Adaptive Replication in Fault-Tolerant Multi-Agent Systems

Ductor Sylvain
LIP6
Pierre et Marie Curie University
Paris, France
sylvain.ductor@lip6.fr

Zahia Guessoum
LIP6, MODECO-CReSTIC
Pierre et Marie Curie University
IUT de Reims
Paris - Reims, France
zahia.guessoum@lip6.fr

Mikal Ziane
LIP6
Pierre et Marie Curie University
Paris Descartes University
Paris, France
mikal.ziane@lip6.fr

Abstract—Distributed cooperative applications are now open, very dynamic and large scale. Thus, they are increasingly designed as Multi-Agent Systems (MAS). This evolution creates new challenges to the traditional approaches of fault-tolerance that mostly relies on centralized and offline solution. In this paper, we propose an adaptive replication-based preventive approach designed for large-scale MAS. Our solution stands on a negotiation protocol to dynamically and transparently adapt the agent replication strategy (number of replicas and their location) to the system’s state. This protocol provides a decentralized resource allocation solution which uses local decisions to guarantee the global MAS reliability.

Keywords—Automated Negotiation; Resource Allocation; Replication; Large Scale;

I. INTRODUCTION

Multi-agent systems (MAS) are particularly attractive for creating software that operates in distributed and open environments, such as the Internet. Being both decentralized and self-organized, these systems suffer from all the problems associated with building traditional distributed systems as well as the additional difficulties that arise from having flexible and sophisticated interactions among a large number of autonomous and adaptive agents. Fault tolerance is one of the most important issues. A fault-tolerant infrastructure that could detect and adapt itself to the failures provides continuity of processing and is thus crucial to large-scale MAS.

There are two major approaches to fault-tolerance: corrective and preventive [10]. In this paper, we focus on the preventive approach. We consider crash failures, it is to say a host that suddenly stops to work or is disconnected from the network. To deal with this type of failures, preventive solutions have been developed (algorithms, architectures and platforms), mostly based on the concept of replication (e.g. [1][3][4][7][10]). These projects have provided replication facilities to build reliable distributed applications. However, in those approaches, the set of components, the number of replicas of each component and their locations are chosen by the designer before the application starts.

Kraus et al. [5] proposed a relevant probabilistic notion of survivability and study the allocation of extra resources (replicas) for agents. Zhang et al. [9] extended the previous work and proposed two exact and five heuristic algorithms to compute the probability of survival of a given deployment. Even if agents are distributed, the proposed algorithms are centralized. Thus, application domains typically only involve a small number of components with non-adaptive behavior. These centralized algorithms are therefore not suited to large-scale and adaptive MASs where resource allocation needs to be applied in real-time to adapt the MAS to the evolution of its environment.

Replication has been proved to be an effective way to achieve fault-tolerance in distributed systems [3]. However massive replication is not always possible since in real environments the MAS performance would be affected. To the best of our knowledge, resource-aware adaptive replication has not been addressed yet in the scope of large scale and open MAS. We introduce in this paper a negotiation protocol that manages the resources and improves the reliability of such MAS. This protocol is transparent and can be tuned to have limited impact on the performance of the MAS.

This paper is organized as follows. Section 2 describes the related work. Section 3 introduces the problem. The negotiation protocol and its decisions processes are described in Section 4 and they are theoretically analyzed in Section 5. Finally, Section 6 reports on the implementation and the experiments.

II. PROBLEM STATEMENT AND FOUNDATIONS OF OUR APPROACH

A replication-based fault-tolerant MAS consists of a set of agents, Agents, that run and can be replicated on a set of hosts, Hosts. In this paper we call agent the group of replicas (see [6] for details on replication management of a replication group under failures).

When a host fails, all the hosted replicas are lost. Agent a fails if and only if all its replicas are lost. Let \( \text{Rep}_a(t) \) be the set of hosts where Agent a is replicated at instant \( t \), and let \( P(\text{Rep}_a(t)) \) be the probability that those hosts fail simultaneously. The availability of agent a is defined as follows:

\[
D_a(t) = 1 - P(\text{Rep}_a(t))
\]
The first purpose of the replication mechanism is to minimize the failure probability of each agent. However, since some agents may have more important roles than others, they should have a higher availability. Informally, the criticality of an agent, \( W_a(t) \), is a measure of the impact of its failure on the whole MAS (see [4][10][7] for details).

An allocation \( A \) is modeled by a bipartite graph:

\[
A = (\text{Agents}, \text{Hosts}, \mathcal{R})
\]

where each vertex \((a, h) \in \mathcal{R}\) represents the replica of Agent \( a \) on Host \( h \). Agents are endowed with the reliability provided by their replicas (see Section II-A) and hosts are endowed with their current loads (see Section II-C). Agent replication is a dynamic process that initializes and updates this graph: it decides which agent to replicate, how many replicas each agent is granted and where to locate them.

A. Agent Reliability

**Definition 1 (Agent reliability):** The reliability of Agent \( a \), \( \mathcal{R}_a(A) \) is an aggregation of its criticality and its availability on a given allocation \( A \).

\[
\mathcal{R}_a(A) = \frac{D_a(A)}{W_a}
\]

Informally, Agent availability should be proportional to its criticality. Since each agent has a role in the application its criticality is at least \( w_{min} \) (the actual parameter value is defined by the designed, e.g. \( w_{min} = 0.1 \)). This interval of possible reliabilities variations is divided into 3 subsets (fragile, thrifty and wasteful) and associate thus a symbolic reliability status to each agent. The definition of the subsets is dynamically adapted according to a dynamic global estimation of the reliability of the agents.

B. MAS Reliability

We aim at keeping out as many agents as possible from the fragile state. Since our objective concerns the less reliable agents, an egalitarian evaluator is appropriate. We propose to use the classical maximin order over the agents reliability ([8]):

**Definition 2 (Global Reliability Order \( \succ^{\mathcal{R}_{\text{MAS}}} \) ):**

\[
A \succ^{\mathcal{R}_{\text{MAS}}} A' \iff \min_{a \in \text{Agents}} \mathcal{R}_a(A) \succ \min_{a \in \text{Agents}} \mathcal{R}_a(A')
\]

The reliability objective is thus to optimize the maximin criterion over the allocations reliability:

\[
\max_{A \in \text{Allocations}} \left( \min_{a \in \text{Agents}} \mathcal{R}_a(A) \right)
\]

Thus, our purpose is to incrementally (re)allocate resources so that each new allocation is an improvement w.r.t. the maximin ordering.

C. Host Load

The host load depends on the local resources used by the agents. In this work, we focus on two important resources: the processor and the volatile memory. The host performances depend on both these parameters; any overload causes their degradation. Our first objective is to avoid any replication that overloads hosts.

**Definition 3 (Host overload):** Let us note \( \text{Proc}_h(t) \) and \( \text{Mem}_h(t) \) respectively, the processor use rate and the memory use rate for Host \( h \) at time \( t \). Let \( \alpha_h \) be a constant parameter that belongs to \([0, 1]\) defined for host \( h \).

\[
h \text{ is overloaded } \iff \max(\text{Mem}_h(t), \text{Proc}_h(t)) > \alpha_h
\]

Meanwhile, we want to optimize the distribution of resources. Uniformly allocating resources when replicating eases the replication of other agents in further steps. It also allows to maintain a good quality of service for each host. However, the two objectives (reliability optimization and load uniformisation) may be at odds. We describe in Section III-B how we combine them.

III. NEGOTIATION PROTOCOL

The action of an agent depends on its current reliability state. If it is thrifty, it has reached its goal and it does not try to modify the allocation of its replicas. If it is wasteful, it randomly selects a subset of replicas to be destroyed without becoming fragile. If it is fragile, it negotiates with hosts to improve its replicas as follows:

Step 1. Apply

Consider \( H_a^{\text{latency}}(t) \) the set of hosts that are inside a certain latency threshold area of Agent \( a \). The agent sends an application to these selected hosts.

Step 2. Accept/Wait

At regular intervals, each host updates its accepted list and its waiting list. It then sends to the accepted agents an accept message and sends to the other agents a wait message. Agents may answer until the end of the protocol. See Section III-A for all the details of host behavior.

Step 3. Confirm/Cancel

After a given timeout, Agent \( a \) has received responses from the hosts (Accept/Wait). It selects the best allocations according to its preferences and sends a cancelation to hosts that do not match them. It may confirm to the hosts of its preferred allocation or wait for possible new acceptances that may arrive before the expiration time of the already received acceptances. These new acceptances result from other agents cancelations. When a host

1Indeed, since the different replicas of an agent may communicate for maintaining their consistency [6], this condition is necessary to avoid high communication extra cost.
receives a confirm from an agent it replicates it. Section III-B and Algorithm 1 give all the details.

The following sections define the decision processes of the agents and the ones of hosts.

A. Host Selection Process

Each host selects the agents to replicate. The host’s first objective is to avoid any replication that would overload it and the second is to allow an optimization of the replica allocation w.r.t. the maximin order (Definition 2). The host selects iteratively the least reliable applicants that can be replicated simultaneously, together with the already accepted agents, without overloading it. Also, it keeps track of the other applicants in a waiting list. Indeed, since some agents (in particular the less reliable ones) may receive too many acceptances, they would cancel the acceptances of the non-preferred hosts. Resources will thus be freed and may be reallocated to agents on waiting list. A host cannot remove from the accepted list an agent which has not canceled.

B. Agent Selection Process

To select the best allocations, each agent is endowed with a preference ordering which considers both its current reliability and its replicas load balancing.

The replicas load balancing of Agent a (E_a(A)) is evaluated as the loads range of the hosts where a is replicated.

\[ E_a(A) = \max_{h, h' \in R_{epu2}} |C_h(A) - C_{h'}(A)| \quad (7) \]

The preference relation over the replicas loads \( \succ_r \) separates the possible allocations into two classes: those which globally reduce range and the others. In terms of resource consumption, Allocation A is preferred over Allocation A' if and only if A reduces range, whereas A' does not.

Agent preferences are designed to choose the safest allocation among those that do not reinforce load inequities (if any exist).

**Definition 4 (Agent preferences \( \succ_a \) (step 3)):**

\[ A \succ_a A' \iff \neg((A' \succ_a C) \land (R_a(A) \succ_R R_a(A'))) \quad (8) \]

This relation is used by agents to order possible allocations. Note that \( \succ_a \) implies that two allocations that reinforce load inequities will be compared w.r.t. their associated reliability. Hence, given a set of possible allocations, they are ordered as follows: the first ones decrease replicas load balancing, the latter ones do not decrease it, and inside each group the subsets are ordered by decreasing reliability.

Algorithm 1 describes the agent selection process. It is executed regularly while an agent is negotiating. It proceeds by maintaining two lists, selectedAllocations (sA) and bestGuaranteed (bG), in order to verify two loop-invariants, (li1) bG contains the currently preferred allocation for which every host has accepted and (li2) sA contains the allocations that are preferred to bG (each allocation contains at least one wait answer).

**Algorithm 1** Agent’s selection process (step 2).

1: Update sA with the newly received message and save in negotiatingHosts the current set of all hosts of sA.
2: Remove from sA the wasteful allocations and the expired allocations.
3: Update bG and remove from sA all the allocations which are worse than bG.
4: if Replicating on every host of bG makes the agent thrifty or bG will become expired in the next execution of the algorithm or sA only contains bestGuaranteed then
5: Confirm to bG hosts, cancel to others and clear sA.
6: else
7: Cancel to all hosts that belong to negotiatingHosts but do not appear in sA.
8: end if

IV. THEORETICAL ANALYSIS

During the execution of the application, each agent makes a sequence of allocations and destructions of replicas. The algorithm of destruction of replicas preserves the reliability of agents since it does not allow the agent to become fragile. We are thus interested in the creations of replicas made after receiving all the answers following the application phase of an agent (Protocol Step 1).

We note \( \delta_a(t) \) a characterization of the replicas allocation of Agent a at time t, \( \delta_a(t) = (H^b_{a,\text{before}}(t), H^a_{a,\text{allocated}}(t)) \) where \( H^b_{a,\text{before}}(t) \) is the set of hosts where a has been replicated before the last application phase, and \( H^a_{a,\text{allocated}}(t) \) is the set of hosts where a has been replicated after the last application phase. It also includes the hosts of the bestGuaranteed list if the agent is currently negotiating (see Algorithm 1). Similarly, we note \( A^b_{a,\text{after}}(t) \) (resp. \( A^a_{a,\text{after}}(t) \)) the allocation where each agent a is replicated on the host of \( H^b_{a,\text{before}}(t) \) (resp. \( H^a_{a,\text{before}}(t) \cup H^a_{a,\text{allocated}}(t) \)). Finally we extend the definition of an allocation (see Section II) by allowing a restriction to a subset of agents.

\[ \forall N \subseteq \text{Agents}, A_N = (N, \text{Hosts}, R) \quad (9) \]

In this section we first prove that any new replication results in an improvement of the MAS w.r.t. the reliability criterion (see Definition 2). Then we show how our mechanism adapts to the environment changes (occurrence of faults, dynamic evolution of hosts failure probabilities or agents criticalities). Finally we present an analysis of complexity of our negotiation protocol.

A. Convergence

We consider here that no fault occurs and that the preference relation of the agent does not consider load balancing (see Definition 4). Lemma 1 states that whenever a new replica is created for an agent, any other agent that was less reliable is also replicated.
Lemma 1: At any time \( t \), for any agent \( a \) and any agent \( l \), if \((\text{hyp.1})\) Agent \( l \) can access at least a host that has already replicated Agent \( a \) \( (H^t_{\text{latency}}(t) \cap H^t_{\text{allocated}}(t) \neq \emptyset) \) and \((\text{hyp.2})\) Agent \( l \) was less reliable than Agent \( a \) \( (R_l(H^t_{\text{before}}(t)) < R_a(H^t_{\text{before}}(t))) \) then either \((\text{assert.1})\) \( l \) is thrifty or wasteful or \((\text{assert.2})\) \( l \) is fragile and \( l \) has been replicated on, at least, all the hosts where \( a \) has been replicated and that it can access \( (H^t_{\text{before}}(t) \cup (H^t_{\text{latency}}(t) \cap H^t_{\text{allocated}}(t))) \) \( H^t_{\text{before}}(t) \cup H^t_{\text{allocated}}(t) \).

Proof. This lemma is easily proven using reduction ad absurdum with the following tree main considerations: (1) an agent applies to every host it can contact, (2) hosts accept first the less reliable agents, and (3) an agent does not remain fragile if it has been accepted on a sufficient number of hosts.

Theorem 1: Any replication made by any agent \( a \) is associated to a maximin improvement of the set of agents that can be replicated on a subset of \( H^t_{\text{latency}}(t) \):

At any time \( t \), \( \forall a \in \text{Agents}, \)

\[
\begin{align*}
\mathcal{N} &= \{ l \in \text{Agents} | H^t_{\text{latency}}(t) \cap H^t_{\text{allocated}}(t) \neq \emptyset \} \\
H^t_{\text{allocated}}(t) \neq \emptyset & \Rightarrow A^t_{\text{after}}(t) \succ_{\text{MAS}} A^t_{\text{before}}(t)
\end{align*}
\]

Proof. Obvious if \( a \) is the least reliable agent of \( \mathcal{N} \). Otherwise, let \( l \) be the least reliable agent, \( l \) is replicated according to Lemma 1.

B. Auto-Stabilization

The auto-stabilization property of the mechanism arises from its incremental behavior. Each agent regularly updates its criticality and its availability, and then recomputes its reliability. Also it is informed of host failures when they occur by a failure detection service [6].

C. Complexity

The computation and communication complexity of our negotiation protocol depends on the number of hosts contacted by each agent, i.e. the cardinality of the \( H^t_{\text{latency}} \) sets. We consider their mean cardinality, denoted \( k \). The numbers of agents and hosts of the system are respectively denoted \( n \) and \( m \).

The most costly step is the agent selection process (Algorithm 1). Since it considers all subsets of a set of cardinality \( k \), its complexity is exponential on \( k \). The worst case computation complexity is thus \( O(n \times 2^k) \) distributed over the \( n \) agents.

The worst case communication complexity (every agent is fragile) of the negotiation protocol is \( O(n \times k) \):

\( n \times k \) applications are sent at Step 1, less than \( n \times (k \text{ Wait } + k \text{ acceptance}) \) at Step 2, and a maximum of \( n \times k \text{ confirm or cancellation} \) at Step 3.

Note that the communication and computation complexity of our protocol strongly relies on the value of \( k \). This value can be easily dynamically adapted to control the extra cost of the replication management middleware.

V. CONCLUSION AND FUTURE WORK

Replication mechanisms improve the reliability of a large and open MAS must they take into account the fact that crucial factors, such as the criticality of agents, may vary while the MAS is running. In this paper, we proposed a solution to replicate the agents according to two dynamically varying factors: agent criticality and the amount of available resources. We thus introduced a negotiation protocol to adapt resource allocation depending on these factors. This protocol was implemented with the fault-tolerant multi-agent framework DimX. Experiments with various scenarios, including large-scale MAS, are being conducted to further validate our approach and evaluate the replication and monitoring costs.

REFERENCES


