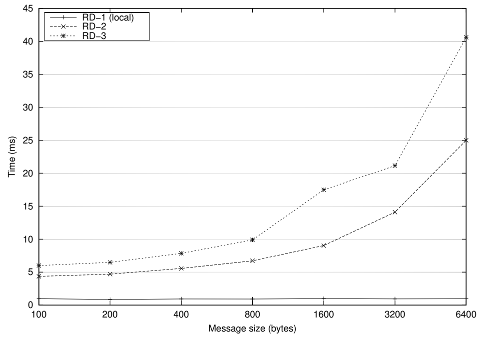


Figure 2: Communication cost as a function of the replication degree

- System-level information. It will be based on standard measurements (communication load, processing time...). We are currently evaluating their significance to measure the activity of an agent.
- Semantic-level information.

Several aspects may be considered (importance of agents, independence of agents, importance of messages...). We decided to use the concept of role, because it captures the importance of an agent in an organization, and his dependencies to other agents.

Note that our approach is generic and that it is not related to a specific interaction language or application domain. Also agents can be either reactive or cognitive. We just suppose that they communicate with some agent communication language such as ACL [6].

5.2 Example

The application designer will manually evaluate criticality of the roles, corresponding to their "importance" in the organization and in the computation.

In the example introduced in Section 2.1, we are considering two roles: Initiator and Participant. Their respective weights will be set by the application designer to respectively 0.7 and 0.3 (see Table 1).

Table 1: Examples of roles and their weights

Roles	Weights
Initiator	0.7
Participant	0.3

5.3 Architecture

In order to track the dynamical adoption of roles by agents, we propose a role recognition method. Our approach is based on the observation of the agent execution and their interactions to recognize the roles of each agent and to evaluate his processing activity. This is used to dynamically compute the criticality of an agent.

We consider two cases. In the first case, each agent displays explicitly his roles or interaction protocols. The roles of each agent are thus easily deduced from interaction events. In the second case, agents do not display their roles nor their interaction protocols. The agent roles are deduced from the interaction events by the role analysis module.

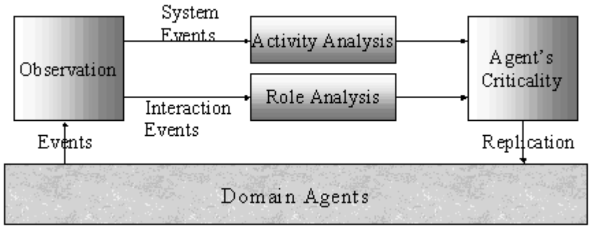


Figure 3: General architecture for replication control

In order to collect the data, we associate an observation module to each DarxServer on each machine (see Section 4.1). It will collect events (provided by DarxServer). The basic architecture controlling the replication of agents is shown in Figure 3.

The analysis of events (system events and interaction events) provides two kinds of information: the roles and the degree of activity of each agent. This information is then processed by the agent's criticality module. The latter relies on a table T (an example is given in Table 1) that defines the weights of roles. This table is initialized by the application designer.

The criticality of the agent A_i which fulfills the roles r_{i1} to r_{im} may be computed as follows (see [3] for details):

$$w_i = (a_1 * \sum_{j=1,m} T[r_{ij}] + a_2 * aw_i)/(a1 + a2) \qquad (1)$$

Where: a_1 and a_2 are the weights given to the two kinds of parameters (roles and degree of activity). They are introduced by the designer. aw_i is the degree of activity of A_i.

For each Agent A_i, his criticality w_i is used to compute the number of his replicas. A_i may be then replicated according to:

- w_i: his criticality,
- W: the sum of the criticalities of all agents,
- rm: the minimum number of replicas which is introduced by the designer,
- Rm: the available resources which define the maximum number of possible simultaneous replicas.

The number of replicas nb_i of A_i can be determined as follows:

$$nb_i = rounded(rm + w_i * Rm/W) \qquad (2)$$

6. EXPERIMENTS

We made some preliminary experiments using the scenario of agents scheduling their meetings, as introduced in Section 2.1.

Agents take randomly roles of Initiator, choose Participants for scheduling meetings or remain inactive (without any role). Several meetings are scheduled simultaneously. The number of critical agents (which can be either Initiator or Participant) is 60% of the number of agents.

As the distributed observation module implementation has not been completed yet, we have run these preliminary experiments on a single machine. In order to simulate the presence of faults, we implemented a failure simulator randomly stopping the thread of an agent (chosen randomly). The number of the introduced faults was set equal to the number of agents. We repeated several times the experiments with a variable number of resources (number of replicas that can be used). From these first experiments, we found that the number of resources should be at least equal to the number of critical agents.

We are currently working on more experiments and measurements in order to better evaluate our adaptive control architecture and to compare it to other control methods (including random replication).

7. RELATED WORK

Several approaches address the multi-faced problem of fault tolerance in multi-agent systems. These works can be classified in two main categories. A first category focuses especially on the reliability of an agent within a multi-agent system. This approach handles the serious problems of communication, interaction and coordination of agents with the other agents of the system. The second category addresses the difficulties of making reliable an agent, particularly a mobile agent, which is more exposed to security problems [13]. This second category is beyond the scope of this paper.

Within the family of reactive multi-agent systems, some systems offer high redundancy. A good example is a system based on the metaphor of ant nests. Unfortunately: 1) we cannot design any application in term of such reactive multi-agent systems. Basically we do not have yet a good methodology. 2) we cannot apply such simple redundancy scheme onto more cognitive multi-agent systems as this would cause inconsistencies between copies of a single agent.

Some work [4] offers dynamic cloning of specific agents in multi-agent systems. But their motivation is different, the objective is to improve the availability of an agent if it is too congested. The agents considered seem to have only functional tasks (with no changing state) and fault-tolerance aspects are not considered.

[10] introduces sentinels to protect the agents from some undesirable states. Sentinels represent the control structure of their multi-agent system. They need to build models of each agent to perform functionalities and monitor communications in order to react to faults. Each sentinel is associated by the designer to one functionality of the multi-agent sys-

tem. This sentinel handles the different agents which interact to achieve the functionality. The analysis of his believes on these agents enables the sentinel to detect a fault when it occurs. Adding sentinels to multi-agent systems seems to be a good approach, however the sentinels themselves represent failure points for the multi-agent system.

[5] propose to use proxies to make transparent the use of agent replication, i.e. enabling the replicas of an agent to act as a same entity regarding the other agents. The proxy manages the state of the replicas. All the external and internal communications of the group are redirected to the proxy. However this increases the workload of the proxy which is a quasi central entity. To make it reliable, they propose to build a hierarchy of proxies for each group of replicas. They point out the specific problems of read/write consistency, resource locking also discussed in [14].

In distributed computing, many toolkits include replication facilities to build reliable application. However, many of products are not enough flexible to implement an adapted replication. MetaXa [12] implements in Java active and passive replication in a flexible way. Authors extended Java with a reactive metalevel architecture. Like in DarX, the replication is transparent. However, MetaXa relies on a modified Java interpreter. GARF [7] realizes fault-tolerant Smalltalk machines using active replication. Similar to MetaXa, GARX uses a metalevel and provides different replication strategies. But, it does not provide adaptive mechanism to apply these strategies.

8. CONCLUSION

Large-scale multi-agent systems are often distributed and must run without any interruption. To make these systems reliable, we proposed a new approach to evaluate dynamically the criticality of agents. This approach is based on the concepts of roles and degree of activity. The agent criticality is then used to replicate agents in order to maximize their reliability and availability based on available resources.

To validate the proposed approach, we realized a fault-tolerant framework (DarX) and we used a multi-agent framework (DIMA [9]) to implement multi-agent systems. The integration of DarX with the multi-agent platform DIMA provides a generic fault-tolerant multi-agent platform. In order to validate this fault-tolerant multi-agent platform, two small applications have been developed (meetings scheduling and crisis management system). They are intended at evaluating our model and architecture viability. The obtained results are interesting and promising. However, more experiments with real-life applications are needed to validate the proposed approach.

9. ADDITIONAL AUTHORS

S. Aknine, (LIP6, email: `Samir.Aknine@lip6.fr`), O. Marin (LIP6, email: `Olivier.Marin@lip6.fr` and P. Sens (LIP6, email: `Pierre.Sens@lip6.fr`).

10. REFERENCES

[1] N. A. Avouris and L. Gasser. *Distributed Artificial Intelligence: Theory and Praxis*, chapter Object-Oriented Concurrent Programming and Distributed Artificial Intelligence, pages 81–108. Kluwer Academic Publisher, 1992.

[2] F. Belkouch, M. Bui, and L. Chen. Self-stabilizing quorum systems for reliable document access in fully distributed information systems. *Studies in Informatics and Control*, 7(4):311–326, 1998.

[3] Jean-Pierre Briot, Zahia Guessoum, Sébastien Charpentier, Samir Aknine, Olivier Marin, and Pierre Sens. Dynamic and adaptive control of replication for reliable agents. In *Adaptive Agents and Multi-Agent Systems*, pages 10–19, London, 2002. AISB.

[4] K. Decker, K. Sycara, and M. Williamson. Cloning for intelligent adaptive information agents. In *ATAL'97*, LNAI, pages 63–75. Springer Verlag, 1997.

[5] A. Fedoruk and R. Deters. Improving fault-tolerance by replicating agents. In *AAMAS2002*, Boulogna, Italy, 2002.

[6] FIPA. Specification. part 2, agent communication language, foundation for intelligent physical agents, geneva, switzerland. http://www.cselt.stet.it/ufv/leonardo/fipa/index.htm, 1997.

[7] R. Guerraoui, B. Garbinato, and K. Mazouni. Lessons from designing and implementing garf. In *Proceedings Objects Oriented Parallel and Distributed Computatio*, volume LNCS 791, pages 238–256, Nottingham, 1989.

[8] R. Guerraoui and A. Schiper. Software-based replication for fault tolerance. *IEEE Computer*, 30(4):68–74, April 1997.

[9] Z. Guessoum and J.-P. Briot. From active objects to autonomous agents. *IEEE Concurrency*, 7(3):68–76, 1999.

[10] S. Hagg. A sentinel approach to fault handling in multi-agent systems. In C. Zhang and D. Lukose, editors, *Multi-Agent Systems, Methodologies and Applications*, number 1286 in LNCS, pages 190–195. Springer Verlag, 1997.

[11] O. Marin, P. Sens, J.-P. Briot, and Z. Guessoum. Towards adaptive fault-tolerance for distributed multi-agent systems. In *ERSADS'2001*, pages 195–201, 2001.

[12] M.Golm. Metaxa and the future of reflection. In *OOPSLA -Workshop on Reflective Programming in C++ and Java*, pages 238–256. Springer Verlag, 1998.

[13] F. De Assis Silva and R. Popescu-Zeletin. An approach for providing mobile agent fault tolerance. In S. N. Maheshwari, editor, *Second International Workshop on Mobile Agents*, number 1477 in LNCS, pages 14–25. Springer Verlag, 1998.

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

[15] R. van Renesse, K. Birman, and S. Maffeis. Horus: A flexible group communication syste. *CACM*, 39(4):76–83, 1996.