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 form 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.

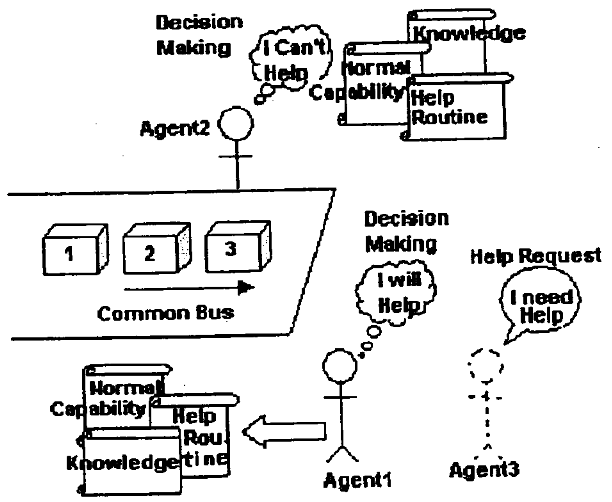

Figure 1: *The Agents should contain normal operations, Help procedure and decision making capability.*

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.

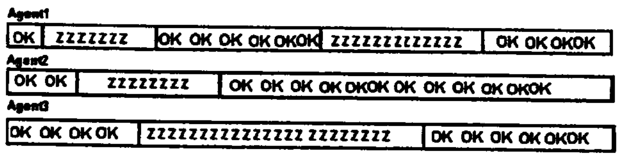

Figure 2: *The Test Scenario used for all Experiments*

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 agnet 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.*

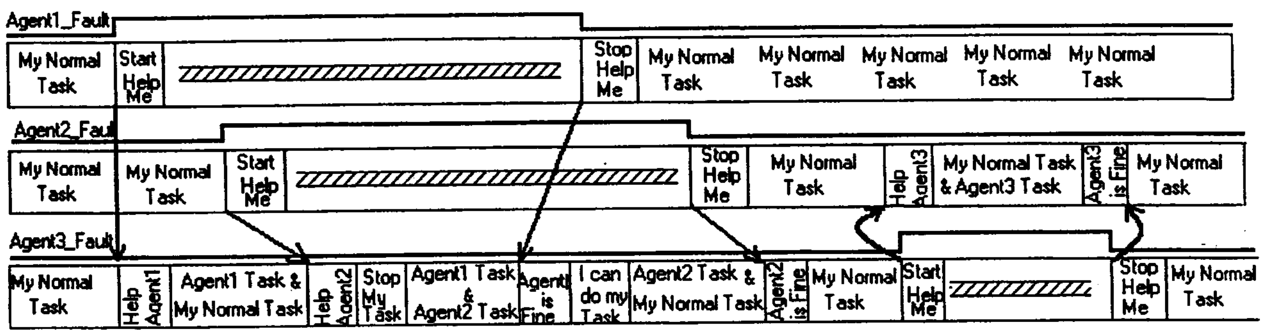

Figure 3: *Internal states of the agents in the strategy based on the Agents' Criticality (No Decision Making).*

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.*

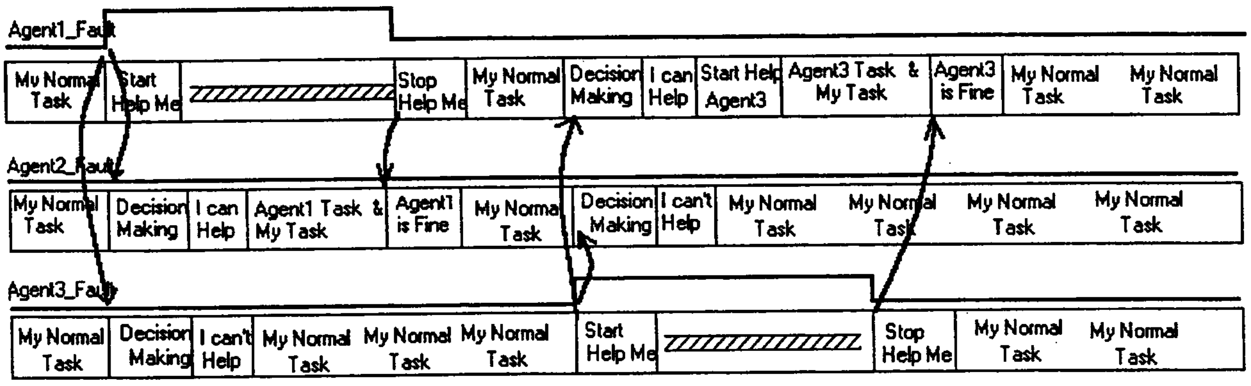

Figure 4: *Internal states of the agents in Decision-Making based Methods (Last Three Experiments)*

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.*

5.2 Decision-Making for Help based on Remaining Time

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 and if they should help which one to be helped first in the case of more than one faulty agent in the system. 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. The agents' internal states during this experiment are shown in Figure 4.

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:

$$\begin{gathered} \mathit{Available\_Time} \\ > \\ \mathit{My\_Task\_Completion\_Time} \\ + \\ \mathit{Agent\_i\_Task\_Completion\_Time} \\ + \\ \mathit{Agent\_j\_Task\_Completion\_Time} \end{gathered}$$

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

$$\begin{gathered} \mathit{Current\_Task\_Time\_Available} \\ > \\ \mathit{My\_Task\_Completion\_Time} \\ + \\ \mathit{Agent\_i\_Task\_Completion\_Time} \end{gathered}$$

or

$$\begin{gathered} \mathit{Current\_Task\_Time\_Available} \\ > \\ \mathit{My\_Task\_Completion\_Time} \\ + \\ \mathit{Agent\_j\_Task\_Completion\_Time} \end{gathered}$$

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 the other agents do not help it. 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.

The performance evaluation according to the given performance index, results in 4% improvement in comparison with the system uses fixed-criticality-based method (shown in Table 2). Besides, The number of lost tasks decreases from 33% to 15%.

5.3 Risking to Provide Help Using *First Come First Served* Strategy

In this experiment, agents have no pre-knowledge from the coming rate of their own data and they are unable to make a time-based decision, so they use the policy of First Come First Served and actually they risk providing help.

If the Frame Generator produces packets with different rate for each agent, the described situation will be simulated practically. To make the policy more clear, here is a scenario: If Agent 3 has to help Agent 1 and Agent 2, it first completes its own task and if its new data has not come in yet, it starts to help to the agent whose request for help has came before the other. When did it, if the help request from the other agent remains active and it has not been expired yet, it will help that agent.

Here the acquired performance is 75%. The simulation results of this experiment are shown in Table 4.

5.4 Risking to Provide Help Using *Shortest Job First* Strategy

In this experiment like the previous one, agents have no estimation of their remaining time, so they have to consider other parameters in their decision-making. Here they consider the amount of time required for completing one task and select the first agent to be helped. For example if Agent 1 has to help Agent 2 and Agent 3, it does this scenario: First does its own task and then starts to help Agent 3 since its task is shorter for it to do than that of Agent 2.

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	18	67	44	23	85	5300 NS
2	1	23	74	60	14	97	6000 NS
3	1	30	73	42	31	103	4000 NS

Table 4: *Simulation Result when the agents help others with the First Come First Served Strategy.*

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	14	71	47	24	85	5300 NS
2	1	23	74	59	15	97	6000 NS
3	1	32	71	42	29	103	4000 NS

Table 5: *Simulation Result when the agents help others with the Shortest Job First Strategy.*

Strategies	Without help	*Fixed criticality based*	Decision Making based on *Remaining Time*	Decision-making based on *First Come First Served*	Decision-Making based on *Shortest Job First*
Performance	59%	80%	**84%**	75%	76%
Lost Tasks	40%	33%	**15%**	24%	23%

Table 6*: Evaluating four presented strategies*

When it is completed, if the help request from Agent 2 remains active and the expiration time of the task is not reached, it starts helping.

The time takes for one agent to complete the task of another agent is assumed to be agent-dependent. It means since one agent completes the task of another agent by its own capabilities, it depends on its processor and internal resources and may be different from the time another agent spends to do the same task. The performance of the system using this strategy is computed as 76%. There is no considerable improvement in comparison with First Come First Served strategy, since both of them risks for help and there is no guarantee that they will not miss their own data while providing help. The detailed simulation results of this experiment are shown in the Table 5.

Considering all these strategies, Table 6 summarizes the results. It shows that as the method becomes more flexible, the number of lost tasks decreases and the performance increases considerably. It also shows that if the agents are aware of the remaining time and take a time-based decision, they are more successful. This strategy is not impractical because in most of the real control applications, the rate of sensory input data for each agent is not unpredictable and is a primary knowledge for them, so they can count on it and take a more accurate decision.

6. 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. Risking and starting an action with the hope of success, may be neither practical nor wise unless help provision is the primary goal of the designer under any situation.

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 FPGA-based distributed system is the next step of this study.

7. 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 Intelligenc* 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] Hitomi Kasahara, Keiki Takadama, Shinichi Nakasuka, Katsunori Shimohara, *Fault Tolerance in a Multiple Robots Organization Based on Organizational Learning Model*, IEEE-SMC'98

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

[9] D.W. Bradley and A. M. Tyrrell , "The Architecture for a Hardware Immune System",

[10] Foad Ghaderi, Majid Nili, "Fault-Tolerance in Cooperative Robots Using others' Helps", Accepted in 7th Computer Conference of Iran 2002.

[11] Majid Nili Ahmadabadi, Foad Ghaderi, "Distributed Cooperative Fault Tolerance In A Team Of Object Lifting Robots", Submitted to IROS2002, May2002

[12] L. E. Parker, "ALLIANCE: An Architecture for Fault tolerant Multirobot Cooperation", *IEEE Trasaction on robotics and automation*, vol. 14, No. 2, pp. 220-240, April 1998

[13] Zainalabedin Navabi, Analysis and Modeling of Digital Systems, McGraw Hill, Second Edition, 1998