

The 2002 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'02)

## Evaluating Task Redistribution Methods for Fault Clearing in Multi-Agent Systems

Maryam S. Mirian, Majid Nili Ahmadabadi, Zainalabedin Navabi

Robotics and AI Lab, Dept. of Elect. and Comp. Eng., Faculty of Engineering, University of Tehran  
 Mirian\_e@mehr2000.com mnili@ut.ac.ir navabi@ece.neu.edu

### Abstract

*Distributed hardware systems can be thought as teams of distributed cooperative agents. This will encourage the designers to develop some new agent-based techniques to increase systems' fault tolerance.*

*Multi agent systems, similar to other distributed systems, are prone to failures. An important challenge to creating an effective and functional multi-agent system is providing it with sufficient capabilities to operate properly and acceptably either in the case of potential faults.*

*In this research, clearing faults by helping the faulty agents in performing their tasks is considered. In addition, some distributed decision-making methods are introduced for each agent to decide if it can help the faulty agents by undertaking their tasks in different conditions.*

*The developed methods are implemented in a simulated Distributed Control System. The results show the effectiveness of the proposed distributed fault clearing method.*

**Keywords:** Multi-agent system (MAS), Fault-Recovery, Help Request, Task criticality.

### 1. Introduction

Traditionally, to have fault tolerant system, we can build subsystems from redundant components placed in parallel. Many fault-tolerant computer systems mirror all operations, e.g., every operation is performed by two or more duplicate systems, so if one fails the other can take over [1].

Fault tolerant techniques used in traditional MAS is limited to using agents as the backups of each other. In other words, they use NMR (N-Modular Redundancy) to achieve more robustness. In general, being modular and acting totally independent of each other, makes it possible to handle a fault in a MAS and isolate it, in order not to produce an error or a failure at worst.

This paper discusses the use of the task performing agents to help the others by reconfiguring their roles to recover the lost capabilities. In the presented method, there is no extra or central agent, sentinel or broker to observe the agents and redistribute the tasks among the agents to clear the fault. They also do not make a model of each other. In fact the system is totally distributed and each agent takes proper actions based on a designed cooperation strategy to clear the fault. The presented methods are tested on a simulated distributed hardware system.

### 2. Related Works

Jennings showed that as the world becomes more complex and variable and plans tend to fail more often, teams as a whole waste fewer resources and are more robust than self-interested agents [2]. Hugg uses external sentinel agents to monitor inter-agent communication, build models of other agents, and take corrective actions [3]. The sentinel agents listen to all broadcast communication, interact with other agents, and use timers to detect agent crashes and communication link failures. A sentinel agent copies the world model of other agents and detects inconsistencies by observing the behavior of other agents as well as its own internal state. Klein proposes to use exception-handling service to monitor the overall progress of a multi-agent system [4]. Agents register a model of their normative behavior with the exceptional-handling service that generates sentinels to guard the possible error modes.

The exception-handling services use a query and action language to interact with the problem solving agents to detect and diagnose faults and take corrective actions. A social diagnosis approach is used by Kaminka and Tambe wherein socially similar agents compare their own state with the state of other agents for detecting possible failure [5]. An explicit teamwork model is used for failure diagnosis. The agents use plan recognition from observable actions as well as communication with

other agents to infer and construct a model of the other agents. Decker, Sycara, and Williamson advocate the use of caching by individual agents in systems that use matchmakers to improve robustness in the face of matchmaker failures [6]. They have also shown that using load balancing by brokers in brokered systems improves performance and hence provides a degree of robustness from aggressive agents. In [7] the ability of reorganization in the organizational learning model to maintain the collective performance of multiple robots in terms of fault tolerance is discussed. The presented method in this reference is not a solution for the real time applications because of time-consuming process of learning method and probably the high number of failed tasks. A novel reconfiguration technique inspired from mechanisms that take place during the embryonic development of living beings is proposed in [8]. It illustrates that the rapid low-level fault-recovery characteristic of the embryonic system makes it a promising approach for real-time control applications. [9] solves the problem by a biological perspective using the human immune system as a source of inspiration. As described in [11], there are different factors to be considered while one robot asked to help the other, e.g. its distance from the faulty robot, mechanical capabilities, expertness, current state and criticality. In [10] the helping capability is added to the Alliance architecture of Parker which was originally described in [12]. Inspired by [11], in this paper, the task performing agents are used to provide help for the faulty one and there is no dedicated helper agent like a broker, a matchmaker or even a sentinel. Since using such solutions specifically dictates the presence of a more powerful agent that is the single point of failure of the system and in contradiction with the original goal of fault tolerance. Using similar and normal agents with the capability of help and taking different roles in fault situations, provides a more general and reusable system.

Figure 1: The Agents should contain normal operations, Help procedure and decision making capability.

### 3. The Approach

Reconfiguring the roles of agents and their capabilities, due to the requested type of help can be used for fault recovery. The required features are shown in Figure 1 and described in details in this section.

#### 3.1 What is each Agent's Capability ?

Each agent has some normal capabilities to perform its assigned tasks. Besides, the agent has to know something about the other agents and its environment. In our task, which will be described later, arrival rate of data, deadline of command submission, the criticality of tasks compose the primary knowledge of the agents. It is also assumed that each agent is capable of doing the others tasks.

#### 3.2 What is the Content of Help Request?

The main problem is that the help request must be as short as possible and contains the required information, such as agent' ID and type of help request. In this paper just the ID of the faulty agent is communicated. If the damage is so severe, that the agent cannot send a help request containing type of help needed, the other agents looking at the common bus, will detect the ID of the faulty agent and try to help.

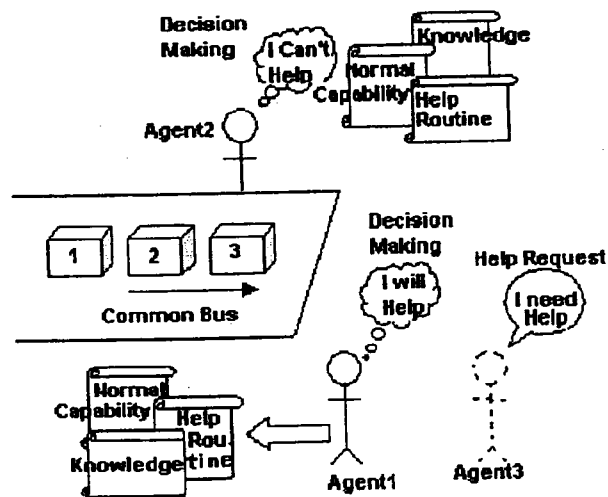
#### 3.3 Who will Receive the Help Request?

If the faulty agent does not know who can help, it can just broadcast a message to call the others for help. So if the criticality of agents' tasks is known, other agents try to help, as they should do. Otherwise they may look at their own capabilities and decide according to decision-making criteria. More details of the testbed will be discussed in the coming section.

### 4. Our Testbed

In order to justify the developed ideas, we designed a test bed similar to a typical distributed control system. It contains a frame generator that produces the normal data for the agents and put it on the bus. These data may be extracted from the sensors in the environment.

These structurally similar but behaviorally different agents gather their own data from the bus and after computing the desired commands; they send them out on the other bus. These commands can control the other subsystems out of this environment. One fault generator is put in the system in order to simulate random faults for each agent during simulation. All system



components must be active in parallel simultaneously. Considering these requirements and the possibility of testing the developed ideas in a hardware system, VHDL simulation was performed [13]. The time resolution of this simulation is as tiny as nano seconds. Describing the agents in a high level behavioral model, enables us to take advantages of the strong features of VHDL like concurrency aspects, design hierarchy, timing control in all levels and many other benefits. We have done four different experiments on this system, which are described below.

**5. Introduced approaches and Simulation Results**

First of all, the test scenario, which is applied to all the experiments, is described and the health status of the agents is shown in Figure 2.

|        |             |                                |                                   |                  |            |
|--------|-------------|--------------------------------|-----------------------------------|------------------|------------|
| Agent1 | OK          | zzzzzzz                        | OK OK OK OK OKOK                  | zzzzzzzzzzzzzzzz | OK OK OKOK |
| Agent2 | OK OK       | zzzzzzzz                       | OK OK OK OK OKOK OK OK OK OK OKOK |                  |            |
| Agent3 | OK OK OK OK | zzzzzzzzzzzzzzzzzzzzzzzzzzzzzz |                                   | OK OK OK OK OKOK |            |

Figure 2: The Test Scenario used for all Experiments

Test Scenario : Agent 1 becomes faulty at 700 NS and becomes healthy at 3000 NS. While Agent 1 is faulty Agent 2 becomes faulty too at 2000 NS and comes back at 6000 NS. Agent 3 is ok since 4000 NS. It fails at this moment. It takes until 10000NS. Agent 1 again at 8000 NS becomes faulty and comes back at 11000 NS. Finally at 11000 NS, all the agents are healthy and continue their normal operations.

**5.1 Helping Strategy based on the Agents' Criticality**

In this experiment, the agents' tasks are assigned some predefined levels of criticality. Therefore, when one agent requests for help, there are some particular agents obliged to help in a predefined manner. The behavior of the agents is described below:

*Agent 1:* Performing the Most Critical Task. Never gives up its own task to help any other agent.

*Agent 2:* Performing the Middle Critical Task. It only helps Agent 1 if it needs help and Agent 3 is faulty. It helps Agent 3 if Agent 1 does not need help.

*Agent 3:* Performing the Least Critical Task. It helps Agent 1 and Agent 2 whenever they request. It may give up its task while helping the two more important agents.

*What will happen in this experiment?*

The most critical agent, Agent 1, becomes faulty and requests for help. Agent 3 starts to help it. It will do its own task while helping Agent 1. After a few nano seconds, Agent 2 becomes faulty too. Now Agent 3 must help Agent 2 too. So it is to give up its task and just perform the most critical tasks of the system without which system will surely fail. After a few more nano seconds, Agent 3 fails and since Agent 2 is faulty too, no one helps them. When Agent 2 comes back it helps Agent 3. But when it understands that Agent 1 needs help it stops helping Agent 3. Finally all the agents came back to the fault free states and continue their normal actions.

Figure 3 shows the internal states of the agents in this experiment. Table 1 and Table 2 demonstrate the simulation results with and without help mechanism respectively. It is worth mentioning that in this experiment, tasks may be lost due to two different reasons: either helping others and losing own task or not being helped by the others.

In order to evaluate this method and the other strategies introduced in this paper, a simple performance index, is considered:

$$Performance = \sum_i N_i C_i$$

Where  $N_i$  is the number of times agent  $i$ 's task is done successfully, and  $C_i$  is the criticality of agents's task.

According to Table 1, Table 2 and the above definition, the performance of the system without the helping capability is 59%, while the performance of the new system with the fixed-criticality-based help is 80%.

| Agent | Number of Lost tasks due to Fault | Total Number of tasks to be done |
|-------|-----------------------------------|----------------------------------|
| 1     | 43                                | 85                               |
| 2     | 26                                | 97                               |
| 3     | 45                                | 103                              |

Table 1: Simulation Result when there is no help mechanism in the system applying the test scenario.

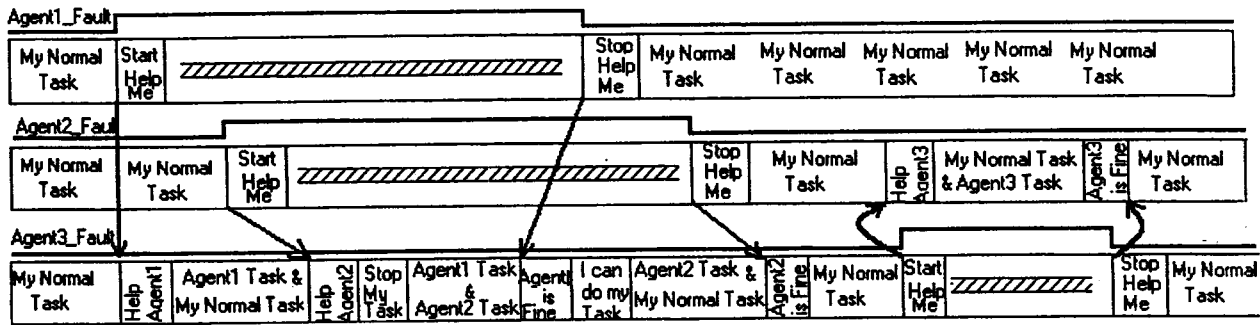


Figure 3: Internal states of the agents in the strategy based on the Agents' Criticality (No Decision Making).

| Agent | Criticality | Number of Lost Tasks | Number of Successfully done Tasks | Number of Tasks done by Itself | Number of Tasks helped by Others | Total Number of tasks to be performed | Fault Duration |
|-------|-------------|----------------------|-----------------------------------|--------------------------------|----------------------------------|---------------------------------------|----------------|
| 1     | 3           | 5                    | 80                                | 46                             | 34                               | 85                                    | 5300 NS        |
| 2     | 2           | 27                   | 70                                | 61                             | 9                                | 97                                    | 6000 NS        |
| 3     | 1           | 63                   | 40                                | 29                             | 11                               | 103                                   | 4000 NS        |

Table 2: Simulation Result when the agents help others with the fixed criticality based method.

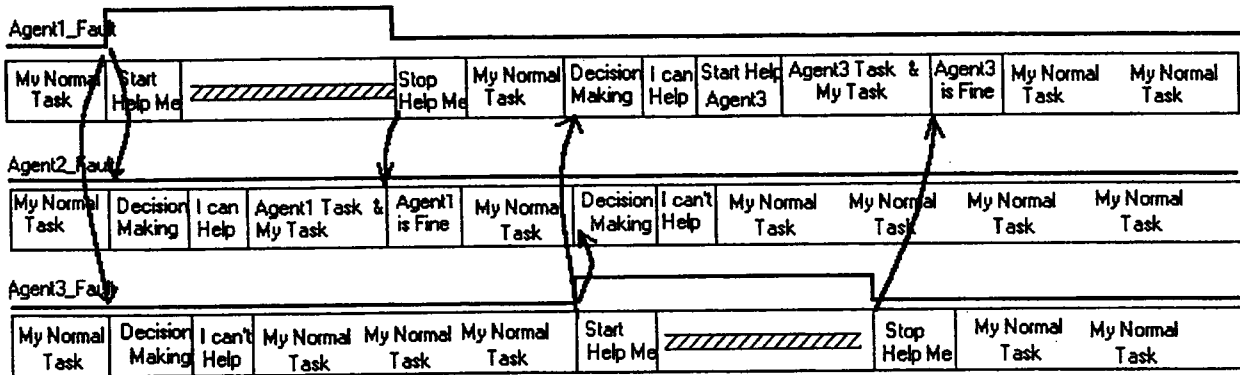


Figure 4: Internal states of the agents in Decision-Making based Methods (Last Three Experiments)

| Agent | Criticality | Number of Lost Tasks | Number of Successfully done Tasks | Number of Tasks done by Itself | Number of Tasks helped by Others | Total Number of tasks to be performed | Fault Duration |
|-------|-------------|----------------------|-----------------------------------|--------------------------------|----------------------------------|---------------------------------------|----------------|
| 1     | 1           | 11                   | 74                                | 45                             | 29                               | 85                                    | 5300 NS        |
| 2     | 1           | 10                   | 87                                | 57                             | 30                               | 97                                    | 6000 NS        |
| 3     | 1           | 24                   | 79                                | 45                             | 34                               | 103                                   | 4000 NS        |

Table 3: Simulation Results when the agents help others with a decision- making based on Remaining Time.