Dynamic Resource Allocation Heuristics for Providing Fault Tolerance in Multi-agent Systems

Alessandro de Luna Almeida
Université Pierre et Marie Curie-Paris6, UMR 7606
4 Place Jussieu
Paris, F-75005 France
Alessandro.Luna-Almeida@lip6.fr

Samir Aknine
Université Pierre et Marie Curie-Paris6, UMR 7606
4 Place Jussieu
Paris, F-75005 France
Samir.Aknine@lip6.fr

Jean-Pierre Briot
Université Pierre et Marie Curie-Paris6, UMR 7606
4 Place Jussieu
Paris, F-75005 France
Jean-Pierre.Briot@lip6.fr

ABSTRACT

In this article, we propose an original method for providing fault tolerance in multi-agent systems. Our method focuses on building an automatic and adaptive replication policy to solve the resource allocation problem of determining where agents must be replicated to minimize the impact of failures. This policy is determined by taking into account the criticality of the agents and the reliability of the machines. We propose then different heuristics for the allocation of the available resources. Some measurements assessing the effectiveness of our approach are also presented.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*; D.2.8 [**Computer Systems Organization**]: Performance of Systems—*Fault tolerance*

General Terms

Algorithms, Performance, Reliability, Experimentation

Keywords

Adaptation, fault tolerance, multi-agent systems, resource allocation, replication

1. INTRODUCTION

The possibility of partial failures is a fundamental characteristic of distributed applications. In order to prevent that a system stops working properly due to the occurrence of faults, many fault tolerance approaches have been proposed, most of which are based on the concept of replication, i.e. creation of copies of a component in distant machines.

Replicating every single component of the application in every machine is not a feasible approach due to the limit on the resources available per machine. In general, it is the responsibility of the designer of the application to explicitly identify what critical components should be made robust and how to parameterize replication. This can be decided statically before the application starts [5, 8] or in a non-automatic way during the execution of the system [4, 7].

However, those works are not suitable for multi-agent systems (MAS) applications for two main reasons: firstly, MASs can be very dynamic and thus, it is very difficult to identify in advance the most critical agents; secondly, a manual control is not realistic, as the application designer cannot monitor the evolution of a distributed application of a significant scale.

In this paper, we will introduce a new approach to building reliable multi-agent systems. We define the resource allocation problem of determining which replication resources must be allocated to each agent so as to maximize the reliability and robustness of the system. The failure probability model used takes into account timing aspects and the choice of the reliability level guaranteed for each agent is based on the concept of criticality, a value (evolving in time) associated to each agent in order to reflect the effects of its failure on the overall system. This paper also reports on different heuristics we proposed to the resource allocation problem.

The remainder of this paper is organized as follows. Section 2 introduces the fault tolerance problem we deal with in this paper. Section 3 defines the resource allocation problem and Section 4 proposes different solutions to it. Section 5 shows the experimental results. Section 6 provides an overview of the state of the art. Finally, in section 7 we present our conclusions and perspectives for future work.

2. CONTEXT OF THIS WORK

The fault tolerance problem described in this paper considers a set of agents that have to complete a set of tasks. We consider an application of assistance for air traffic control through assistant agents (this is a simplified scenario of an ongoing collaborative project with Eurocontrol). The airspace is divided into sectors, each sector being controlled by a human controller (see Figure 1). Each controller is assisted by an assistant agent who monitors the air traffic to suggest decisions about traffic control. Agents communicate in order to assist with collaborative procedures, e.g. hand off procedures, that is when a controller passes the responsibility of an airplane exiting from its supervision sector to

SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil

the controller of the sector the plane is entering.

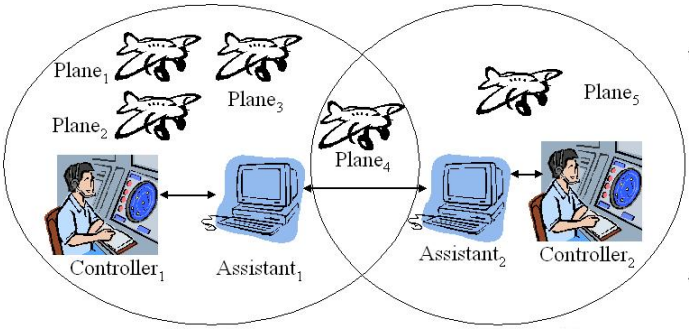

Figure 1: Air traffic control

While trying to accomplish their tasks, agents can be faced to different kinds of failures. In our model, we consider the crash type of failures, i.e. the execution of the agents located at the machine which fails stops definitively and they answer no longer to messages. A machine that fails can be restarted, but with a new state completely reinitialized.

To minimize the impact of failures, agents can be replicated. Since the available resources are often limited, one cannot replicate every agent in each machine. The problem consists in finding a replication scheme which minimizes the probability of failure of the most critical agents. This scheme must also be revised over time, considering that the multi-agent execution context of tasks is dynamic and, thus, the criticalities of the agents vary at runtime.

To control replication in an autonomous way, we address two successive issues. Firstly, we have proposed several metrics for estimating the criticality of the agents, a value that measures the potential impact of the failure of that individual agent on the organization. More details on the metrics (static dependences, dynamic dependences, roles, norms, plans) can be found in [3]. The second issue, which we adress in the next section, is a resource allocation problem which consists of using the criticalities of the agents to determine the resources to allocate to each agent so as to minimize the failure probability of the most critical ones.

3. RESOURCE ALLOCATION PROBLEM

The problem of resource allocation considers a set of agents $S = \{Agent_1, Agent_2, \cdots, Agent_{n_a}\}$ and a set of machines $M = \{m_1, m_2, \cdots, m_{n_m}\}$, where n_a and n_m are respectively, the total number of agents and of machines.

Definition 1. A *resource allocation matrix* is a matrix D_{n_a,n_m}, where d_{ij} is a variable that represents the number of copies of $Agent_i$ in m_j.

The resource allocation matrices are subjected to three constraints: (1) The first one takes into account the fact that it is useless to deploy more than one replica of the same agent in a same machine (because our model of failure considers machine failures):

$$\forall i, j \cdot d_{ij} = 0 \vee d_{ij} = 1 \tag{1}$$

(2) The second constraint imposes a limit on the number of resources nr_j available in the machine m_j, i.e. the number of agents that can be deployed in m_j:

$$\forall j \cdot \sum_{i=1}^{n_a} d_{ij} = nr_j \tag{2}$$

(3) Lastly, since every agent must be deployed somewhere in the system, we add this last constraint:

$$\forall i \cdot \sum_{j=1}^{n_m} d_{ij} \geq 1 \tag{3}$$

Definition 2. For each machine m_j of the system, we define the *failure rate* (denoted as λ_j) as the frequency with which m_j fails. The failure rate is initialized by our framework at each boot of the machine.

The value of each λ_j can be computed using data collected by observing the past failures of each machine. Let T_j be the total time when m_j has been up and N_j be the number of failures during T_j:

$$\lambda_j = \frac{N_j}{T_j} \tag{4}$$

Definition 3. Let λ_j be the failure rate of the machine m_j, and X_j be the random variable representing the time to failure of m_j. Since λ_j is constant, we can define the *failure density function* corresponding to X_j by the following exponential density function:

$$f_j(x) = \lambda_j e^{-\lambda_j x} \tag{5}$$

Definition 4. The *reliability* of a machine m_j at the interval of time $[0, t]$ (denoted by $v_j(t)$) is the probability that it will not crash before the time t.

$$v_j(t) = P(X_j \geq t) = e^{-\lambda_j t} \tag{6}$$

If we assume that the failures of the machines where an agent $Agent_i$ is deployed are independent, it is easy to show that the probability p_i that this agent $Agent_i$ will not fail before t (i.e. its reliability) is given by the equation:

$$p_i(t) = 1 - \prod_{j=1}^{n_m} \left(1 - d_{ij} e^{-\lambda_j t}\right) \tag{7}$$

Definition 5. Let S be the set of agents and c_i the criticality of $Agent_i$ (calculated using one of the metrics described in section 3). Then, we define the *utility of the multi-agent system* (denoted as $u(S)$) as the total importance of all the agents of the system:

$$u(S) = \sum_{i=1}^{n_a} c_i \tag{8}$$

Definition 6. Let D be a resource allocation matrix, c_i the criticality of $Agent_i$ and p_i the probability that $Agent_i$ will not fail (the time parameter t is omitted for simplicity reasons). The *expected utility of the MAS* deployed using D is defined as the expected value of the utility function. It can be calculated as follows:

$$E_D\left(u\left(S\right)\right) = \sum_{i=1}^{n_a} c_i \times p_i \tag{9}$$

The higher the value of $E_D\left(u\left(S\right)\right)$, the more efficient and fault-tolerant the allocation D.

Definition 7. We define the *resource allocation problem* as the optimization problem of finding one resource allocation matrix D_{max} which respects the constraints (1), (2) and (3) and that gives a maximum value for the expected utility of the MAS ($E_{D_{max}}\left(u\left(S\right)\right)$ is maximum).

In the next section, we will present the strategies we have proposed to solve this problem. These strategies make no assumption about the distribution model. In fact, they can be implemented in a centralized or in a distributed way. Furthermore, the allocation matrix found by them are revised along time. The revision strategies do not require a recalculation of the allocation since it can just be adapted in an incremental way. More details can be found in [1].

4. RESOURCE ALLOCATION STRATEGIES

4.1 Optimal allocation

To find the optimal allocation, we run a depth first search in the space of possible allocations. The agents are initially ordered by their criticalities in different levels such that the k^{th} recursion level in this search corresponds to the k^{th} most critical agent. At each level k, the variables of the agent $Agent_k$ are assigned (corresponding to the k^{th} row of the allocation matrix D).

To prune the search tree, we backtrack the search whenever a constraint is violated. Furthermore, a property of the optimal solution is that an agent must be less reliable than all the agents more critical than it. Then, when assigning the variables corresponding to the agent $Agent_k$, our method guarantees that the reliability of $Agent_k$ (given by the equation 7) is lower than the reliabilities of all the agents more critical than it, otherwise we backtrack.

Another pruning technique is the branch and bound, which consists in backtracking whenever the optimal solution can not be obtained using the current assignment of the variables. For that, the algorithm determines an upper bound for the value that can be obtained for $E_D(u(S))$ with the current assignment of variables. If this upper bound is lower than the best solution found until now, it is certain that the current assignment of the variables does not correspond to an optimum. Let k be the current level of the search and nd_j the number of resources still available in the machine j, an upper bound is given by:

$$bound = \sum_{i=1}^{k-1} c_i \times p_i + c_k \times \sum_{j=1}^{n_m} nd_j \times v_j \tag{10}$$

The worst case complexity to find the optimal allocation is exponential in the number of machines and of agents.

4.2 Proportion-based Heuristics

[6, 2] propose what we call the Proportion_Number heuristic, which limits for each agent the number of machines where it can be deployed. This limit is directly proportional to the criticality of the agent and to the number of available resources.

The problem of those works is that the reliabilities of the machines are not taken into account. Hence, we propose a new heuristic (Proportion_Reliability), where an agent can be deployed in a set of machines as long as the sum of reliabilities of those machines does not exceed a threshold, as shown by the following equation:

$$\forall i \cdot \sum_{j=1}^{n_m} \left(d_{ij} \times v_j\right) \leq \frac{c_i \times \sum_{j=1}^{n_m} \left(nd_j \times v_j\right)}{\sum_{k=1}^{n_a} c_k} \tag{11}$$

Among the sets of machines that satisfy the equation 11, we look for the one with minimal failure probability. For that, firstly we sort the machines in decreasing order of reliability. Then, for each available resource in each machine, we allocate it to the agent that is not deployed in this machine and that has the highest threshold. After allocating a resource to an agent, the threshold of the agent is decremented by the reliability of the corresponding machine.

The complexity of the Proportion-based strategy is $O(n_a \times log(n_a) + n_m \times log(n_m) + n_r \times n_a)$, where n_r is the total number of resources available in all the machines.

4.3 Greedy Heuristics

The problem of the proportion-based heuristics is that they do not try to maximize explicitly the expected utility. We propose then greedy heuristics that incrementally builds the solution and that takes directly into account the expected utility so that it can be maximized.

For that, machines are initially sorted in decreasing order of reliability. Each resource of each machine is then allocated to the agent that is not deployed in the machine and such that the value of $E_D\left(u\left(S\right)\right)$ is maximized. The corresponding algorithm is given by the Algorithm *Greedy Allocation*.

Algorithm 1 Greedy Allocation

```
sort the machines in decreasing order of reliability
for all machine m_j do
  for all resource in m_j do
    Δ_max ← 0
    for all agent i such that d_ij = 0 do
      D' ← D
      d'_ij ← 1
      Δ_i ← E_D'(u(S)) − E_D(u(S))
      if Δ_i > Δ_max then
        Δ_max ← Δ_i
        D'_max ← D'
      end if
    end for
    D ← D'_max
  end for
end for
```

The complexity of this algorithm is $O(n_a \times log(n_a) + n_m \times log(n_m) + n_r \times n_a)$, the same as the proportion-based one, but in practice it is slower since floating point operations are necessary to calculate the values of Δ_i.

The basic greedy algorithm does not guarantee that an agent is less reliable than all the agents more critical than it. We propose then two variants to solve this problem.

The first one (Greedy with ordering) consists of making the agents exchange among them the set of machines where they will be deployed (i.e. the rows of the allocation matrix) in the end of the greedy algorithm. The exchanges are made according to the reliabilities of the set of machines allocated to each agent. The less reliable set is allocated to the less critical agent, the second less reliable set to the second less critical agent, and so on to the most reliable set that will be allocated to the most critical agent. The complexity of this variant is $O(2 \times n_a \times log(n_a) + n_m \times log(n_m) + n_r \times n_a)$.

In the second variant (Greedy with swap), instead of delaying the swap operations to the end of the algorithm, we make them after the allocation of each resource but only if necessary. The complexity of this variant is $O(n_a \times log(n_a) + n_m \times log(n_m) + 2 \times n_r \times n_a)$.

4.4 Local Search Heuristics

None of the preceding heuristics guarantees to find the optimum allocation matrix. We propose then to use two different local search techniques to try to ameliorate a given allocation: Hill Climbing and Tabu Search.

The algorithm Hill Climbing consists of starting with an initial solution and moving iteratively in the search space toward better neighbor solutions. In our case, neighbor allocations are obtained by reallocating one resource from one agent to another. The problem of Hill climbing is that it can get stuck in local optimums. The Tabu Search method pursues the local search whenever it encounters a local optimum by allowing non-improving moves. Cycling back to previously visited solutions is prevented by the use of memories that record the recent history of the search.

5. EXPERIMENTATION AND RESULTS

To implement the resource allocation heuristics described in the preceding section, we have extended the framework DARX [10] with an adaptive replication control module which calculates and updates in a distributed way the criticalities and uses the replication service to deploy the resource allocation matrix obtained. For space reasons, we omit the details of our architecture.

5.1 Experimental Setup

We have conducted experiments in order to evaluate the different heuristics proposed in this paper and compared them to a random allocation. We varied the number of agents $(100, 200, 300, \cdots, 1000)$ and used a number of machines equal to 20% of the number of agents. Moreover, we varied the number of agents that can be deployed at each machine in the range $\{6, 8, 10, 12, 14\}$. The criticalities of the agents and the failure rate of the machines were varied according to five homogeneity levels. Table 1 shows the interval of values for the criticalities and for the failure rates (in failures per minute).

For each combination (number of agents, number of machines, number of resources by machine, homogeneity level), 1000 test cases were generated, by varying the criticalities and the failure rates. Each heuristic has been tested using the same test cases. We have compared the expected utility of the allocations obtained ($E_D(u(S))$) for each heuristic as well as the CPU time necessary for execution. The results shown are the average of the 1000 test cases. For the local search heuristics, 1000 iterations have been performed.

Table 1: Homogeneity Levels Intervals of Values

Level	Criticality Interval	Failure Rate Interval
Very Low	[1,10000]	[0.01, 4]
Low	[1250,8750[	[0.1, 2]
Medium	[2500,7500[	[0.3, 1.3]
High	[3750,6250[	[0.5, 1]
Very High	[4500,5500[	[0.6, 0.8]

5.2 Results and Discussion

The order of the heuristics in increasing order of $E_D(u(S))$ for all the configurations is as follows: Random, Proportion_Number, Proportion_Reliability, Hill Climbing, Greedy and, finally, Tabu Search (see Figure 2 for a very low homogeneity level and 10 resources by machine). As expected, the Greedy heuristic is better than the other ones since it maximizes explicitly the expected utility. Hill Climbing enhances the result obtained by the algorithms that define the initial allocation. Tabu Search enhances the results of the Greedy heuristic, but less than 0.1%.

The difference of the results between the Greedy and the others is higher when the number of agents is augmented (see Figure 2). This is due to the fact that when there are fewer agents, fewer allocation matrices exist. The difference of the results is smaller with a higher homogeneity level. Indeed, when the homogeneity level is very high, the different possible allocations give similar values for $E_D(u(S))$.

Table 2 shows the CPU time (in ms) for the very low homogeneity level, 14 resources by machine and 1000 agents.

Table 2: Comparison of the CPU Time of the heuristics (in ms)

Random	Proportion	Greedy	Hill	Tabu
10.3	25.5	46.6	2982.7	3 hours

The results shows that the Greedy heuristic is the best one with respect to the expected utility and it scales well when the number of agents and machines augments. However, the proportion-based one can also be used if the homogeneity level is high, since it can lead to similar values for the expected utility but with better time performance. Finally, the small gain in the expected utility obtained by Tabu search does not justify its use, since it is very time-consuming.

6. RELATED WORK

Several approaches have addressed the problem of fault tolerance. [5] implements passive replication in a transparent way using proxies. All messages going to and from a replicate group are funneled through the replicate group message proxy. In [8], the problem of fault tolerance is defined as a deployment problem and a probabilistic approach is proposed to deploy the agents in a multiagent application. The main problem of these two works is that replication is applied statically before the application starts.

In distributed computing, there are some software infrastructures for adaptive fault tolerance [4, 7] where existing strategies can be dynamically changed. Nevertheless, such a

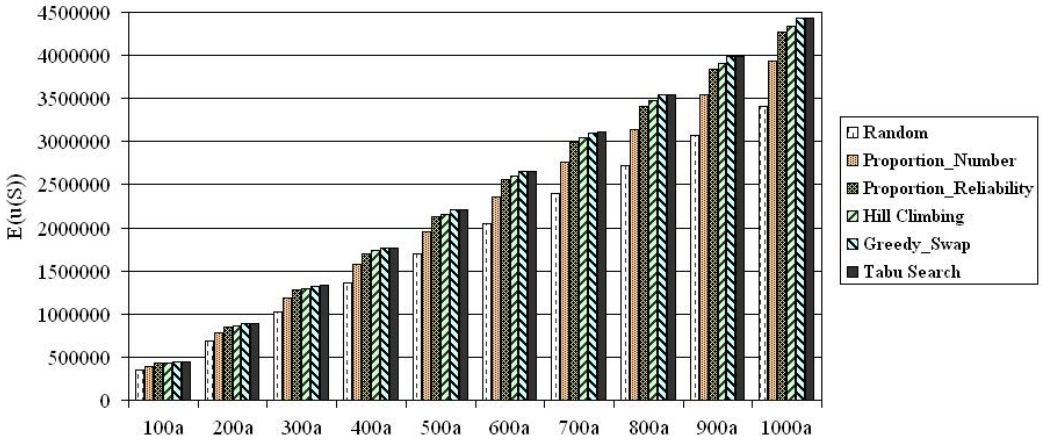

Figure 2: Comparison of the heuristics for a very low level of homogeneity and 10 resources by machine

change must have been devised before runtime or the modifications must be specified and applied in a non-automatic way during the execution of the system.

As we said before, there are works [6, 2] that propose an adaptive replication mechanism. However, they do not consider the probability of failures of replicas nor the cost of replication and they do not address the problem of where deploying the agents.

Several distributed and asynchronous approaches exist for Distributed Constraint Optimization Problems, such as ADOPT[11] and OptAPO[9]. However, as noted by these authors, their algorithms are less efficient for n-ary constraints, which is our case.

7. CONCLUSION

Multi-agent systems are often distributed and must run without any interruption. In our approach to make these systems reliable, firstly we have proposed several metrics to evaluate dynamically the criticality of the agents. Then we proposed different heuristics to the resource allocation problem of deciding where to deploy each agent in order to maximize the reliability of the system. To test the heuristics, we have implemented the corresponding algorithms in a middleware for distributed fault-tolerant applications. We think that our current results are promising because: (1) the algorithms allow to decide automatically and dynamically where to deploy each agent; (2) they present a negligible complexity and overhead; (3) they provide a very satisfactory reliability level for the agents.

One of the perspectives of this work is to consider other constraints such as the memory space available by machine, the memory space required by each agent and the minimal reliability required by the agents. Additionally, we are currently testing other strategies to the resource allocation problem and we will experiment our allocation strategies on scenarios for air traffic control applications.

8. REFERENCES

[1] A. L. Almeida, S. Aknine, J.-P. Briot, and J. Malenfant. A predictive method for providing fault tolerance in multi-agent systems. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2006)*, pages 226–232, 2006.

[2] S. Bora and O. Dikenelli. Applying feedback control in adaptive replication mechanisms in fault tolerant multi-agent organization. In *Software Engineering for Large-Scale Multi-Agent Systems*, pages 5–12, 2006.

[3] J.-P. Briot, Z. Guessoum, S. Aknine, A. L. Almeida, N. Faci, M. Gatti, C. Lucena, J. Malenfant, O. Marin, and P. Sens. Experience and prospects for various control strategies for self-replicating multi-agent systems. In *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'2006)*, pages 37–43, 2006.

[4] M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr, and R. E. Schantz. AQuA: An adaptive architecture that provides dependable distributed objects. In *17th IEEE Symposium on Reliable Distributed Systems*, pages 245–253, 1998.

[5] A. Fedoruk and R. Deters. Improving fault-tolerance by replicating agents. In *AAMAS'02*, pages 737–744, 2002.

[6] Z. Guessoum, J.-P. Briot, O. Marin, A. Hamel, and P. Sens. Dynamic and adaptive replication for large-scale reliable multi-agent systems. In *Software Engineering for Large-Scale Multi-Agent Systems*, pages 182–198, 2003.

[7] Z. T. Kalbarczyk, S. Bagchi, K. Whisnant, and R. K. Iyer. Chameleon: A software infrastructure for adaptive fault tolerance. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):560–579, 1999.

[8] S. Kraus, V. Subrahmanian, and N. C. Tacs. Probabilistically survivable MASs. In *IJCAI'03*, pages 789–795, 2003.

[9] R. Mailler and V. Lesser. Solving Distributed Constraint Optimization Problems Using Cooperative Mediation. In *AAMAS'04*, pages 438–445, 2004.

[10] O. Marin, M. Bertier, and P. Sens. Darx - a framework for the fault-tolerant support of agent software. In *IEEE International Symposium on Software Reliability Engineering*, pages 406–417, 2003.

[11] P. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal*, 161(1-2):149–180, 2005.