Proceedings of the 9th International Conference on Neural Information Processing (ICONIP'02), Vol. 4
Lipo Wang, Jagath C. Rajapakse, Kunihiko Fukushima,
Soo-Young Lee, and Xin Yao (Editors)

A DECISION-MAKING BASED APPROACH FOR FAULT-HANDLING IN MULTI-AGENT SYSTEMS

Maryam S. Mirian[1], Majid Nili Ahmadabadi[2], Zainalabedin Navabi[3]
[1] Robotics and AI Lab, [2] Control & Intelligent Processing Center of Excellence, [3] Hardware &CAD Lab
Dept. of Elect. and Comp. Eng., Faculty of Engineering, University of Tehran
mirian@ut.ac.ir mnili@ut.ac.ir navabi@ece.neu.edu

ABSTRACT

This paper focuses on distributed fault recovery in agent-based systems by providing help for faulty members. In the presented method, if one faulty agent requests for help or agents are informed of fault in one of their teammates, they first decide if they are able to help or not. In the case that they are able to help and several help requests exist, helper agents specify a sequence of help actions through another distributed decision-making phase.
The introduced fault clearing method is totally distributed in the sense that each helper agent makes its decisions by itself and no central or special agent exists in the system. In fact, the decision making process and the required information are designed such that the agents cooperate implicitly to prevent the system performance loss.
The developed ideas are implemented in a simulated Distributed Control System. As it is shown, the proposed distributed fault-clearing method through reconfiguring the agents' roles is very effective.

1. INTRODUCTION

Distributed systems can be considered as teams of distributed cooperative agents. This idea encourages a designer 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 create an efficient multi-agent system is providing it with sufficient capabilities to operate properly even if there are some faults in the system. 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 MAS and isolate it. This paper discusses the use of the task performing agents to recover the potential faults of the system by assigning some new roles to the agents, when a fault occurs, to recover the lost system capabilities. In the presented method, there is no extra or central agent, sentinel or broker to observe the agents and redistribute the tasks among them 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.
In [13] we described the fixed criticality strategy, which is time independent and just considers defined criticalities for each agent. [14] declares the strategies that consider neither criticality nor time. These policies are called risking strategies, since the agents have no estimation of their remaining time while they asked for helps. Therefore, they help with the hope of success and their own work interrupts their help action whenever it comes. As a continuation, [15] discusses a help mechanism used in agent-based systems with similar criticality coefficients and different impatience values. This impatience value depends on the agent's characteristics and conditions.
This paper describes the strategies designed for the agents with different criticality coefficients. The difference in relative criticalities affects the willingness of each agent on providing help. Strategies focused in this paper are practically much more effective than fixed criticality based and risking methods.

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. As described in [10], 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 [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. 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.

3. THE APPROACH

Reconfiguring the roles of agents and their capabilities, due to the requested type of help can be used for fault recovery. A simplified overview of the system is shown in Figure 1 and described in details in this section.

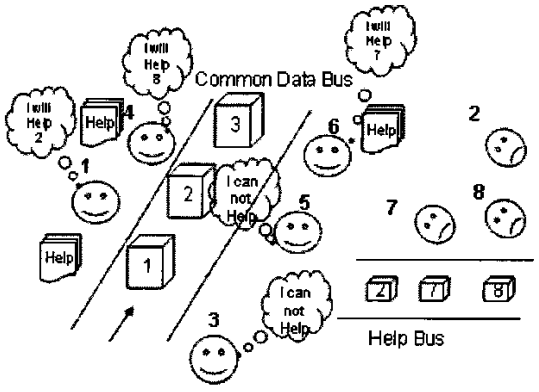

Figure 1. The Agents contain normal abilities, defined help routines and a decision making capability.

3.1 The Agents Structure and 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, rate of coming data and the criticality of tasks compose the primary knowledge of the agents. It is also assumed that each agent is capable of performing some of the other agents' tasks.

3.2 The Help Request

The help request must be as short as possible and also contains the required information, such as agent' ID and the type of required help. 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.
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.

3.3. Two-Phase Decision-Making Summary

In the following sections, developed decision making ideas are introduced. The first phase is designed to decide if the helper agent is willful to provide help. The results of the second phase specify the sequence of helping actions to the selected faulty agents.

3.3.1 Phase 1: Competition on Giving Help

When a help action is required, it is very likely that there is more than one healthy agent in the system. Therefore,

the competition problem on giving help starts. This is solved by considering an additional phase of decision-making. This phase computes a propensity value that represents the tendency of each agent to give help to others. This value depends on factors listed below:

- Inverse relation with the amount of time remains if the help action is done.
- Relation with the number of recent help actions to the designated faulty agent
- Relation with the number of faulty agents in the system.
- Relation with the Criticality of the Faulty Agent.
- Inverse Relation with the Criticality of the Helper Agent.

If the agent has much more time than needed, then it is not wise to make it busy with a small task. It is preferred to let it be free for the probable longer next help actions. Although, there is no explicit learning in the system, number of help actions is considered to show if one agent has helped many times to another agent before, it may be more expert alternative to help. This property is virtually simulated by spending less time when an action has been performed before and more time if it is being done for the first time. By considering the number of faulty agents in the system, the helper can estimate the system capability on fault recovering. If there are many faulty agents in the system, the agent's stimulus for help is amplified. The agent's criticalities are considered to show that more critical agents must be helped first.

The willingness factor is computed by all healthy agents and based on this value a winner is motivated. An adaptive threshold is also tuned to be considered in the second decision-making phase. If this value is very high or very low, the threshold is adjusted automatically. If more than one agent, are eager to help, the first one who starts help, stops the other from helping. Figure 2 basically demonstrates what happens in the system when a fault occurs.

3.3.2 Phase 2: Selecting a Set of Faulty Agents to Help and Decide about the Sequence of Help Actions

In this phase, those agents their willingness to help exceeds the threshold, are ready to decide which agents should be helped and in what order. In order to specify the order of helps to faulty agents, a helper may consider one of the scheduling strategies specified in the design time:

- ✓ Best Fit
- ✓ First Come First Served
- ✓ Shortest Job First

These strategies are described in details in section 6. Based on these strategies, a sorted list of agents to be helped is created and from the top of the list, a helper starts to give service. From the list, *m* agents with total task completion time less than Available Time is selected and performed.

From *n* Help Requests sorted based on the strategies, the helper will give service to first *m* of them until the available time finishes:

$$Available\_Time > My\_Task\_Completion\_Time + \sum_{i=1}^{m} Faulty\_Agent_i\_Task\_CompletionTime$$

Where *m* is a number less than *n* that shows the number of those faulty agents can be serviced in the available time. 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 *My_Task_Completion_Time* is not fixed and when it shortens gives the agent the opportunity to help.

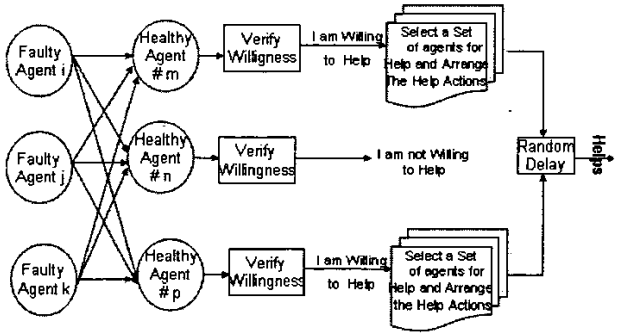

Figure 2. Multiple Faulty Agents, Multiple Helper Agents to Decide Who to Help and in What order

4. 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 a common bus. Thinking about a real application, 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 [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 the strong features of VHDL like concurrency aspects, design hierarchy, timing control in all levels and many other benefits.

5. EVALUATION TECHNIQUE AND TEST CASE

In order to evaluate introduced methods, a simple performance formula is defined:

$$Performance = \frac{\sum_{i=1}^{Agents\_No} n_i C_i}{\sum_{i=1}^{Agents\_No} N_i C_i}$$

Where *Agents_No* is the number of agents in the system, n_i is the Number of successfully performed tasks of $Agent_i$, C_i is the Criticality of $Agent_i$ and N_i is the total number of assigned tasks to each agent during the experiment.

It is notable that a task may be lost because the agent is faulty and it is not helped. This is either because of the others unwillingness to help or because the timing constraint does not let the other agents to help. The applied test case is shown in Figure 3. "Z" shows the fault case and "OK" demonstrates the normal situation. As it is shown the simulation is done for 4200 nano seconds and the total number of tasks for each agent is 210. The data coming rates for all agents are equal and constant.

6. STRATEGIES AND SIMULATION RESULTS

In this part, different strategies are described in more details.

In this set of experiments, in contrast with the experiments done in [15], more critical agents are more important than less critical ones for the system evaluation.

6.1 Best Fit Policy

In this experiment, the helper will help the agent with the longest task completion time that best fits in the available time. Choosing this task to perform, it will be more probable that the other agents can help the remaining faulty agent. As Table 1 demonstrates, the performance is 80%. The last column of the table shows the results if there is no help mechanism in the system.

6.2 First Come First Served Policy

In this strategy, the first request for help is responded first and the next requests are answered as their reception order specifies.

The simulation results are summarized in Table 1. This strategy shows almost 3% losses in system performance in comparison with Best Fit Policy.

6.3 Shortest Job First Policy

In SJF the agents consider the amount of time required for completing one task and select the agent with the shortest job from a queue as the first candidate for help. As Table 1 represents, there is 19% performance loss in comparison with Best fit strategy.

7. CONCLUSIONS AND FUTURE WORKS

These experiments show that using a 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 [14]. 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 [15]. From the proposed scheduling strategies, Best Fit seems to be the most effective one that improves the system performance from 57% in without help case to 80%.

In our future research, we intend to study some more effective decision-making method for the agents to process the help request. Evaluating MAS with the serial / parallel structure and generalizing the help method to cover such systems would be considered.

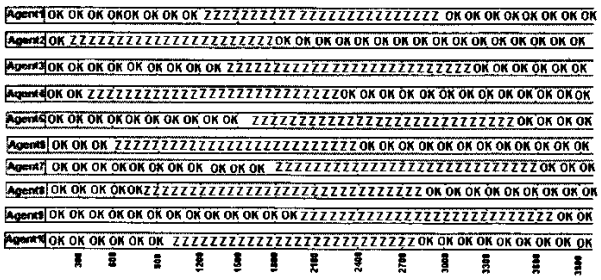

Figure 3. The Test case Scenario

Table 1. Successfully Performed Tasks done of Each Agent and the Performance Evaluation

	Criticality	BF	SJF	FCFS	Without Help
Agent1	1	120	126	120	120
Agent2	2	121	179	186	120
Agent3	1	120	119	120	120
Agent4	3	207	178	203	120
Agent5	5	160	118	160	120
Agent6	5	205	132	192	120
Agent7	4	180	119	160	120
Agent8	5	152	120	160	120
Agent9	5	170	119	120	120
Agent10	5	170	117	170	120
Performance	-	**0.80**	**0.77**	**0.61**	**0.57**

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

[10] Foad Ghaderi, Majid Nili Ahmadabadi, "Distributed Cooperative Fault Tolerance In A Team Of Object Lifting Robots", *Accepted in IROS2002, October 2002*

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

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

[13] Maryam S. Mirian, Majid Nili Ahmadabadi, Zainalabedin Navabi, "A New Task Redistribution Method for Fault Clearing in Multi-Agent Systems", *Accepted in SMC2002, Tunisia, October2002*

[14] Maryam S. Mirian, Majid Nili Ahmadabadi, Zainalabedin Navabi, "Evaluating Task Redistribution Methods for Fault Clearing in Multi-Agent Systems", *Accepted in ERSA'02, Las Vegas, USA, June2002*

[15] Maryam S. Mirian, Majid Nili Ahmadabadi, Zainalabedin Navabi, "Agent's Role Reconfiguration Based on Decision-Making for Distributed Fault Recovery in Multi Agent Systems", *Accepted in AIM'02 Workshop of EURASIA-ICT 2002, Shiraz, Iran, October 2002.*