A New Task Redistribution Method 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
P.O.Box: 14395/515, Tehran, Iran

mirian@khorshid.ece.ut.ac.ir, mnili@ut.ac.ir, navabi@ece.neu.edu

Abstract-- **Looking at distributed hardware systems as teams of agents enables the designers to use new techniques developed for multi-agent systems to increase fault tolerance.**

Multi agent systems like 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. Most of the work done in this area, deals with exception handling and detecting inconsistencies among agents and to resolve them in order not to cause harmful damages.

This paper introduces some fault tolerance techniques that provide help for faulty agents in order to reconfigure the system. Agents take new roles to compensate the deficiency of the faulty components. A new help strategy is represented and the implementation results in a simulated distributed hardware environment are discussed.

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

I. Introduction

The ability of a system to respond gracefully to an unexpected hardware or software failure is called *fault tolerance*. 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 are 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 MASs 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 using VHDL.

II. 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. A novel reconfiguration technique inspired from mechanisms that take place during the embryonic development of living beings is proposed in [7]. It illustrates that the rapid low-level fault-recovery characteristic of the embryonic system makes it a promising approach for real-time control applications. In [8] the problem is solved by a biological perspective using the human immune system as a source of inspiration.

As described in [10], 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 [9] the helping capability is added to the Alliance architecture of Parker which was originally described in [11]. Inspired by [10], 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. Using such agents specifically dictates the presence of a more powerful agent that is the single point of failure of the system, which is 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.

III. THE FRAMEWORK

Reconfiguring the roles of agents and their capabilities due to the requested type of help can be used for fault recovery. For this purpose, a designer should consider some important features for each agent. These required features are shown in Figure 1.

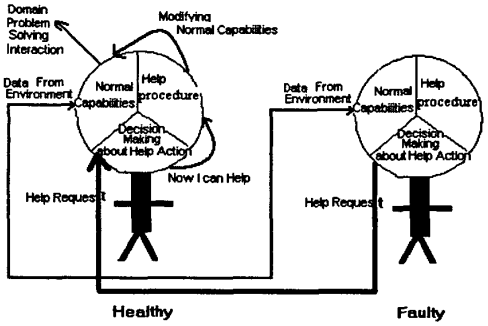

Figure 1: *The Agent should contain normal capabilities, help procedure and decision making capability.*

A. Agent Capabilities and Primary Knowledge

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.

B. Information Content of the Help Request

The main problem is that the help request must be as short as possible and contains the required information, such as its ID and type of help request. In this paper just the ID of 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.

C. The Receivers of 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 their decision-making criteria. More details of the testbed will be discussed in the coming section.

IV. TESTBED DESCRIPTION

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.

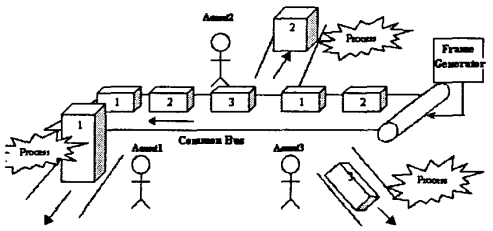

Figure 2: *A simplified schematic view of the test bed.*

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. The simplified schematic view of the system is shown Figure 2.

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 [12]. 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 strong features of VHDL like concurrency aspects, design hierarchy, timing control in all levels, etc. We have done two different experiments on this system as described below.

V. INTRODUCED METHODS AND SIMULATION RESULTS

A. Fixed Criticality-based method

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:

Agent1: The Most Critical Task. Never gives up its own task to help any other agent.

Agent2: The Middle Critical Task. It only helps Agent1 if it needs help and Agent3 is faulty. It helps Agent3 if Agent1 does not need help. When Agent3 requests for help, it will help with half frequency of doing its own task.

Agent3: The Least Critical Task. It helps Agent1 and Agent2 whenever they request. It may give up its task while helping the two more important agents.

Test Scenario1: The most critical agent, Agent1, becomes faulty and requests for help. Agent3 starts to help it. It will do its own task while helping Agent1. After a few minutes, Agent2 becomes faulty too. Now Agent3 must help Agent2 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 minutes, Agent3 fails and Agent2, which is healthy now, starts to help. Finally all the agents come back to the fault-free states and continue their normal actions.

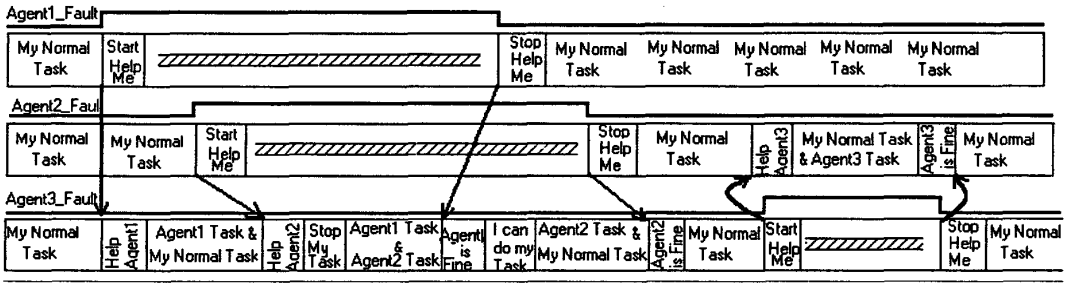

Figure 3: *Internal states of the agents when fault occurs and the fixed-criticality-based method used (in Scenario 1).*

Table 1: *Simulation Result when the agents help others with the fixed criticality based method (Using Scenario1).*

Agent	Criticality	Number of Successfully done Tasks without delay	Number of Delayed Tasks & Delay Amount	Number of Lost Tasks	Fault Duration	Number of Tasks done by Itself	Number of Tasks helped by Others	Total Number of tasks to be performed
1	3	60	2: 70, 70 NS	0	2300 NS	49	13	62
2	2	55	1: 180 NS	11	3800 NS	46	10	67
3	1	27	3:60,60,180 NS	36	5000 NS	21	9	66

Figure 3 shows this scenario. 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.

Table 2: *Simulation Result when there is no help mechanism in Test Scenario 1.*

Agent	Criticality	Lost tasks due to Fault	Successfully done tasks
1	3	23	39
2	2	32	35
3	1	37	29

In order to evaluate this method, a simple performance index, is considered:

$$Performance = \sum_i N_i C_i$$

where N_i is the number of times agnet i's task is done successfully, and C_i is the criticality of agents's task.

According to the two above tables and definition, the performance of the system without the helping capability is 56%, while the performance of the new system with the fixed-criticality-based help is 82%. This is a significant improvement as there is only 18% decrease in the system performance in comparison with the perfect fault-free system.

B. Decision-Making based method

In this experiment, the principal assumption is that the task completion of each agent has the same degree of importance for the total system and no criticality is specified in the design time. Therefore the agents must decide in the run time whether to help or not. So, the agents are given some level of knowledge about the timing constraints of the other agents and also their own limitations that must be considered while deciding to help.

Each agent while receives a help request, looks if it can help. The parameters it considers are:

1) *My_Task_Completion_Time (Its own task completion time):* This parameter is float according to the sensory input data. The reason is that the required processing time depends on the input information. This will enable the agents to decide more dynamically.

2) *Agent_I_Task_Completion_Time (The maximum time required to compute the task of agent I on the helper's processor.)* This is a part of the knowledge of one agent about the others.

3) *Available_Time (The remaining time until the next data comes in):* This time parameter limits the agent to complete its current task during a period of time. If the current task is not completed in this interval of time, it will be assigned a new task and the previous one will be lost and overwritten.

In general, when one agent receives two Help Requests from Agent_i and Agent_j, it will try to decide according to this inequality:

Available_Time
>
My_Task_Completion_Time
+
Agent_i_Task_Completion_Time
+
Agent_j_Task_Completion_Time

If this inequality cannot be satisfied, the agent will think if it can help to just one of them:

Current_Task_Time_Available
>
My_Task_Completion_Time
+
Agent_i_Task_Completion_Time
or
Current_Task_Time_Available
>
My_Task_Completion_Time
+
Agent_j_Task_Completion_Time

If both of these inequalities can be satisfied, the helper agent will help the agent with a longer task completion time. Choosing this task to perform, it will be more probable that the other agents can help the remaining faulty agent. Otherwise, if none of the agents can be helped, the agent ignores the help request and just completes its own task. It is worth mentioning that in such a case, the faulty agent may be helped later since, as described before, *My_Task_Completion_Time* is not fixed and when it shortens gives the agent the opportunity to help.

In this experiment, we expect that only the faulty agent may lose tasks if and only if it is not helped by other agents. In other words, no healthy agent may lose its own task any more because of helping others. This fact is the actual reason of high loss of tasks in the previous experiment.

1) Scenario 1

The first testing scenario is exactly the same as the test scenario shown in Figure 3. The simulation results are shown in Table 4. The performance evaluation according to the given performance index results in 8% improvement in comparison with the system uses fixed-criticality-based method (shown in Table 1). The number of lost tasks decreases from 24% to 9%. The detailed results justification comes after the second scenario.

2) Scenario 2

The second scenario is shown in Figure 4. The simulation results of the system without and with using decision making-based help mechanism are shown in Table 3 and Table 5 accordingly.

Table 3: *Simulation Results when there is no help mechanism in the system, using scenario 2.*

Agent	Criticality	Lost tasks due to Fault	Successfully done tasks
1	1	37	27
2	1	0	67
3	1	21	39

Table 6 and Table 7 summarize all the results. They show that as the method becomes more flexible, the number of lost tasks decreases and the performance increases considerably. It is important to note that, there exists a delay time while one agent helps another one. In spite of the previous experiment that each helpful agent acts exactly the same as the faulty agent in terms of timing constraint, here we can see a delay time in this helping schema. This delay is acceptable and always is less than 60 ns which never disturb the system output. It is reasonable since each agent when decides to help, first completes its own task and then helps the other.

Table 4: *Simulation Results when the agents help others with a decision- making- based method (Using Scenario1).*

Agent	Criticality	Number of Successfully done Tasks with at most 60 ns Delay	Number of Lost Tasks	Fault Duration	Number of Tasks done by Itself	Number of Tasks helped by Others	Total Number of tasks to be performed
1	1	56	6	2300 NS	47	9	62
2	1	60	7	3800 NS	42	18	67
3	1	60	6	5000 NS	36	24	66

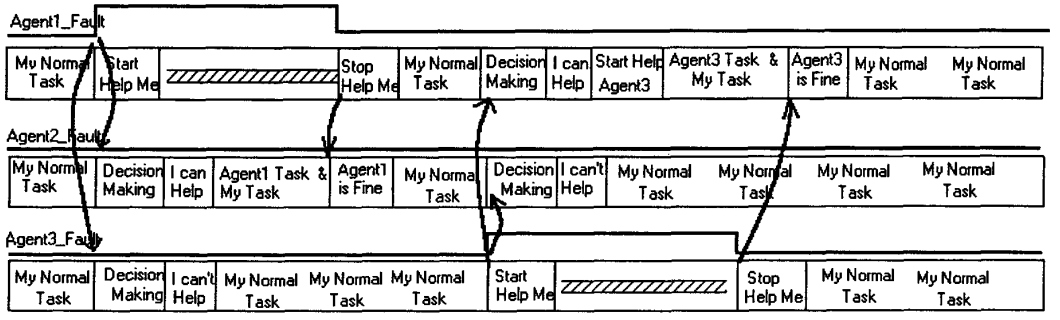

Figure 4: *Internal states of the agents when fault occurs in Decision-Making based Help (Scenario 2)*

Table 5: *Simulation Results when the agents help others with a decision- making- based method (Using Scenario2).*

Agent	Criticality	Number of Successfully done Tasks with at most 60 ns Delay	Number of Lost Tasks	Fault Duration	Number of Tasks done by Itself	Number of Tasks helped by Others	Total Number of tasks to be performed
1	1	58	6	7500 NS	19	45	64
2	1	67	0	0 NS	67	0	67
3	1	57	3	2000 NS	49	11	60

Table 6: *Performance evaluation*

	Without help	Fixed criticality based help	Decision-making based help
Scenario 1	56%	82%	90%
Scenario 2	69%	87%	95%

Table 7: *Lost Tasks Evaluation*

	Without help	Fixed criticality based help	Decision-making based help
Scenario 1	47%	24%	10%
Scenario 2	30%	17%	4%

VI. Conclusions and Future Works

These experiments show that using a more complete decision making mechanism is necessary. In this system and any other multi agent system which is designed to utilize the help capability, the designer has to consider different parameters to make a powerful activation function for help processing. Using a fixed criticality-based method and obliging the agents to help under fault conditions regardless of their own time and capability constraints is not proper for some applications.

In our future research, we intend to study some more effective decision-making method for the agents to process the help request.

Implementing the introduced methods on a FPGA-based distributed system is the next step of this study.

VII. References

[1] Bary Johnson, Design and Analysis of Fault Tolerant Digital Systems, Adison Wesely, 1989

[2] N. R. Jennings, "Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems using Joint Intentions", Artificial Intelligence 75(2), pages 195-240, 1995.

[3] S. Hugg, "A Sentinel Approach to Fault Handling in Multi-Agent Systems", Proceedings of the 2nd Australian Workshop on Distributed AI, Cairns, Australia, 1997.

[4] M. Klein and C. Dellarocas, "Exception Handling in Agent Systems", Autonomous Agents '99, Seattle, 1999.

[5] G. A. Kaminka and M. Tambe, "What is wrong with us? Improving Robustness through Social Diagnosis", Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98), 1998.

[6] K. Decker, K. Sycara, and M. Williamson. Matchmaking and Brokering. Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96), Dec-96.

[7] Cesar Ortega and Andy Tyrrell, "Biologically Inspired Fault-Tolerant Architectures for real-time Control Applications", Control Engineering Practice, pp. 673-678, July 1999.

[8] D.W. Bradley and A. M. Tyrrell , "The Architecture for a Hardware Immune System", *in proceedings of 3rd NASA/DoD Workshop on Evolvable Hardware,* Pages 193-200, Long Beach, California, USA, July 2001

[9] Foad Ghaderi, Majid Nili, "Fault-Tolerance in Cooperative Robots Using others' Helps", 7th Iranian Computer Conference, pp. 220-227, Feb. 2002.

[10] Majid Nili Ahmadabadi, Foad Ghaderi, "Distributed Cooperative Fault Tolerance In A Team Of Object Lifting Robots", Accepted in IROS2002, October2002

[11] L. E. Parker, "ALLIANCE: An Architecture for Fault tolerant Multi-robot Cooperation", IEEE Transaction on robotics and automation, vol. 14, No. 2, pp. 220-240, April 1998

[12] "IEEE Standard VHDL Language Reference Manual", ANSI/IEEE Std 1076-1993, IEEE Inc., 1993.