New Fault Tolerance Approach using Antecedence Graphs in Multi Agent Systems

Ramandeep Kaur1, Rama Krishna Challa1, Rajwinder Singh2
1 Department of Computer Science, National Institute of Technical Teachers’ Training and Research, Chandigarh, India.
2 Department of Computer science & Engineering, CEC, Landran, (Mohali), Punjab, India.

Abstract: Mobile agents are distributed programs which can move autonomously in a network, to perform tasks on behalf of user. They are susceptible to failures due to faults in communication channels, processors or malicious programs. In order to gain solid foundation at the heart of today's e-society, the mobile agent technology must address the issue of fault tolerance. Checkpointing has been widely used technique for providing fault tolerance in mobile agent systems. But the traditional message passing based checkpointing and rollback algorithms suffer from problems of excess bandwidth consumption and large overheads. This paper proposes use of antecedence graphs and message logs for maintaining fault tolerance information of agents. For checkpointing, dependent agents are marked out using antecedence graphs; and only these agents are involved in process of taking checkpoints. In case of failures, the antecedence graphs and message logs are regenerated for recovery and then normal operation continued. The proposed scheme reports less overheads, speedy execution and reduced recovery times as compared to existing graph based schemes.

Keywords: Mobile agents, fault tolerance, antecedence graphs, checkpointing, message logs.

I. INTRODUCTION

Mobile agents are becoming a major trend in distributed systems and applications. A mobile agent is a program that represents a user in a computer network and can migrate autonomously from node to node, to perform some computation on behalf of the user [1]. Its tasks, which are determined by the agent application, can range from online shopping to real-time device control to distributed scientific computing. It can bring benefits such as reduced network load and overcoming of network latency. Applications can inject mobile agents into a network, allowing them to roam in the network, either on a predetermined path or one that the agents themselves determine based on dynamically gathered information. Having accomplished their goals, the agents can return to their home site to report their results to the user [2]. Most of these applications require high degree of reliability and consistency. Therefore, fault tolerance is a key issue in designing mobile agent systems [5, 11]. In this paper we consider the scenario of multi-agent system consisting of several collaborating agents and amalgamate the concept of checkpointing and antecedence graphs for fault tolerance in multi agent systems.

The rest of the paper is organized as follows: section 1.1 briefs the related research in the area of fault tolerance for mobile agent systems. Section 2 describes the basic framework of the proposed scheme and illustrates the procedure and algorithm of proposed scheme of checkpointing and recovery. The performance analysis and results of comparison with existing schemes is given in section 3 followed by conclusion about effectiveness of proposed scheme in section 4.

A. RELATED WORK

As mobile agent systems scale up, their failure rate may also be higher. Several techniques have been proposed for providing fault tolerance in mobile-agent systems [3] which broadly fall under two basic categories i.e. replication and checkpointing. Checkpointing is one of the widely used fault tolerance techniques and can be classified into synchronous, asynchronous and quasi-synchronous algorithms [6, 10]. For recovery an agent needs to rollback to its consistent state. Message logging for rollback require that each agent periodically saves its local state and logs its every message sent and received. Message logging protocols are classified into pessimistic, optimistic and causal [9]. Replication schemes as discussed in [4, 8, 12] mainly rely on replicated servers or agents to mask the failures. Graph based fault tolerance approach for multi agents has been proposed in [15] where the fault tolerance is achieved by use of antecedence graphs combined with message logs.

Majority of checkpointing schemes approaches suffer from the overhead that result from forcing all the agents in multi-agent system to checkpoint. The blocking of agents during checkpointing increases the execution time of transaction. To overcome the problem of recovery latency and blocking, we propose coordinated checkpoint algorithm that is able to force the most limited number of agents carrying out process, for putting checkpoint. The global checkpointing is done from antecedence graph [15] where dependent agents are identified and only they are forced to put checkpoints. The concept of antecedence graphs for fault tolerance in distributed systems was originally introduced in Manetho [14] which utilized antecedence graphs and message logs for fault tolerance in distributed systems. But the overhead due to size of antecedence graph with large number of agents involved causes greater overheads in case of multi-agent systems if used without checkpointing. Our proposed scheme combines the antecedence graph approach with parallel checkpointing and message logging. The proposed scheme significantly resolves the associated problem of overhead besides improving execution and recovery time.
II. SYSTEM FRAMEWORK

The system consists of cooperating multiple agents (on a single or multiple mobile hosts) which form multi agent group and collaborate with each other to perform a single computationally complex task by passing messages between each other as shown in Fig. 1.

Each group has a Base Agent (BA) which coordinates the participating agents of group and is assumed to execute in fail safe mode. It also acts as recovery manager and maintains access to persistent data storage, where agent checkpoints and recovery bookkeeping is held. Under our strategy, each mobile agent will send its current antecedence graph to the agent that it is sending a message to. All the messages exchanged would be stored by each agent in its volatile storage in form of message logs. The mobile agents may perform checkpointing of the antecedence graph either when the depth exceeds certain threshold of specified nodes in its antecedence graph or after elapsing of specific time.

In general, most of the operations of internet applications are based on read operation, so we can safely assume that all the operations executed by the mobile agents are idempotent, thus the exactly once execution property is adhered to automatically. The three basic steps involved in the proposed scheme are formation of antecedence graph at individual agents followed by parallel checkpointing and rollback recovery in case of failure. These are discussed in detail in the following sections.

A. Antecedence Graph (AG) Formation For Dependency Information

Considering a scenario of a multi-agent system consisting of only three agents, agent A, agent B, and agent C. Its inter agent communication can be depicted in form of a graph as shown in Fig. 2. Each agent, at the start of its execution, is at state $\Omega_0^A$, $\Omega_0^B$ and $\Omega_0^C$ respectively. Each message receipt forms a deterministic interval. For example, the receipt of message $m_1$ from B to C forms the deterministic interval and the antecedence graph of state $\Omega_1^B$ provides information about what happened before $\Omega_1^B$.

![Fig. 2 An example of multi-agent system with three agents](image)

B. AG Formation for Agent A

The formation of antecedence graph for Agent A takes the following steps: Message $m_2$ is received by Agent A from Agent B. A combines the antecedence graph received from B to its own graph for the formation of the event $\Omega_1^A$. The resultant graph is illustrated in Fig. 3.

![Fig. 3 AG for agent A](image)

![Fig. 4 AG for agent B](image)

![Fig. 5 AG for agent C](image)
Similarly agent B and C construct their antecedence graphs as shown in Fig. 4 and Fig. 5.

**C. Parallel Checkpointing**

The main goal of proposed scheme is to minimize the global checkpointing latency and to reduce the total recovery time. Coordinated checkpointing is utilized for checkpointing as it shows better performance as compared to other schemes as shown by comparative studies in [6].

The dependent agents are the active agents of the collaborating group of n number of mobile agents performing the operation. These dependent agents for each mobile agent are stored in form of nodes of antecedence graphs. In proposed scheme, the dependence information is accessible to the agent which requires for the checkpoint from its antecedence graph. When the antecedence graph depth exceeds certain threshold or after elapsing of certain time, mobile agent (MA) may request for checkpointing. For requesting 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 checkpointing. At threshold event, if MA\(j\) starts a checkpoint request and informs all dependent agents (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 checkpointing request to the DAs.

When MA\(j\) sends this request, it attaches with CA, a numeric weight of value \(1/|GD_j|\). 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 checkpointing 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 checkpointing, 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 cancelled 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 BA is then sent the final checkpointed 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. Using the AG, 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. Using the AG and message logs, messages required for recovery are replayed. This results in achievement of global consistent state. After recovery, the normal operation continues.

**III. PERFORMANCE ANALYSIS AND COMPARATIVE STUDY**

The proposed system of multiple agents performing in collaboration in a group has been implemented on IBM Aglets [7] over a network of systems with configuration of 1 GB RAM and 3.2 GHz processor connected be 10/100 MBPS Ethernet. Aglets [13] is a java based graphical interface for developing the distributed multi-agent systems. The case scenario used to implement the proposed system is searching for best deals offered by suppliers in the market. An agent acts as supplier of items. In order to get the best deals, the server acting as supplier is used to retrieve this information from various agent servers acting as supplier. There may be more than one mobile agent at each server. The inter agent coordination is achieved through exchanging messages among mobile agents. The proposed system is designed to minimize the global checkpointing latency and to reduce the total recovery time. It is coordinated for checkpointing as it shows better performance as compared to other schemes. The dependent agents are the active agents of the collaborating group of mobile agents performing the operation. These dependent agents for each mobile agent stored in form of nodes of antecedence graphs. In proposed scheme, the dependence information is accessible to the agent which requires for the checkpoint from its antecedence graph. When the antecedence graph depth exceeds certain threshold or after elapsing of certain time, mobile agent (MA) may request for checkpointing. For requesting 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 checkpointing. At threshold event, if MA\(j\) starts a checkpoint request and informs all dependent agents (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 checkpointing request to the DAs.

When MA\(j\) sends this request, it attaches with CA, a numeric weight of value \(1/|GD_j|\). 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 checkpointing 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 checkpointing, 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 cancelled 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 BA is then sent the final checkpointed antecedence graphs by starting as well as by dependent agents. At BA the maximum length graph from these individual agents is 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. Following is the brief proposed checkpointing algorithm:

*Requesting Agent MA\(j\) identifies Dependent Agents (DA)*

For each Agent \(\epsilon AG\)

Create Check Agent (CA)

MA\(j\) sends a CA with temp-checkpoint request and value \(1/|GD_j|\) to all DAs (where \((1 \leq j \leq n)\))

\[ W=0 \]

For each agent\(\epsilon AG\)

MA\(j\) receives reply to temp-check request.

For each reply compute:

\[ W=W + 1/|GD_j| \]

if \(W\neq 1\) then

cancel checkpointing & wait for threshold event

if \(W=1\) then

At MA\(j\) and all DAs:

Save AG as checkpoint.

Send the final checkpointed AG to BA.

Discard successfully checkpointed nodes from AG.

Continue again from temporary AG.

At BA:

Construct maximum length AG from received AGs.

Write it to stable storage.

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. 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 BA. The recovering agent requests for maximum length AG from BA which has been the latest saved checkpointed AG. 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. Using the AG and message logs, messages required for recovery are replayed. This results in achievement of global consistent state. After recovery, the normal operation continues.
communication is through mobile agents using messages. The dependent agents are the active agents of the collaborating group of mobile agents performing the operation. The number of dependent agents is gradually increased to study the variations in parameters.

Fig. 6 shows the comparison of checkpointing for non-checkpointing antecedence graph approach [15] and the proposed scheme. The proposed approach reports much less checkpointing time as the only dependent agents are involved in checkpointing. Participation of only dependent agents reduces the overhead of waiting for response from all agents of the group. Reduction in checkpointing time is significant advantage of our approach.

The execution of the operation being performed by the collaborating group has been done once without checkpointing as in [15] and secondly with checkpointing using the proposed scheme. To measure the variation in execution time, five iterations were done for different number of dependent agents as shown in Fig. 7. Analysis of the results shows that the execution time for both approaches (with and without checkpointing) remains nearly same for smaller number of dependent agents. When the number of dependent agents increases, the proposed checkpointing approach, results in faster execution. This can be attributed to the fact that due to checkpointing the antecedence graph piggybacked on the messages exchanged by agents, never exceed a preset limit. On the other hand the size of the graph piggybacked in non-checkpointing approach increases with increase in number of dependent agents.

This results in increase in execution time. The integration of checkpointing with antecedence graph as in proposed approach can greatly reduce the time for normal execution of operation in multi agent group. Besides the recovery too can be faster in case of failing agents. Thus checkpointing can greatly enhance the performance of the antecedence graph approach for fault tolerance.

The execution of the operation being performed by the collaborating group has been done once without checkpointing as in [15] and secondly with checkpointing using the proposed scheme. To measure the variation in execution time, five iterations were done for different number of dependent agents as shown in Fig. 7. Analysis of the results shows that the execution time for both approaches (with and without checkpointing) remains nearly same for smaller number of dependent agents. When the number of dependent agents increases, the proposed checkpointing approach, results in faster execution. This can be attributed to the fact that due to checkpointing the antecedence graph piggybacked on the messages exchanged by agents, never exceed a preset limit. On the other hand the size of the graph piggybacked in non-checkpointing approach increases with increase in number of dependent agents.

This results in increase in execution time. The integration of checkpointing with antecedence graph as in proposed approach can greatly reduce the time for normal execution of operation in multi agent group. Besides the recovery too can be faster in case of failing agents. Thus checkpointing can greatly enhance the performance of the antecedence graph approach for fault tolerance.

The execution of the operation being performed by the collaborating group has been done once without checkpointing as in [15] and secondly with checkpointing using the proposed scheme. To measure the variation in execution time, five iterations were done for different number of dependent agents as shown in Fig. 7. Analysis of the results shows that the execution time for both approaches (with and without checkpointing) remains nearly same for smaller number of dependent agents. When the number of dependent agents increases, the proposed checkpointing approach, results in faster execution. This can be attributed to the fact that due to checkpointing the antecedence graph piggybacked on the messages exchanged by agents, never exceed a preset limit. On the other hand the size of the graph piggybacked in non-checkpointing approach increases with increase in number of dependent agents.

This results in increase in execution time. The integration of checkpointing with antecedence graph as in proposed approach can greatly reduce the time for normal execution of operation in multi agent group. Besides the recovery too can be faster in case of failing agents. Thus checkpointing can greatly enhance the performance of the antecedence graph approach for fault tolerance.

IV. CONCLUSIONS

In this paper we proposed an approach to introduce fault tolerance in multi agent system through checkpointing using antecedence graph approach. The integration of checkpointing with antecedence graph approach significantly improves the performance of collaborating group of agents. Experimental results show that checkpointing done through collection list of only dependent agents underlined by antecedence graphs results in better execution time and low checkpointing time. In future, comparison of the graph based approach with other approaches can be made on the suitability of approach for various applications. Besides, the proposed scheme can be implemented into real life applications for providing reliability.
REFERENCES


