Towards a Fault-Tolerant Multi-Agent System Architecture

Sanjeev Kumar
Oregon Graduate Institute
20000 NW Walker Road,
Beaverton, OR 97006, USA
+1-503-748-7803
skumar@cse.ogi.edu

Philip R. Cohen
Oregon Graduate Institute
20000 NW Walker Road,
Beaverton, OR 97006, USA
+1-503-748-1326
pcohen@cse.ogi.edu

ABSTRACT

Multi-agent systems are prone to failures typical of any distributed system. Agents and resources may become unavailable due to machine crashes, communication breakdowns, process failures, and numerous other hardware and software failures. Most of the work done in fault handling in multi-agent systems deals with detection and recovery from faults such as state-inconsistencies, relying on the traditional techniques for recovering from other distributed systems failure. However, the traditional fault-tolerance techniques are designed for specific situations and they require special infrastructural support. We argue for fault-tolerance techniques that can be readily implemented using a generic agent shell with minimal or no modification to the agent infrastructure.

We propose that theories from multi-agent systems literature can be effectively combined with basic fault-tolerance principles to design robust multi-agent systems. In particular, we argue that (1) teamwork may be used to create a robust brokered architecture that can recover a multi-agent system from broker failures without incurring undue overheads, (2) teamwork may also be used to guarantee a specified number of brokers in a large multi-agent system, and (3) agent autonomy can be used to prevent thrashing and guarantee acceptable levels of quality of service by an agent. We also describe the Adaptive Agent Architecture (*AAA*), a fault-tolerant brokered multi-agent system architecture, and present experimental evidence using the AAA to validate our approach.

Keywords

Multi-agent systems, fault-tolerance, teamwork, autonomy.

1. MOTIVATION

Multi-agent systems are prone to the failures that can occur in any distributed software system. An agent may become unavailable suddenly due to various reasons. The agent process may die due to unexpected conditions, improper handling of exceptions and other bugs in the agent program or in the supporting environment. The machine on which the agent process is running may crash due to hardware and software faults. Network partitioning may also make agents unavailable for unforeseen periods.

Various techniques have been developed in traditional distributed systems to deal with recovery from such failures. As we shall see in section three, these techniques are meant for specific failure situations and they require special infrastructure support. For instance, the techniques of object group replication [2], virtual synchrony [2] and N-version voting [3] need prescribed mechanisms for communication and synchronization among the replicas. Therefore, a multi-agent system intending to use any of these techniques must be built using an infrastructure that provides support for these techniques. It may not be possible for a multi-agent system to use these powerful fault-tolerant techniques without extensively modifying the agent architecture. On the other hand, a technique based on the multi-agent system concept of teamwork [18], may be implemented by simply adding a plan to the plan library of a generic agent.

If the broker with whom an agent was registered becomes unavailable, the agent may look for another broker using transport level broadcasting or multicasting. However, this mechanism cannot be relied upon unless reliable multicasting is used. The reliable multicasting techniques require special protocols and architectures [22] that must be supported by the agent infrastructure. On the other hand, a technique based on teamwork to ensure connectivity with another broker may be readily implemented using a generic agent shell. Agents with specific behavior can be created from a generic agent shell by adding plans and actions to its plan and action libraries respectively.

It is apparent, therefore, that we do need fault-tolerance techniques designed for multi-agent systems. The focus of this paper is to investigate fault-tolerance techniques based on multi-agent system concepts. Thereafter, subsequent research can investigate the ease of implementing these techniques using generic agent shells.

Multi-agent systems require brokers[1] or middle agents for certain tasks such as accepting requests, locating capable agents, routing requests and responses, sharing of information, managing the system, registering agent capabilities, and for legal purposes as an independent third party [13]. As a result, a large number of multi-agent infrastructures such as OAA [5], RETSINA [10], JATLITE [14], and Infosleuth [21] provide some kind of middle agents or at least some form of facilitation and routing service. This observation motivates us towards designing fault-tolerant brokered multi-agent system architectures. Moreover, our experiences with Quickset [9], a multi-agent system based on

[1] The term *broker* has a special connotation in multi-agent systems [20]. However, in this paper we will use the term broker to be synonymous with middle agent, as the work described in this paper may be extended to most middle agents.

OAA, reinforces the need for an agent architecture that can recover quickly from broker failures.

As a brokered multi-agent system becomes more complex and as the number of agents increases, the number of brokers in the system will also increase due to reasons such as domain specialization and efficiency. We note that in a large multi-agent system with multiple brokers, the brokers could potentially serve as backups for each other, thus achieving a high level of fault-tolerance. However, instead of using warm and hot backups, N-version voting and other traditional techniques that build upon redundancy, we propose a scheme that uses teamwork as the distributed interaction protocol among these brokers.

It has been shown that load balancing by brokers in multi-agent systems improves the performance of the system [10]. However, we observe that a malicious or faulty agent can still bring down a brokered multi-agent system by continuously sending requests at a rate much higher than what the broker can handle. We believe that autonomous agents should be less susceptible to external influences and this motivates us to investigate exploiting an agent's autonomy to protect against such problems.

The above discussions indicate that we need (1) fault-tolerance techniques designed for multi-agent systems, (2) robust brokered architectures that can recover from broker failures, and (3) agents that are robust from fault-causing influences of other agents.

2. OVERVIEW

We review the work done in the area of fault tolerance in multi-agent systems in the next section. Two divergent approaches that have been used to diagnose failures and attempt recovery are (1) using sentinels, external to the agents, that monitor inter-agent communication, and (2) using introspection to monitor an agent's own run time behavior. We also briefly review some of the important fault-tolerance techniques used in the database and traditional distributed systems and observe that redundancy is the basic principle behind most of those techniques.

In section four, we briefly introduce the Adaptive Agent Architecture (*AAA*), a fault-tolerant multi-agent system architecture that is currently under development and forms the basis for most of this paper.

Thereafter, in section five, we show that a multi-agent system can recover from broker failures if the brokers form a team with certain commitments and the agent architecture enables the brokers to honor those commitments. This recovery scheme based on teamwork avoids the overhead of using redundant brokers just for the purpose of fault-tolerance. We also show that this technique does not appreciably interfere with the normative behavior of the multi-agent system and that a multi-agent system using this technique will continue to function as long as there is at least one broker left in the system.

We propose an extension to the above teamwork based technique in section six, wherein it can be specified that the system needs at least N brokers at all times for its functioning. We discuss our current implementation where it is possible to have at least two brokers at all times, irrespective of broker failures, and compare it with the classical technique of regenerative processes.

Thereafter, in section seven, we argue that autonomous brokers are more fault-tolerant than non-autonomous brokers. In particular, we show that brokers should be able to refuse requests so as to prevent thrashing and buttress our argument with experimental evidence. We discuss the future work in the direction of this paper in section eight and finally, in section nine, we conclude with a summary of the present work.

In this paper, we do not address the detection of unavailability of an agent (or broker) due to network, machine, and process failures and rely on TCP mechanism and timeouts for this purpose.

3. REVIEW OF RELATED WORK

Here we review some of the main approaches to fault-tolerance in multi-agent systems as well as in the traditional distributed and database systems.

3.1 Fault Handling in Multi-agent systems

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 [15]. This approach is similar to ours in that both approaches are based on the theory of teamwork. However, we explicitly address the problem of fault-tolerance whereas Jennings work is more focused towards cooperative problem solving.

Hägg uses external sentinel agents to monitor inter-agent communication, build models of other agents, and take corrective actions [12]. 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. In our teamwork-based approach, the problem solving agents themselves participate in fault-tolerance as opposed to the external sentinel agents used in this work. Further, Hägg's work does not use middle agents whereas the current work mainly focuses on recovery from failures of middle agents. Lastly, the sentinels in this approach analyze the entire communication going on in the multi-agent system to detect state inconsistencies. However, this approach is not realistic for systems such as Quickset [9] that we use due to the high volume and frequency of messages in these systems.

Klein proposes to use exception-handling service to monitor the overall progress of a multi-agent system [17]. Agents register a model of their normative behavior with the exceptional-handling service that generates sentinels to guard the possible error modes. The exceptional-handling services use a query and action language to interact with the problem solving agents to detect and diagnose faults and take corrective actions. The exception-handling service is a centralized approach whereas our teamwork-based approach is essentially a decentralized approach. Moreover, Klein's approach relies on being able to communicate with the agents whereas the current work attempts to restore connectivity when communication with a broker is not possible.

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 failures [16]. 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. This work is similar to the current work in that the teamwork model is used in both cases. However, we mainly concentrate on middle agents whereas Kamnika's work is related to a system that is not based on middle 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 [10]. 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. These approaches compliment our work and they can be used along with our teamwork-based techniques.

3.2 Traditional Fault-Tolerance Techniques

A large number of techniques for fault-tolerance can be found in the traditional database and distributed systems literature. Figure 1 lists some of the techniques that have been developed for database recovery, for application recovery and recovery of distributed systems.

Most of these recovery methods primarily focus on replication techniques that permit critical system data and services to be duplicated as a way to increase reliability [2]. Active replication is also used for processes wherein the inputs are duplicated and the outputs produced are consolidated. Most of the methods used for application recovery advocate either logging the application messages or frequently saving the application state and therefore, require either a database or a recoverable queue [19]. The current work attempts to recover a multi-agent system without recovering an inaccessible broker process and so does not require the brokers to save their state to persistent storage.

Considering the specific problem of recovering a multi-agent system from broker failure, the closest traditional techniques that may be applicable are the techniques of warm and hot backups and the object group replication technique used in conjunction with virtual synchrony. In the warm backup technique, a process is replicated and when the main process goes down, the replica immediately starts recovering to the last known state of the dead process. A hot backup is similar to a warm backup except that the input and output of the main process as well as the replica are synchronized at all times and so the replica can immediately take over without having to first bring itself to the state of the dead process [1].

Database Recovery:

Redo-undo Logs, Fuzzy and Basic Checkpointing, Database Replication

TP Monitors, Application Servers, Resource Managers:

Recoverable Queues, Pseudo-conversations, Fault-tolerant Input Logging, Checkpointing based Recovery, Transaction based Recovery, Stateless Servers, Warm Backups, and Hot Backups, Regenerative Processes

Fault-Tolerant Distributed Systems:

Object Group Replication + Virtual Synchrony, Message Logging, N-Version Voting

Figure 1: Traditional Fault-Tolerance Techniques

The object group replication used with virtual synchrony is essentially the same as the hot backup technique except that here objects are replicated instead of active processes [2]. Groups of objects are treated as a single object and all objects in a group receive the same messages in the same sequence. Therefore, if we form a group of similar objects, there is a high probability that the different objects in a group will be in possibly different but correct states. So if one object fails or gets into some unforeseen problem, another object can take over the responsibility of responding to messages. A slightly different technique is that of N-version voting in which N independently developed modules from the same specification run in parallel and the result is decided by voting [3].

The above three categories of fault-tolerance require explicit replication for the purpose of fault-tolerance. These replicas are overheads in the sense that they exist only as backups and perform no useful task. These techniques also require infrastructural support to keep the process replicas synchronized or to implement object groups and virtual synchrony. The technique that we propose uses the brokers that are already present in the system and it uses teamwork to achieve an effect similar to warm backups and object groups plus virtual synchrony. Moreover, generic agents that already have reasoning and planning capabilities may be able to implement a technique based on multi-agent system concepts with minimal support (for example, by adding a plan and the corresponding actions to the plan and action libraries respectively) as opposed to the aforementioned techniques that require specific infrastructural support.

4. A ROBUST AGENT ARCHITECTURE

The Adaptive Agent Architecture (AAA) is a facilitated multi-agent system architecture under development and forms the basis of our research in fault-tolerance and agent communication languages. The agent library has been developed in Java and it uses Horn Clause for internal knowledge representation and reasoning. The AAA facilitator is a generic middle agent that serves as a broker, a matchmaker and a recruiter. Henceforth, in this paper, we will refer to the AAA facilitator as the AAA broker. The AAA brokers can be interconnected to form arbitrary networks and the agent library supports facilitated as well as direct inter-agent communication. The brokers as well as other agents can dynamically enter and leave AAA-based multi-agent systems. The support for fault-tolerance is built into the agent library. All of the experiments presented in the subsequent sections use AAA agents and the AAA broker. The AAA can also interoperate with the Open Agent Architecture [5].

5. RECOVERY FROM BROKER FAILURE

Here we discuss the teamwork-based technique used by the AAA for automatically recovering a multi-brokered multi-agent system from sudden broker unavailability. The broker under consideration may be inaccessible due to machine crash, network breakdown, or death of the broker process. We also make the simplifying assumption that the brokers in the system are fully connected. We first present the logical characterization of our teamwork model and briefly describe the steps in our recovery scheme. Thereafter, we walk through a recovery scenario describing the commitments involved and the actions taken by the agents.

5.1 Logical Characterization

Team activity has been explained in terms of the theory of joint intentions [6, 7, 8, 18]. This theory characterizes an agent's behavior in a team in terms of its internal state described in modal logic, linear time temporal logic, and dynamic logic of action. A joint persistent goal (*JPG*) formalizes the notion of joint commitment. The existence of a JPG between a group of agents is

a sufficient condition for the formation of a team with respect to that JPG. Two agents have a joint intention (*JI*) to do an action *a* if they have a JPG to do *a* while being in a particular mental state. A joint intention requires the starting mutual belief that the team members are going to do the jointly intended action next. Joint intention brings about one of the following mental states. (1) The agents mutually believe that *a* has been done; (2) They mutually believe that *a* is impossible; (3) They mutually believe that *a* is irrelevant.

We argue that the recovery of an AAA-based multi-agent system from broker failures is a consequence of the following specification of teamwork that is implemented by the AAA brokers:

$\models \forall y$ [(agent y) $\wedge$ (**DONE** (registered y T)?) $\supset$
(**JI** x1 x2…xn *a*(y) (registered y T))] where,
a(y) = (**WHILE** (registered y T)
DO [if $\neg$(connected y T) **THEN** (connect y T)])
T = team of brokers consisting of x1, x2,...,xn

It means that whenever an agent registers with the broker team, the brokers have the joint intention of connecting to the agent, if it ever disconnects, as long as it remains registered with the team. Using this proposition, along with other logical properties of the AAA, we can establish the commitments of the brokers in the team. These commitments result in fault tolerant behavior when the brokers act rationally and take appropriate actions to act on the commitments. Here, we state without proof, theorems about commitments of the AAA brokers. The basic theorems can be found in [8, 18].

Theorem 1: Whenever an agent registers or unregisters with a broker, the broker has a commitment to make this fact mutually believed by the broker team[2].

Theorem 2: When a broker discovers that an agent that is registered with the team is not connected, it has a commitment to make this fact mutually believed.

Theorem 3: When an agent that is registered with the broker team gets disconnected, the brokers have a joint commitment to connect to that agent. Moreover, all the brokers in the broker team have an individual commitment to connect that agent to the team.

Theorem 4: When a broker successfully connects to an agent that is registered with the broker team, but got disconnected, it has a commitment to bring about mutual belief about this fact.

*Theorem 5:*When a broker that was committed to the disconnected agent's being reconnected to the team, learns that the agent has been connected to the broker team, it gives up its commitment to connect to that agent.

The proof of these theorems along with a detailed logical characterization of AAA will be discussed in a subsequent paper.

[2] The sending of an inform that registration has taken place, followed by an acknowledgement that the prior inform was believed brings about mutual belief by default. Note that a circular data structure enables the representation of mutual belief compactly by referencing only two messages [4].

5.2 Recovery Scheme

The recovery scheme that follows from the above commitments consists of the following steps:

1) Each broker communicates the following information to its adjacent brokers:
   (1) The name and the address for direct connection (lets call it *dc-addr*) of agents that register with it.
   (2) The name of the agent that unregisters with it.
2) Each facilitator joins a recoverable broker team consisting of the adjacent brokers. The entire multi-agent system will consist of such teams with overlapping members.
3) Normally, an agent contacts some broker on startup, perhaps specified by the person starting the agent.
4) Agents listen for request from brokers at address *dc-addr* and they inform a broker about this address at the time of their registration with that broker. Brokers can initiate connection with agents for which they know the direct connection address.
5) When a broker disconnects from its teammates, all the brokers on its team attempt to directly contact the agents that were registered with the now disconnected broker.
6) When a broker successfully contacts an agent in this manner, it informs its teammates due to step (1). The other brokers give up their attempts to contact this agent directly.

The multi-agent system has recovered from failure of the disconnected broker when all the agents registered with that broker have been contacted in this manner. The requests that were in progress at the time of the failure, and hence could not be completed, may be sent again by the requesting agent.

5.3 A Recovery Scenario

We explain the recovery scenario from one of our experiments using the theory of teamwork. Figure 2 illustrates the initial system setup. The client agent periodically sends requests for which the distance agent is the only capable agent. The three brokers form a robust team as described earlier. This system can function only if both the client and the distance agents are registered with a broker.

From theorem 1, the brokers have an individual commitment to bring about mutual belief when an agent registers with a broker. Therefore, when the client agent registers with broker2, broker2 informs this fact along with the name and address of the distance agent to broker1 and broker3. Similarly, when the distance agent registers with the broker3, broker3 informs this fact to broker1 and broker2.

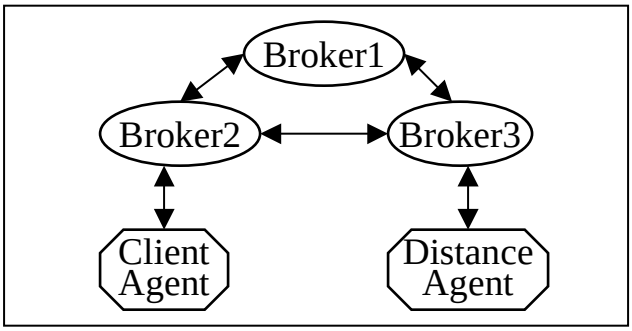

Figure 2: Setup for demonstrating the recovery process

In the middle of the experiment, we kill broker3. When a broker teammate is no longer accessible, the other brokers believe that all the agents registered with that broker are disconnected. When broker3 gets killed, at least one of the remaining brokers, broker1 or broker2, discovers that broker3 is no longer accessible to it and believes that the distance agent is not connected to the broker team. Therefore, from theorem 2, this broker has an individual commitment to bring about a mutual knowledge about its discovery. As a result, there would be some communication, in the general case, among the brokers in the team. However, each of the remaining brokers knows that the other was also connected to broker3 and believes that the other broker should have also discovered the fact about broker3 being no longer accessible[3]. Hence, no communication takes place at this point.

From theorem 3, the broker team has a joint commitment to connect to the agent that it mutually believes is disconnected from the team. Moreover, each of the remaining brokers has an individual commitment to contact the disconnected agent. The two brokers act rationally by attempting to contact the distance agent at the address given to them earlier by broker3. If the distance agent accepts registration request from one of the brokers, it refuses subsequent registration requests from other brokers. Figure 3 illustrates the situation when broker1 has successfully contacted the distance agent.

From theorem 4, broker1 now has an individual commitment to inform the successful connection of the distance agent to its teammates. As a result, broker1 will act rationally by communicating this information to broker2, and from theorem 5, broker2 will give up its attempt if it was still trying to contact the distance agent as the mutual goal has already been achieved. Moreover, from theorem 1, broker1 needs to inform the registration and address information of the distance agent to broker2. In the current AAA implementation, these two communication attempts are combined into one and just one message gets sent from broker1 to broker2.

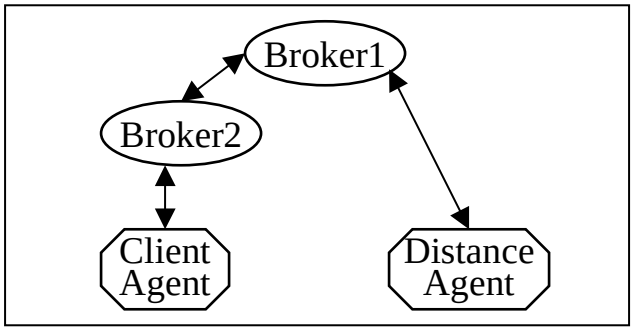

Figure 3: System after successful recovery

After successfully contacting the distance agent, the broker1 requests it for agent-specific information such as its capabilities. The multi-agent system has recovered from the failure of broker3 at this point and any request from the client agent that could not be completed due to the failure of broker3 may be sent again to the distance agent and the system continues to work.

[3] This belief will not hold in general. However, recall that the AAA brokers use TCP for detection of connection failures. Therefore, every broker that was connected to broker3 will eventually discover the fact that broker3 is not reachable.

5.4 Performance Characteristics

The experiments that follow and those in the subsequent sections were conducted on a 366 MHz Pentium™ II single-CPU machine, 128 MB SDRAM, 66MHz system bus, and running Windows™ NT 4.0 SP5. The machine was used as a standalone, disconnected from the network, and there was no application running on the machine other than the minimal operating system services and the agents and brokers used in the experiments. The experiments used JRE™ version 1.2.2 with just-in-time compiler enabled.

The experimental setup was same as that of the previous section with varying number of brokers fully connected to each other. The client agent sent synchronous requests that were answered by the distance agent. A constant delay of 20 ms was introduced in the distance agent before responding to requests so as to obtain observable response times. The client agent sent the requests 20 ms after receiving the reply to the previous request. The response time was measured as the real time elapsed between the sending of a request and the receipt of the corresponding response. The brokers used a policy wherein a request is sent to all the brokers even if a capable agent is found locally. This policy was used to prevent the response time from being biased by any particular configuration when the system reorganizes after a broker failure.

5.4.1 Effect of Recovery on Response Time

The experiment in figure 4 was started with eight fully connected brokers and the brokers were killed one by one. The stair-step plot shows the number of brokers present at any time. The agents mentioned near the peaks are the agents that were connected to the broker that was killed.

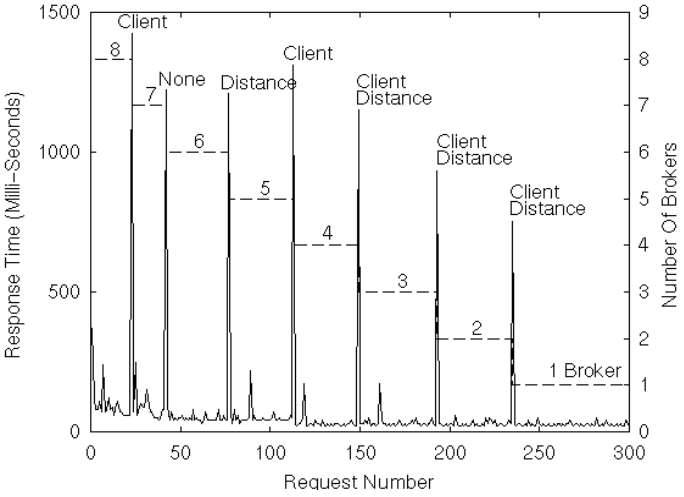

Figure 4: Effect on response time as brokers are killed

The plot in figure 4 clearly shows that *the AAA recovery scheme enables the system to function, despite broker failures, as long as there is at least one broker left in the system.* The peak response times are for the requests that were in progress when the broker got killed. The peak values go down as the number of brokers in the system decreases because (1) there is less team overhead in reorganizing a smaller team, and (2) the single-CPU system used in this experiment gets less loaded due to fewer processes.

5.4.2 Overheads of Using Teamwork

The establishment of joint persistent goals for teamwork requires communication overheads. However, the implementation of teamwork described above comes into play only when there is some transition in the system such as brokers or agents being added or removed. As such, we would expect that there should be no teamwork overhead in the steady state. Figure 5 shows two plots, one with the normal recoverable system, and the other with the teamwork code disabled. For each number of brokers, the response time was collected for 100 requests and the maximum, the minimum and the mean have been plotted.

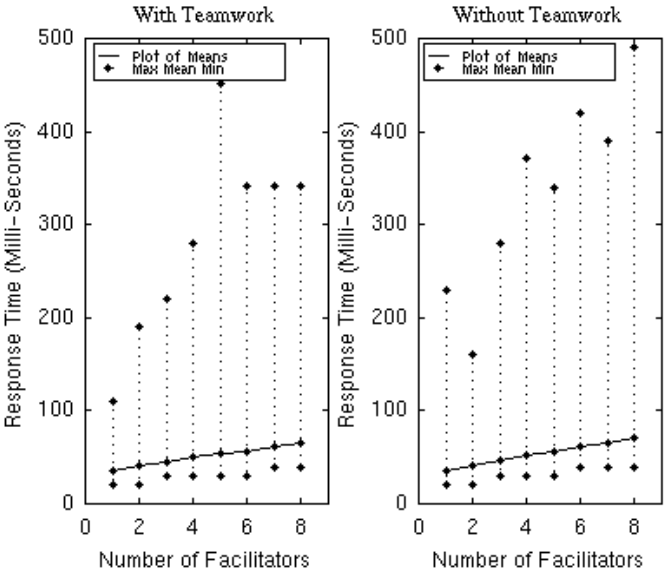

Figure 5: Mean response times with and without teamwork

We observe that a plot of the mean response times in the two cases will nearly overlap. A paired t-test of the mean response times for the same number of brokers, with and without teamwork, was unable to confirm a difference at 0.05 significance level. Therefore, we can say that *statistically the AAA implementation of teamwork for fault-tolerance does not present appreciable overhead in the steady state.*

5.4.3 Effect of Transitions on Response Time

Broker-team reorganization does affect the response times of the requests in progress at the time of the reorganization. The effect of adding additional brokers to a working system is expected to be similar to that when brokers are killed (assuming that the effects of the CPU getting overloaded due to additional processes and the disturbance due to process creation are accounted for).

When an agent enters or leaves the multi-agent system, a message is sent to all the broker teammates as prescribed by theorem 1, section 5.2. However, this message generates too little traffic to appreciably affect the ongoing agent conversations. The experiment in figure 6 consisted of four interconnected brokers and six additional agents. These additional agents did not participate in the interaction between the client agent and the distance agent. The system was run until completion and the six additional agents were killed one by one every second in this period. We observe that there is no appreciable and consistent peak in response time whenever an agent was killed. Moreover,

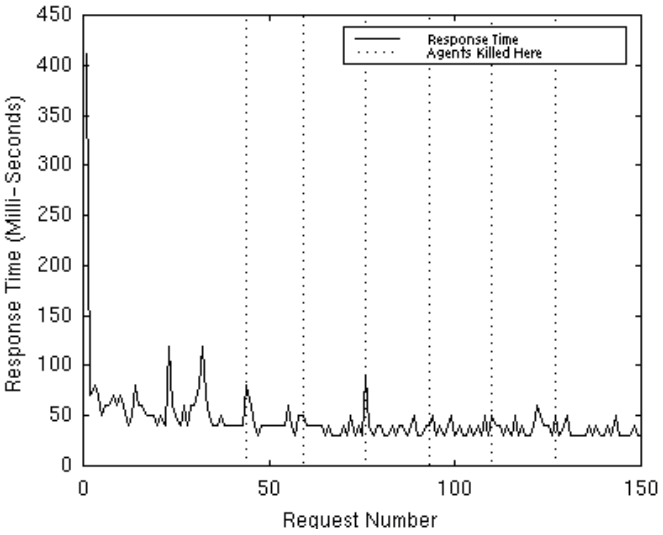

Figure 6: Response time when agents leave the system

small peaks if any, are within the bounds of random effects (the peak in response time that we see even before any agent was killed is due to random effects such as thread scheduling). This preliminary result meets our expectation that agents leaving (or entering) the AAA multi-agent system do not cause the teamwork implementation to create noticeable disturbance in the application.

6. MAINTAINING SPECIFIED NUMBER OF BROKERS

A large multi-agent system will typically use a number of brokers (or middle agents) to achieve an optimum between redundancy, resource utilization, efficiency and load balancing. Moreover, when a number of independent multi-agent communities are interconnected, it is generally desirable for each local agent community to have its own middle agents. The teamwork-based recovery scheme discussed in the previous section can be extended to have at least N brokers in the system at all times. When brokers fail or become inaccessible, new brokers can be started to maintain the specified number of brokers. Infrastructure support is required from the agent library so as to enable agents (including brokers) to start other brokers. Proper coordination is required among the brokers to ensure correct mutual beliefs, to track the progress of a recovery process, and to reorganize the agents after recovery.

The current AAA implementation can maintain at least two brokers at all times despite broker failures. The implementation of teamwork in this case is trivial because the teamwork code becomes effective only when there is one broker left in the system in which case the team consists of just one broker. The AAA agents commit to the AAA broker, with which they are registered, to honor its requests for starting broker processes. Whenever an agent registers or a broker teammate disconnects from an AAA broker, it checks to see if there are at least two brokers in the team. If not, it searches for an AAA agent on a different machine. If it finds such an agent, it requests that agent to start a broker and request the newly started broker to join the broker team of which

the requesting broker is a member[4]. If the AAA broker fails to find an agent on a different machine, it picks up an agent on the local machine at random and repeats the process. Further, an AAA broker started as a result of this process is committed to connect to (and form a team with) the broker that initiated its birth. The AAA broker also keeps track of pending requests to start brokers and may request another agent to start a broker if needed.

This technique resembles the technique of regenerative processes in the traditional fault-tolerance literature wherein a critical process can be restarted by a monitoring process upon failure. However, there are a few major differences between the two techniques.

(1) If the monitoring process fails, there needs to be another level of monitoring process to restart the first monitoring process. This can go up to any level but all these levels have to be explicitly designed and configured for each machine. In the AAA scheme, no separate configuration is needed for each machine. Moreover, all of the requisite N-1 brokers can be started even if there is just one broker left in the system, thereby automatically providing N-1 levels of monitoring.

(2) Special monitoring infrastructure is required to be able to start processes on different machines. A convenient way would be to have separate monitoring processes on each machine that are coordinated using a special distributed algorithm. However, any such algorithm needs proof of correctness before it can be relied upon. In the AAA scheme, no separate monitoring infrastructure is required as the problem solving agents themselves participate in the fault-tolerance process. The specification of teamwork provides a distributed coordination protocol that is logically proven to work.

The general case in which N brokers can be maintained at all times is under implementation. A subsequent paper will provide the detailed logical analysis for this technique.

7. AUTONOMY AND ROBUSTNESS

Autonomous agents act in pursuit of their own agenda. They evaluate their internal goals and beliefs and the consequences of their action even when serving the requests of other agents. As a result, we would expect an autonomous agent to be less susceptible to the influences of other agents. A weaker notion of autonomy is the ability to refuse requests [11]. We show that even this simplistic notion of autonomy can prevent process thrashing and hence protect against performance degradation beyond a point where an agent effectively becomes useless.

The experiment in figure 7 consisted of a base case in which asynchronous requests were sent to the broker every 200 ms that then were forwarded to a capable agent and the answers were returned back. The mean response time was calculated to be 34.4 ms. In the second case, the client agent aggressively sent asynchronous requests every 17 ms (half the mean response time). This rate of sending requests was beyond what the broker could handle and as a result, the response time increased without bounds. A performance analysis tool revealed that the memory usage increased linearly with a sharp slope as the message got queued up (the memory usage graph was linear since all the requests were of nearly the same size in bytes). The broker was then setup to refuse new requests when the agent capable of answering the request was busy and as a result, the response time can be seen to oscillate around the base case. The experiment was repeated with the broker set to refuse alternate requests and the results were identical.

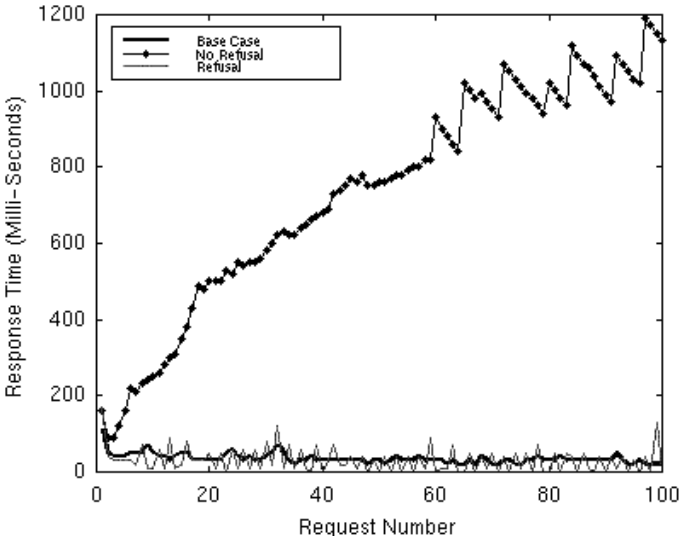

Figure 7: Effect of refusal on performance

These simple experiments conclusively show that a malicious or faulty agent or an aggressive agent can almost bring down a multi-agent system unless the agents concerned start refusing messages. The queueing up of messages for future processing, without any limit on the number and size of messages that can be queued, appears to be the main cause of this problamatic behavior. Even a simple policy, such as refusing messages when the input queue length grows beyond a certain limit, can prevent an agent from thrashing and hence from becoming unusable. In general, an agent should evaluate its performance and the intentions of the requesting agent (for instance whether it appears to be a malicious or faulty agent) and refuse requests if they tend to degrade its performance beyond certain acceptable levels (which of course can be determined dynamically by an the agent).

8. FUTURE WORK

We are currently underway implementing a teamwork model that can maintain N brokers in a multi-agent system where N is more than two.

We are also working towards the pragmatics of designing an ACL (Agent Communication Language) for fault-tolerance. For instance, the ACL message structure should be designed to directly access fields such as speech-act type and the message identifier so that refusal messages can be sent without parsing and interpreting the entire incoming message. Further, we believe that certain minimal context information in the ACL message can be used to detect circular deadlocks if this information is carried over to any new message sent by (1) an agent after task decomposition, and (2) a middle agent forwarding a message.

[4] Note that this algorithm will result in a maximally connected broker graph if (1) the brokers are started one at a time, and (2) a new broker is started after a previously started broker has joined the broker team.

The recovery scheme described in this paper guarantees recovery of the system connectivity but it is left to an agent to guarantee idempotent behavior when a message that was presumably lost due to broker failure is resent by the agent shell. However, recovery of the ongoing conversations is required for a complete recovery and we are investigating an approach for recovering conversations using ACL support.

Finally, we also need to investigate the ease of implementing the above techniques using generic agents.

9. CONCLUSION

We investigated the possibility of using concepts from multi-agent systems literature for designing robust multi-agent systems and showed that teamwork and autonomy can be used to achieve this end. We introduced the Adaptive Agent Architecture (*AAA*) and discussed the design and performance of its fault-tolerance implementation. Multi-agent systems that use AAA can recover from broker failures arising out of machine, network, or process failures. It was statistically shown that the AAA implementation of teamwork did not present any appreciable overhead during steady state. We also presented experimental evidence to show that agent autonomy can prevent an agent from thrashing and hence becoming unresponsive.

In summary, this paper showed that (1) teamwork can be used to create a robust brokered architecture that can recover a multi-agent system from broker failures without incurring appreciable overheads, (2) teamwork can be used to guarantee a specified number of brokers in a large multi-agent system, and (3) autonomous agents can make a multi-agent system more robust.

10. ACKNOWLEDGEMENT

We gratefully acknowledge the support of the DARPA CoABS Program (contract F30602-98-2-0098, A0 G352) for the research presented in this paper. We would also like to thank David McGee for useful feedback during this work.

11. REFERENCE

[1] P. A. Bernstein and E. Newcomer, editors. *Principles of Transaction Processing*. High Availability, chapter 7, 1997.

[2] K. P. Birman, editor. *Building Secure and Reliable Network Applications*. Part III, Reliable Distributed Computing, chapters 12-26, 1996.

[3] L. Chen and A. Avizienis. N-version programming: A fault-Tolerance Approach to Reliability of Software Operation. *In Digest of Papers of the 8th Annual International Conference on Fault-Tolerant Computing*, Toulouse, France, 1978. As referred in [12].

[4] P. R. Cohen. On Knowing What to Say: Planning Speech Acts. *Ph.D. Thesis*, Department of Computer Science, University of Toronto, 1978.

[5] P. R. Cohen, A. Cheyer, M. Wang, and S. C. Baeg. An Open Agent Architecture. *AAAI Spring Symposium*, pages 1-8. March 1994.

[6] Philip R. Cohen and Hector J. Levesque. Intention is Choice with Commitment. *Artificial Intelligence*, Volume 42, pages 213-261, 1990.

[7] P. R. Cohen and H. J. Levesque. Rational Interaction as the Basis for Communication. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors. *Intentions in Communication*, chapter 12, pages 221-256. System Development Foundation Benchmark Series, Bradford Books, MIT Press, 1990.

[8] P. R. Cohen and Hector J. Levesque. Confirmations and Joint Action. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Morgan Kaufman Publishers, Inc., San Mateo, California, August 1991, pages 951-957.

[9] P. R. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow. QuickSet: Multimodal Interaction for Distributed Applications, *Proceedings of the Fifth International Multimedia Conference (Multimedia '97)*, ACM Press, pages 31-40.

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

[11] S. Franklin and A. Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. *Proceedings of the Third International Workshop on Agent Theories Architectures and Languages*. Springer-Verlag, 1996.

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

[13] C. Heckman and A. Roetter. Designing Government Agents for Constitutional Compliance. *Autonomous Agents '99*, Seattle, 1999.

[14] JATLite. http://java.stanford.edu.

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

[16] G. A. Kaminka and M. Tambe. What is Wrong With Us? Improving Robustness Through Social Diagnosis. *In Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, 1998.

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

[18] H. J. Levesque, P. R. Cohen, and J. Nunes. On Acting Together. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-90)*, Morgan Kaufmann Publishers, Inc., San Mateo, California.

[19] D. Lomet and G. Weikum. Efficient Transparent Application Recovery. In Client-Server Information Systems. *SIGMOD 98*, Seattle, WA, USA, 1998.

[20] K. Sycara, K. Decker, and M. Williamson. Middle-Agents for the Internet. *Proceedings of IJCAI-97*, 1997.

[21] M. Nodine, B. Perry, and A. Unruh. Experience with the InfoSleuth Agent Architecture. *Proceedings of AAAI-98 Workshop on Software Tools for Developing Agents*, 1998.

[22] Reliable Multicast Protocol. http://research.ivv.nasa.gov/RMP