THWARTING FAILURE IN MOBILE AGENT SYSTEM USING ANTECEDENCE GRAPH APPROACH

RAJWINDER SINGH AND MAYANK DAVE
Department of Computer Engineering,
National Institute of Technology, Kurukshetra-136119, India
rwsingh@yahoo.com

ABSTRACT: Mobile Agent System (MAS) is an open multi agent system, in which mobile agents can transport its state from one environment to another, with its data intact and capable of performing in new environment. This technology gave biggest opportunity to internet based applications. But fault detection and recovery is a vital issue in the deployment of MAS, whose purpose is to provide a distributed computing infrastructure for supporting applications in which components can move freely in heterogeneous environment without any hindrance. Despite the considerable efforts spent on researching and developing Multi-Agent Systems there is a noticeable absence of deployed systems. In the past the MAS research community ignored this problem – arguing that it is not a genuine MAS problem and consequently of lesser importance than other unsolved issues like cooperation, coordination, negotiation and communication. This paper presents a Watchdog approach to discover communication failure during MAS execution and recover it through parallel check pointing antecedence graph approach with low overhead. We believe that introducing fault identification and recovery in a mobile agent system through antecedence graphs is novel and provides a low overhead and effective solution for fault-tolerance in a multi-agent system.

KEYWORDS: Mobile Agent System, Antecedence Graph, Fault Tolerance.

INTRODUCTION

MAS are decentralized self-organizing systems consisting of autonomous entities called mobile agents that are an independent software program that runs on behalf of a network user. It can be characterized as having more or less intelligence and it has the ability to learn. Mobile agents (MA) add to regular agents the capability of traveling to multiple locations in the network, by saving their state and restoring it in the new host [2]. During the execution of mobile agents, communication and host failures lead to partial or complete loss of mobile agents or blocking of execution. Therefore, fault tolerant protocols are fundamental in the development of mobile agent systems in order to guarantee reliable execution of mobile agents [3].

The following undesirable scenarios may occur when MAs are sent from one host to another.

- **Communication Failure**: When mobile agent travels from one host to another, it never reaches its destination, if the destination host has failed or there is a communication link failure, implying that all routes between two hosts are disabled. Therefore, if mobile agent at one agent host (AH) wants to travel to another, it will stop advancing to it and wait until the communication link is enabled again. This is a communication failure.
- **Agent Failure**: When mobile agent travels from one host to another, it never reaches its destination due to crashes or because it is terminated by some malicious host or agent. This is agent failure.

- **Agent Host Failure**: The host platform, on which mobile agent resides, crashes or shuts down unexpectedly, due to failures. Many mobile agents on the host may be in an inactive but waiting state, due to the unavailability of external events. If more mobile agents migrate to this host, it may run out of memory. This is AH failure.

In this paper we propose a monitoring and recovery approach, comprises of two main schemes: Watchdog Agent (WA) schemes for monitoring and parallel check-pointing scheme for recovery. In Watchdog monitoring scheme, the master agent overhears the internal and external parameters of MAS. During recovery, in case of failure, using check-pointed information, the antecedence graphs and message logs are regenerated for recovery and then normal operation will continue.
The rest of the paper is organized as follows. Section 2 introduces the work related to the main topic of this paper. Section 3 describes the assumption related to proposed approach. Section 4 describes the proposed work, Section 5 gives details on experimental and performance evaluation details and section 6 proves the correctness of proposed work. Finally, Section 7 presents the conclusion and future work.

RELATED WORK

The Advanced Automation System (AAS) [16] is a distributed real-time system integrating all the services of the US air traffic control network. The requirement on high availability for this system causes to introduce both hardware and software redundancy. Here the critical services are replicated using either the active or passive approach according to the application semantics and hardware configuration. It also employs a reliable broadcast protocol maintaining the causal order of messages exchanged between the group members and a combination of rollback-recovery protocols.

Bondavalli et al. [8] proposed a framework for adaptive fault tolerance in real-time context. Their work explicitly addresses the real time constraints and employs a flexible and adaptable control strategy for managing the redundancy within the application software modules. Here the programmers can specify fault tolerance strategies for the application modules, including adaptive strategies taking into account available resources, task importance, deadlines and observed faults.
Hiltunen and Schlichting[9] introduced a general model for adaptive systems and presented some examples of how this model can be applied for different scenarios. AFTM [12] is an adaptive fault tolerant middleware based on CORBA-compliant object request broker. In AFTM the most suitable fault tolerant and resource allocation scheme is dynamically selected through user requests. It allows automatic reconfiguration of the groups and transparent masking of the faults.
Hägg [10] presented an approach for fault tolerance in which sentinel agents monitor inter-agent communication, build models of other agents and take corrective actions. Because the sentinels analyze the entire communication occurred in the system to detect state inconsistencies, their overhead could be high and they can be additional points of failures themselves.

Decker et al. [11] proposed different levels of adaptation, and concentrate only on execution adaptation where agent cloning is used for load balancing.

Kumar et al. [12] described the agent-based brokers' fault tolerant architecture, and the adaptation mechanism based on the feedback control real-time scheduling was proposed by Stankovic et al. [8].

Guessoum et al. [17] presented an adaptive multi-agent architecture with both agent level and organization level adaptation. The organization's adaptation is based on the monitoring of the system's behavior. Here the software components can be either replicated or un-replicated, and it is possible to change the replication strategy at run time.

Fedoruk and Deters [18] implemented transparent replication via proxies which handle the communication between the replicas and other agents in the MAS. Also, the proxy controls the execution of the replica group. They chose FIPA-OS agent toolkit as a platform for their implementation. As FIPA-OS does not support any replication mechanism, the replication server was implemented as a standard FIPAOS agent. Kraus et al.[13] described the problem of fault tolerance as a deployment problem, and proposed a probabilistic approach deploying the agents in a multi-agent application.

Dong Yeol Lee et al. [2] propose a monitoring and replication scheme answering the questions above. It makes a decision on the replication in the point of data. The monitoring scheme is used to acquire detailed and accurate information related to the status of the agents such as the amount of load, task length, and role. One agent called agent leader gathers the information on the status of the agents in the system, merges the information, and then sends them to the platform. The collected information is used by the replication decision scheme residing in the agent container of the platform to make a decision on the agents required to be replicated or replaced (also the agent replacing it).

ASSUMPTIONS

- Before Execution of mobile agents, MAS is tested and all the agents are working properly.
- Each host has its antecedence graph which is maintained to check whether all agents are inter-connected via communication links.
- A watchdog or the master agent that will monitor internal and external parameters of each and every mobile agent on agent hosts.

PROPOSED APPROACH FOR MONITORING AND RECOVERY IN MOBILE AGENT SYSTEM

In this section, we present a different approach that facilitates mobile agent system to detect and mitigate failure occurs at run time. For monitoring or detecting failure, we use watchdog approach. The main goal of using watchdog approach is to make system fault tolerant at execution time. Watchdog agent is an active agent in a collaborating group of n numbers of mobile agents performing similar types of operation. The watchdog agent does this by listening promiscuously to the link and the expected behavior of mobile agents. It keeps the record of internal and external states in the form of tables and checkpointing information in the form of antecedence graph. For recovery, we use parallel checkpointing antecedence graph approach, the antecedence graph information is accessible to the mobile agents and watchdog agents which requires information for checkpoint.

Watchdog Approach for Monitoring

During the execution, MAS is vulnerable with the faults such as software bugs, system crashes, shortage of resources or failures in the communication links because it operates in a distributed system environment while providing the users with various services [20]. Also, a fault in mobile agent can propagate through the system and cause the overall system to fail [19]. In order to prevent the MAS from stopping the operation because of the occurrence of any fault, we proposed a fault tolerant approach. In proposed approach, a watchdog agent is an active agent in a collaborating group of n numbers of mobile agents that keeps the record of internal states such as CPU usage, memory load and external states such as communication load, exact location of mobile agent and the number of requested messages. Now, based on the type of fault and position of mobile agent at the instant of fault occurrence, we proposed a methodology explained below.

Communication Failure. When a mobile agent travels from one host to another, it never reaches its destination, if the communication link fail, implying that the route between two hosts are disabled. Due to link failure, agents are unable to deliver the status information. Watchdog agent helps them to recover depending on the position of mobile agent at the time of failure.

Case 1: Mobile agent has just started travelling on its way to destination and link fails somewhere in between source and destination host.
Case 2: Mobile agent is nearly or exactly at the position of link failure.
Case 3: Mobile agent has arrived to its destination host and links fails afterward.

In case 1 listed above, watchdog agent will roll back the process to the recent checkpoint and sends the updated antecedence graph to source agent host for alternate path. In case 2, mobile agent may lost due to link failure and watchdog agent may or may not able to find the lost mobile agent. Hence, it automatically initiates recovery process explained in section 4.2 and intimate the same to agent host and agent server about recovery process via communication links as shown in Fig.1. In case 3, watchdog agents sends updated antecedence graph to all hosts to choose an alternate path for sending and receiving status information.

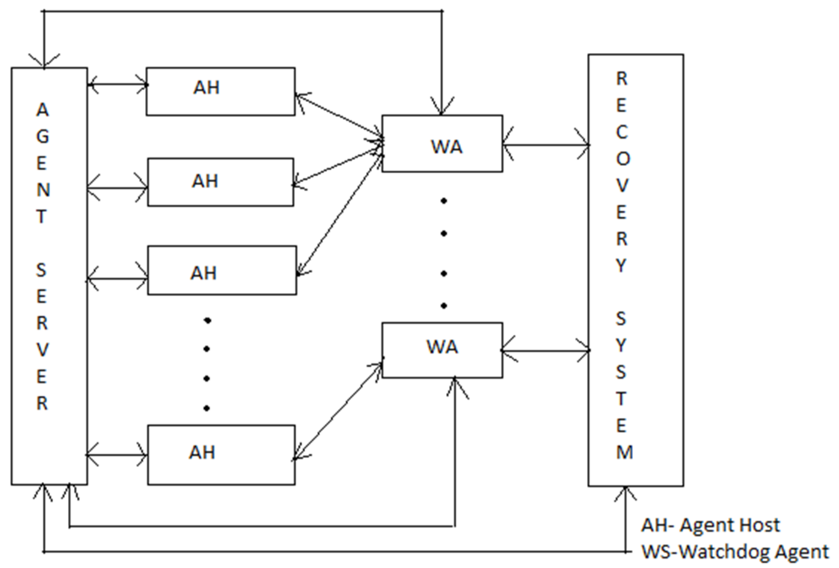

Fig.1 Proposed Fault Tolerant approach

Mobile Agent Failure. Mobile Agent Failure occurs, when destination hosts fails or terminated by malicious agent or host. Watchdog agent match the mobile agent's outputs by expected output, if any unexpected behavior will be noticed, it will immediately start recovery process explained in section 4.2 and also inform to agent host and agent server for replacement via communication links as shown in Fig.1.

Agent Host Failure. Agent Host Failure occurs, when the host platform on which a mobile agent resides, crashes or shuts down unexpectedly. Many mobile agents on the host may be inactive or in waiting state due to unavailability of external events. A watchdog agent discovers the failure by examining its internal parameters and informs the same to agent server through communication links as shown in Fig.1.
The next sub-section presents the check-pointing approach for recovery using antecedence graph.

Check-pointing Approach for Recovery using Antecedence Graph
A parallel checkpointing algorithm is used to checkpoint the antecedence graphs of individual mobile agents in the multi agent group. Parallel signifies that all the collaborating agents can take checkpoint synchronously and the checkpointed antecedence graphs are then stored into stable storage which is ultimately used for recovery and fault tolerance. The main goal of using parallel check-pointing approach for recovery is to reduce message overhead, execution, and recovery times.

Checkpointing Algorithm. In this check-pointing algorithm, each checkpoint is associated with a unique sequence number. The sequence number of each mobile agent (MA) increases monotonically and the jth checkpoint of MA_i is denoted as $C_{i,j}$. The sending and receiving events of message (Ω) are denoted as send (Ω) and receive (Ω), respectively.

The dependent agents (DA) are the active agents of the collaborating group of n number of mobile agents performing the operation. These dependent agents stores each mobile agent in form of hosts in antecedence graphs. The stored information is accessible to the watchdog agents and mobile agents which requires for the checkpoint from its antecedence graph. Whenever a fault occurs or depth of antecedence graph exceeds certain threshold or after elapsing of certain time, mobile agent (MA) and watchdog agent (WA) may request for check-pointing.

For requesting mobile agent MA_j, ($1 \leq j \leq n$), we set a variable Graph Depth (GD_j), which is the depth of requesting agent's antecedence graph at initialization of check-pointing. At threshold event, if MA_j starts a checkpoint request and informs all DA of its antecedence graph. It carries out this request through a MA called check Agent (CA) which is made for every DA during the start of checkpoint agent and the time of sending check-pointing request to the DAs. When MA_j sends this request, it attaches with DA, a numeric weight of value (1/ mod (GD_j)). In parallel the requesting agent as well as DAs make temporary AGs of the events occurred during execution of checkpointing operation. The time of this temporary logging is overlapped with actual execution of the transaction and check-pointing and so it does not have any extra load for system and is therefore non-blocking. Now all the dependent agents specified in the antecedence graph would receive the inquiry message through CA and if they agree on check-pointing, they would send back the numeric weight indicating positive response, to the starting agent. The received responses from dependent agents are added together and if they equal 1, it means that all the relevant agents have responded. In this moment, the request for changing the temporary checkpoint to the main one is issued. But even if one of them responds back negatively, the checkpointing is canceled and all DAs are informed. The distinctiveness of our scheme is that the checkpoint request is distributed through all the agents in a parallel manner.

Finally, if the starting agent received the positive response from all the dependent agents, it makes the real checkpoint and informs the others, respectively. The agent host (AH) is then sent the final checkpointed antecedence graphs by starting as well as by dependent agents. At AH the maximum length graph from these individual agents us constructed and stored in stable storage. After final checkpointing, the previous antecedence graphs are deleted which considerably reduces the size of the graph piggybacked on the message thereby helping to maintain the efficiency of algorithm in scenario where large number of agents participate in performing a transaction. After successful completion of checkpointing, the involved agents for construction of new antecedence graphs may continue from the temporarily saved antecedence graphs. In case of failure, the recovering agents request the AH to send the maximum length antecedence graph and not individual agents as in [15]. The recovering agent reconstructs its own graph from the received last maximum length antecedence graph checkpointed and stored at AH. Once the AGs of agents have been checkpointed, the agents now don't have to piggyback the checkpointed AG, thus the message size is considerably reduced. This in turn would reduce bandwidth consumption and cause speedy executions.

Recovering Algorithm. In case of failure, the checkpointed state is used for recovery. The checkpointed state here is the maximum length AG stored in the stable storage of AH. The recovering agent follows the following steps for its recovery:

1. Request for maximum length AG from AH which has been the latest saved checkpointed AG. In the existing approach in [15], the recovering agent has to request all the agents in multiagent system for their individual AGs. But contrary to that, here the recovering agent requests only AH for its AG. This in turn greatly reduces the recovery time and also eliminates the need to wait for all MAs to reply with their respective AGs. Now having received the maximum length AG from AH, the recovering agent constructs its own AG and uses it for recovery

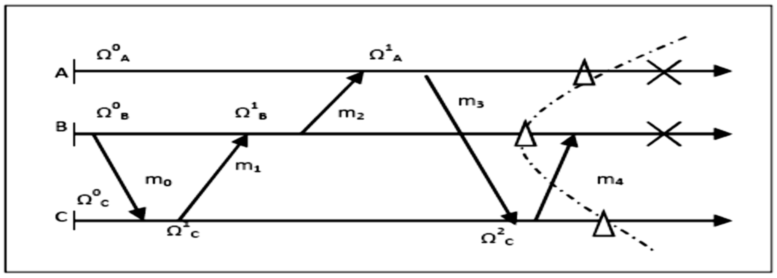

Fig.2 Multiagent system with three agents

2. The recovering agents will now create a message log using the AG constructed through above step. This message log will contain the necessary messages that need to be replayed to recover the state of each failed agent. For example, in case the agents in Fig.2, the generated message logs are shown below taking the cases for agent A and agent B as recovering and agent C as non-recovering.

Message Log for agent A (Recovering)
    receive m2 - send m3
Message Log for agent B (Recovering)
    send m0 - receive m1 - send m2
Message Log for agent C (Non-Recovering)
    receive m0 - send m1 - receive m3 - send m4

3. Using the AG and message log, messages required for recovery are replayed. This results in achievement of global consistent state.

It is important to note that messages that are to be sent in the message log are not sent. Instead, they are just present in the message log and follow “pull-based” strategy, i.e., a message is only sent if requested. After recovery the normal operation continues.

EXPERIMENT AND PERFORMANCE EVALUATION

Experimental Detail

The proposed system of multiple agents performing in collaboration in a group (i.e., MAGs) has been implemented on IBM Aglets over a network of systems with configuration of 1 GB RAM and 3.2 GHz processor connected to 10/100 Mbps Ethernet.

To evaluate the effectiveness of the proposed approach, we developed an agent system that will provide health care service provider in MAS. It consists of multiple agents for query. In order to search the suitable hospital for patient, the system keeps database (name, department, the symptoms of disease, and availability of emergency room) of all the hospitals in the city. To find a suitable hospital, the agents query the agent server. The agent server has only one plan of matching the request message to the hospital database instances, and returning the matched hospital information.

Performance Evaluation

The above setup is run for gaining insight into the performance against the earlier discussed parameters for the existing replication decision mechanism for service-oriented multi-agent system and proposed watchdog approach for fault tolerance. To compare the effectiveness of proposed scheme we have calculated the recovery time as shown in Fig.3. The comparison of recovery times is calculated with varying number of faults. Fig.3 shows the recovery times of the proposed approach as the number of failed agent increases and those of the existing scheme where the agents are replicated on the basis of condition value. As depicted in Fig.3, the recovery time of the agents with the proposed mechanism is much smaller than the existing scheme, and the difference gets bigger as the number of failed agents increases. This is because, different from the existing scheme, the proposed scheme use parallel antecedence graph approach for recovery.

We next evaluate the processing overhead of the proposed approach. The CPU time (in milliseconds) is measured assuming reliable environment of no occurrence of faults. Fig.4, shows the comparison of CPU times assuming no faults. Fig.4 shows the overhead of proposed approach with the overhead of the existing replication scheme by comparing its CPU time spend for the entire operation. Note that the proposed approach and existing scheme does not need to spend any CPU time for recovery because no fault occurs. The difference between the two schemes is because of different algorithm they are following to identify the faulty agents. In proposed

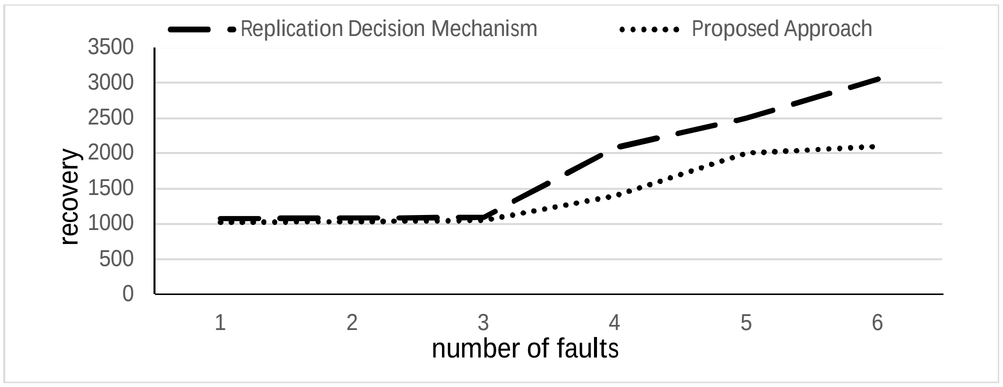

Fig.3 Recovering time with varying number of Faults

approach it is done by introducing watchdog agent doing this matching the internal and external parameters whereas in existing scheme condition value is calculated based on these parameters.

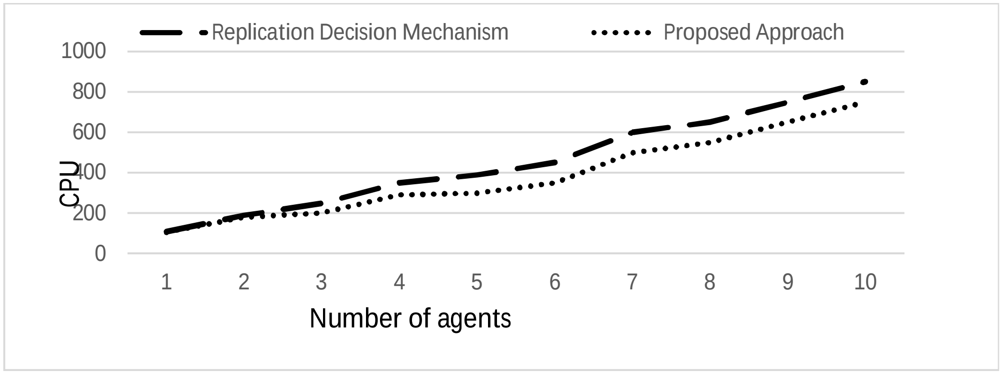

Fig.4 CPU time in no fault condition

From Fig.5, we can illustrate that the total application message overheads for almost identical for first few sets of mobile agents and the overhead of existing approach is higher than that of proposed approach. The difference between these schemes is because of checkpointing antecedence graph approach for recovery does this by AG piggybacked to messages gets checkpointed at the occurrence of threshold event, thus reducing the overhead of messages exchanged.

Lastly, we evaluate the synchronization message overhead of existing and proposed approach. Fig.6 illustrates the synchronization message overheads for existing and proposed approach. From Fig.6, we can see that the performance metric for both the approaches, all increases as the number of MAs increases. This is because Cost of sending messages between two hosts is much larger than cost of sending a message between agents of same group and cost incurred to locate a MA and forward a message to or from its group.

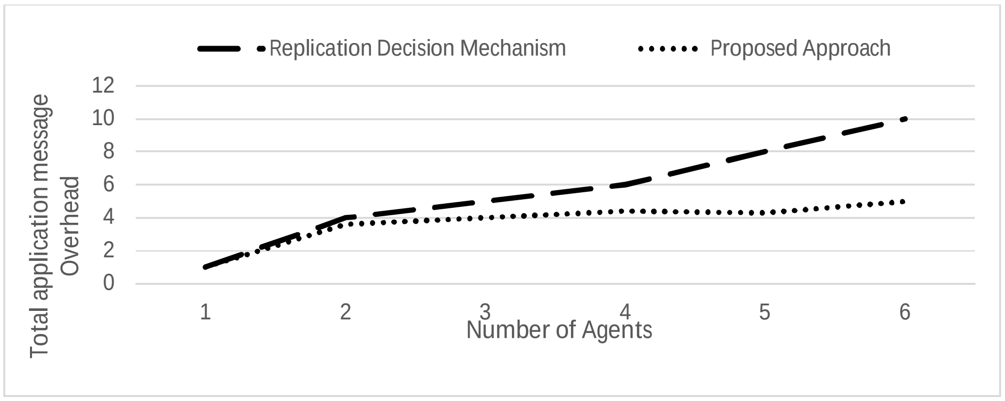

Fig.5 Total Application message overhead Comparision

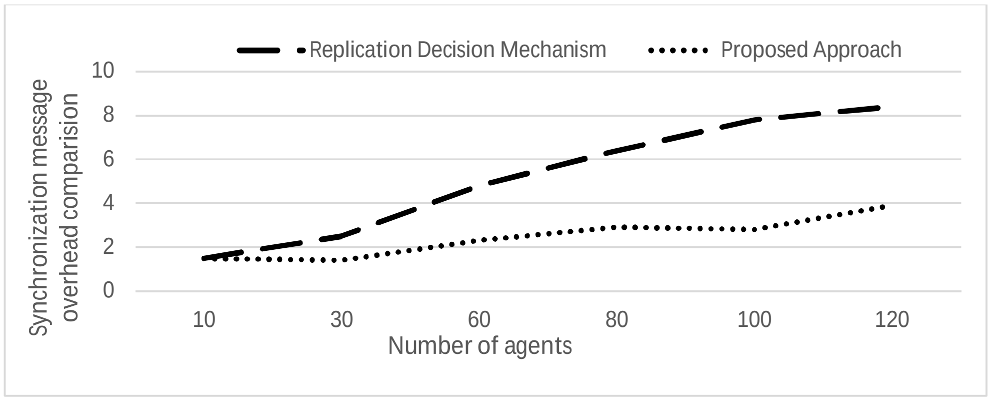

Fig.6 Synchronization message overhead Comparion

PROOF OF CORRECTNESS

Theorem 1. *A MA takes a checkpoint if and only if the MA is checkpoint dependent on the* initiator. In the proposed algorithm, only if MA receives a checkpointing request (being identified) it takes a checkpoint. So we only need to show that the identifying procedure informs MAs if and only if they are checkpoint dependent on the initiator. Because if an BA finds a wanted MA does not exist in its cell, it is responsible for forwarding the checkpointing request message to proper MA, which does not affect the identifying procedure. Therefore, for ease to present, we assume there are no handoffs in the system. If a MA, say MA_j is checkpoint dependent on the initiator, say MA_{init}, according to the message exchange procedure, there must exist a DA Є AG of MA_{init}, so the identifying procedure is able to inform DAs depending on the entry in antecedence graph. Once a DA is identified, it is sent CA and then the closure containing set of identified WAs cannot be changed due to its message exchanges. Before the identifying procedure ends, only one way can enlarge the closure, that is one of those DAs which belong to the closure set but have not been identified receives a message from another MA that does not belong to the closure set presently.

Since new message exchanges add corresponding items into antecedence graph of MA, identifying procedure can still identify such DAs. Similarly, if a MA, say MA_j, is not checkpoint dependent on the initiator, there must exist none of such entry in its antecedence graph as above according to the message exchange procedure. So it is impossible for MA_j being informed by the identifying procedure if it is not DA.

Theorem 2. *The algorithm creates a consistent global checkpoint.*
The proposition can be proved by contradiction. Suppose there is a pair of mobile agents, say MA_i and MA_j, such that at least one message s has been sent from MA_i after its last checkpoint $C_{i;q}$ and has been received by MA_j before its last checkpoint $C_{j;p}$ (note that it is an inconsistent state). We also assume $C_{j;p}$ is associated with the initiator MA_{init} checkpoint $C_{init;k}$. Based on Theorem 1, in the kth checkpointing procedure initiated by MA_{init}, MA_i must take a new checkpoint because it is checkpoint dependent on MA_{init} as it is one of the DA $\in$ AG. So the sending event of is recorded at $C_{i;q}$ which is a contradiction.

CONCLUSION AND FUTURE WORK

This paper has presented the monitoring and recovery mechanism for the fault tolerant service-oriented multi-agent system. The watchdog approach monitors the internal state of the mobile agents such as CPU usage, memory load, and the external state such as communication load, the exact location of agent and the number of requested messages of the mobile agents. The recovery mechanism takes place only for the faulty agents identified by watchdog agent as well as number of rollback messages are reduced through parallel checkpointing approach. Hence, the fall in recovery time and overhead in fault tolerance has been observed. The effectiveness of the proposed approach has been verified using the agent based health care service provider system. The experiment displays that the proposed approach significantly outperforms the existing approach in terms of recovery latency.

The future work will focus on further enhancing the performance of the proposed scheme by developing the model properly that will calculate a static value based on internal and external state of a mobile agent.

REFERENCES

Yang Xiaofan and Tang Yuan Yan, "Efficient Fault Identification of Diagnosable Systems under the Comparison Model", IEEE Transactions on Computers, Vol. 56, No. 12, December 2007.

Xu Peng, Deters Ralph, "MAS & Fault-Management", Proceedings of the 2004 International Symposium on Applications and the Internet (SAINT'04), 2004.

Choi SungJin, Baik MaengSoon, Kim HongSoo, Yoon JunWeon, Shon∗ JinGon, Hwang ChongSun," Region-based Stage Construction Protocol for Fault tolerant Execution of Mobile Agent", Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA'04), 2004.

Mitrovic Dejan, Budimac Zoran, Ivanovic Mirjana and Vidakovic Milan " Improving Fault Tolerance of Distributed Multi-Agent System with Mobile Network Management System" in Proceedings of the International Multi conference on Computer Science and Information Technology. ISSN 1896-7094. pp. 217-222.

D.E.Knuth. "The Art of Computer Programming", *Vol. 2.*Auerbach Publications, 1998.

N.A Lynch., M.Merritt., A. F. W.Weihl, and R. R.Yager, “Atomic Transactions”, Morgan Kaufmann, 1994.

Dong Yeol Lee, Seung Yeop Shin, and Hee Yong Youn,” Replication Decision Mechanism for Service-Oriented Multi-Agent System”, IEEE DOI 10.1109/SERVICES-I.2009.

Andrea Bondavalli, J Stankovic, and Lorenzo Strigini,”Adaptable fault tolerance for real-time systems”. In D. Fussell and M. Malek, editors, Responsive Computer Systems: Steps toward Fault- Tolerant Real-Time Systems, pages 187-208. Kluwer Academic Publishers, Boston, 1995.

Hiltunen, Matti. A., Schlichting, and Richard. D. “Adaptive distributed and fault-tolerant systems”, Computer Systems Science and Engineering, 11(5):275--285. CRL, Sept. 1996.

Staffan Hägg, “A sentinel approach to fault handling inmulti-agent systems”, In Proc. of the Second Australian Workshop on Distributed AI, Cairns, Australia, Aug. 27, 1996.

Keith Decker, Katia Sycara, and Mike Williamson, “Intelligent adaptive information agents”, Journal of Intelligent Information Systems, vol. 9, 1997, pp. 239 - 260.

Sanjeev Kumar, Philip R. Cohen, and Hector Joseph Levesque, “The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams”, ICMAS 2000, Boston, MA, July 2000.

Sarit Kraus, V.S. Subrahmanian, N. Cihan, “Probabilistically survivable MASs”, In Proc. of Eighteenth International JointConference on Artificial Intelligence (IJCAI-03), pp. 789-795, 2003.

Rajwinder Singh and Mayank Dave, “Antecedence Graph Approach to checkpointing for Fault Tolerance in Mobile Agent System ”IEEE Transactions on Computers, Vol. 62, No. 2, February2013.

M.M. Khokhar, A. Nadeem, and O.M. Paracha, “An Antecedence Graph Approach for Fault Tolerance in a Multi-Agent System,”Proc. IEEE Seventh Int’l Conf. Mobile Data Management, 2006.

Flaviu Cristian and San Jose, “Fault-Tolerance in the Advanced Automation System” In 20th International Conference on Fault-Tolerant Computing, Newcastle upon Tyne, England (1990).

Zahia Guessoum, Jean-pierre Briot, Olivier Marin, Athmane Hamel, and Pierre Sens, “Dynamic and adaptive replication for large-scale reliable multi-agent systems”, In Software Engineering for Large-Scale Multi-Agent Systems (SELMAS), LNCS 2603, pp.182-198, April 2003.

Alan Fedoruk and Ralph Deters, “Improving fault-tolerance by replicating agents”, In Proc. AAMAS-02, Bologna,Italy, pp.737-744, 2002.

Salvatore F. Pileggi, Manuel Esteve “An adaptive and flexible fault tolerance mechanism designed on multi-behavior agents for Wireless Sensor/Actuator Network” IARIA’07, pp. 283 – 288, Aug. 2007.

Hyun Ko, Hee Yong Youn, “A new agent characterization model and grouping method for multi-agent system”, IRI’08, pp.86-91, Jul.2008.