Dynamic Fault Tolerance in Distributed Simulation System

Min Ma, Shiyao Jin, Chaoqun Ye, and Xiaojian Liu

School of Computer Science, National University of Defense Technology,
Hunan Changsha 410073, China
maminde@163.com

Abstract. Distributed simulation system is widely used for forecasting, decision-making and scientific computing. Multi-agent and Grid have been used as platform for simulation. In order to survive from software or hardware failures and guarantee successful rate during agent migrating, system must solve the fault tolerance problem. Classic fault tolerance technology like checkpoint and redundancy can be used for distributed simulation system, but is not efficient. We present a novel fault tolerance protocol which combines the causal message logging method and prime-backup technology. The proposed protocol uses iterative backup location scheme and adaptive update interval to reduce overhead and balance the cost of fault tolerance and recovery time. The protocol has characteristics of no orphan state, and do not need the survival agents to rollback. Most important is that the recovery scheme can tolerant concurrently failures, even the permanent failure of single node. Correctness of the protocol is proved and experiments show the protocol is efficient.

1 Introduction

Distributed simulation is popularly used in economy, education and society. The scale of simulation system develops from small to large rapidly, the platform of simulation system combines the technology of Grid and mobile agent, and furthermore, system executive time extends from hours to days even months. All these changes challenge fault tolerance in simulation system, and moreover the mobile agent system must guarantee the successful rate of migration. However there are few fault tolerance mechanisms in simulation system at present. When a simulation system fails, it will simplify restart. Some simulation systems like forecast and decision-making system have real-time requirement, the restarting method can not meet the real time requirement. It will be insignificant if the arrival results exceed the deadline.

Now, little is considered about the problem of fault tolerance in architecture of simulation system, like the popular HLA (High Level Architecture), though RTI (Run-Time Infrastructure) supplies with function of Save/Restore, the simulation entity can not guarantee the global state consistent during recovery by simple functions of Save/Restore. Multi-Agent system is a popular platform for complexity system simulation. Agents have characteristic of autonomy and intelligence, and moreover they have mobile abilities so they can migrate from node to node in network. Commonly the distributed discrete events simulation system is driven by events, so the order of interactive messages between simulation entities (usually

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part I, LNCS 3991, pp. 769 – 776, 2006.

agent) is important. However the fault tolerance scheme in mobile agent system concerns local computing only. The requirement of simulation system must keep global state consistent and events logic causal order, so it needs to present a suitable fault tolerance protocol for simulation systems.

The paper is organized as follows. In sections 2 and 3 we state the motivation and propose a novel fault tolerance system framework. In section 4 recovery operations are discussed and the correctness is proved. Experiment and results are showed in section 5, at last a summary concludes the paper in section 6.

2 Background and Motivation

Researchers have paid more attention to fault tolerance in simulation. Most of them focus on using classic fault tolerance technology such as redundancy and checkpoint scheme. Damani and Garg introduced a fault tolerance scheme based optimistic message logging[1]. Failed simulation entity will restore checkpoint and replay messages in log, while the optimistic message logging can not guarantee all delivered messages have been logged, so orphan state may be introduced. The system must roll back relevant simulation entities to keep the global state consistent. The recovery operation will cost system much time, and moreover, this scheme can only been used in simulation system which implements optimistic time management, however local virtual time of entity can not rolled back in conservation time management. Luthi and Berchtold presented a fault tolerance framework based on active replica in HLA[2]. Every entity has several backups, the prime and backup entities compute concurrency. The output of entities will be selected by vote. This method can solve Byzantine and failure-stop error. But multi-backup costs many system resources, and it is difficult to keep consistent between prime and backup entities.

Combining the advantages of causal logging[3] and prime-backup scheme, we present a novel fault tolerance protocol in distributed simulation system. The scheme is orphan free and does not need survived entities to rollback; furthermore neither it will block the system to coordinate entities nor introduce additional messages.

3 System Model and Framework

We consider the simulation system composed of simulation nodes and interconnect network. A simulation entity can be regarded as a logic process executed in simulation nodes. An entity is called federation in HLA and agent in multi-agent system. Entities have inner states and communicate by messages. We use a graph $G(V,E)$ to illustrate the distributed topology of simulation nodes. A simulation entity locates in a node and communicates with other entities located in other nodes, an entity and the relevant entities can be presented by graph $G'(V,E)$, like Fig.1.

3.1 Prime-Backup and Heartbeat Mechanism

We use backup entity to monitor prime entity. Backup entity sends heartbeat signals to prime entity in fixed time interval, and receives updating message from the prime

entity. T_{update} is set to control the updating frequency. After a time interval of T_{update}, backup entity updates once. When there is a failure happened, the backup will substitute for the prime entity, so the simulation system can continue executing.

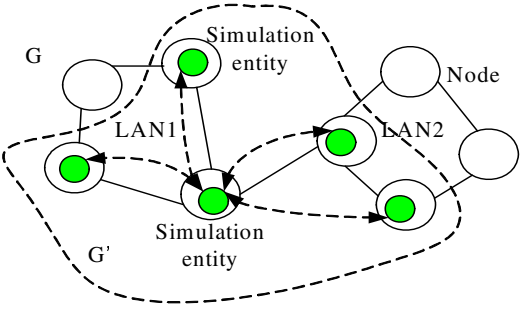


Fig. 1. Topology of simulation nodes and entities

Advantages of proposed adaptive prime-backup scheme:

1. Backup entity and prime entity are distributed in difference simulation node, so after prime entity failure, the backup will take place of prime entity quickly.
2. Though backup entities occupy some system resources, but the cost of updating is less than the cost of saving and restoring. Furthermore the checkpoint algorithm must consider the problem of checkpoint coordination and garbage recycled.
3. According output and input operation, the external environment can not be rollback. If the environment changes too much, the simulation results may be different after recovery. While using T_{update} can control the difference between system recovery before and system recovery after, so we can reduce the difference by select suitable T_{update}.

3.2 Iterative Backup Placement for Reducing Updating Cost

In order to reduce the cost of backup entity updating, we applied iterative[4] scheme to locate backup entity. Assume a simulation entity migrates between the nodes. The path is $A \Rightarrow B \Rightarrow C \Rightarrow D \Rightarrow E$ in Fig.2, when the entity arrives at the node C, the

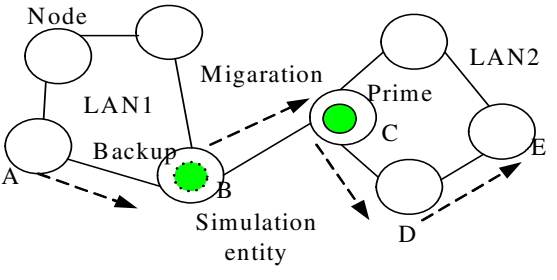


Fig. 2. Migration of simulation entity

backup entity will locate in node B, next step the prime entity migrates to node D, the backup entity will be placed in node C. That is to say the migrating path is $node_1 \Rightarrow node_2 \Rightarrow ... \Rightarrow node_n$, if prime entity is in $node_i$, then the backup entity will locate in $node_{i-1}$.

This scheme can guarantee only one step far away between the prime entity and backup, so it is more efficient than the scheme of fixed backup placement.

1. The scheme can guarantee stable of communicating between the prime entity and backup entity, because the distance of network topology between them will not be too far.
2. When the prime entity migrates to new node, it will clone itself in new node, at same time the old entity in node can be used as backup entity directly, then the old backup entity is notified to destroy itself. The whole procedure reduces backup entity migration, so it easy to implement.

4 Recovery Operation

We propose a distributed fault tolerance protocol. Every entity logs the messages received or sent within update interval T_{update} .The recovery contains several phases.

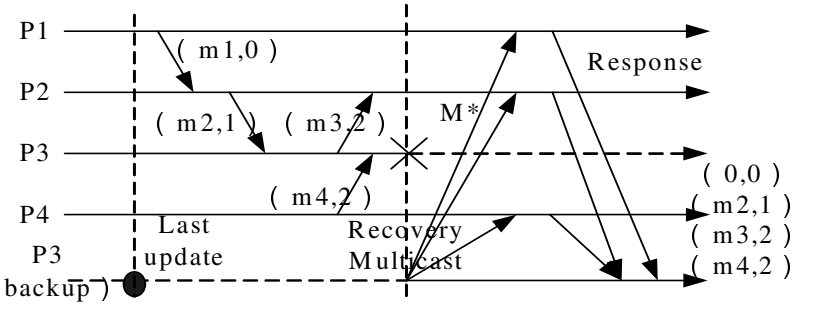


Fig. 3. Process recovery procedure operations

Firstly, the backup entity multicasts recovery announcement to relevant entities and requests them to return the concerned messages stored in history message log. Secondly, the backup entity receives the messages from relevant entities and arranges the messages by the logic time order. Thirdly, the backup entity replays the sorted message queue and restores the state up to same with the prime entity. Then the backup entity substitutes for failed prime entity in simulation system.

In fig.3, there are four logic processes p_1, p_2, p_3, p_4 and the messages m_1, m_2, m_3, m_4. The number after the message is the LVT which wrapped in message. We can see that after p_3 received m_4 sent by p_4, process p_3 failed, so the backup process of p_3 takes the recovery action. Firstly, p_3 multicasts the recovery messages m* to p_1, p_2, p_4, when the survival processes receives the recovery message, it checks the m*.lvt and wrapped process id of the sender, then it returns all the messages relevant with

process id and m.lvt>m*.lvt . p_3 will receive the message set of m_2, m_3, m_4 which return by p_1, p_2, p_4. Message m_1 will not be received by p_3, because it has no relation with p_3. Secondly p_3 will arrange the received message set by the LVT order. It will be set with order $m_2 \rightarrow m_3 \rightarrow m_4$. The LVT of m_3, m_4 are both 2, so m_3, m_4 are regard as concurrency and we can place m_3 before m_4 or m_4 before m_3. The result will be same. Thirdly, p_3 replays the message $m_2 \rightarrow m_3 \rightarrow m_4$ and restores the state before failure, and then the simulation system state is consistent and continues simulating.

4.1 Correctness of Recovery Protocol

Assume the set of all processes in simulation system is $\mathbb{N}$, the set of failure processes is $\mathbb{R}$, $\mathbb{R} \subseteq \mathbb{N}$ and there is only one process fail at anytime.

Definition 1. $Depend(m)$ is set of processes whose states depend on the message m.

$$Depend(m) \overset{def}{=} \{p_j \in \mathbb{N} \mid (j \in m.dest) \vee (\exists e_j : (deliver(m) \rightarrow e_j))$$

Definition 2. $Log(m)$ is set of process which log the m in volatile memory.

$$Log(m) \overset{def}{=} (p_j \in \mathbb{N} \mid j = m.dest \vee j = m.send)$$

Definition 3. $Orphan(p)$ represent process p is an orphan process.

$$Orphan(p) \overset{def}{=} (\mathrm{p} \in \mathbb{N}\text{-}\mathbb{R}) \wedge \exists m : (p \in Depend(m) \wedge (p_{m.send} \notin Log(m))$$

Lemma 1. If single process failed, $\forall \mathrm{m}, \exists p \in Log(m) \wedge p \in \mathbb{N} - \mathbb{R}$

Proof: Assume that message m lost:

$$\exists m, \forall p \in \mathbb{N} - \mathbb{R} : p \notin Log(m)$$

$$\because Log(m) \overset{def}{=} (p_j \in \mathbb{N} \mid j = m.dest \vee j = m.send)$$

$$\therefore p_{m.send} \in \mathbb{R} \wedge p_{m.dest} \in \mathbb{R} \wedge (p_{m.send} \neq p_{m.dest})$$

Then there two failure processes exist. It contradicts original assumption of only one process fails; the result is correct. □

Lemma 2. If there is only single process failure, recovery protocol guarantee no orphan process created,

$$\forall p \in \mathbb{N} : \neg Orphan(p) .$$

Proof: According to lemma 1

$$\forall \mathrm{m}, \exists p \in Log(m) \wedge p \in \mathbb{N} - \mathbb{R}$$

There are two cases during recovery procedure:

Case 1: $p_{m.send} \notin \mathbb{R}$, then according the definition of orphan(p)

$$p \in \mathbb{N} : \neg Orphan(p)$$

Case 2: $p_{m.send} \in \mathbb{R}$, then $p_{m.send}$ will retrieve the message m from $p_{m.dest}$ in recovery operation. Then

$$p \in \mathbb{N}: \neg Orphan(p)$$

Conclude from case 1 and case 2, the result is correct. □

Lemma 3. Failure process p arranges collected messages with LVT wrapped in messages; the new message sequence $\{m'_1, m'_2, ...m'_n\}$ has the same causal order with the history message sequence $\{m_1, m_2, ...m_n\}$.

Proof: The message sequence $\{m_1, m_2, ...m_n\}$ which process p received has happen before relationship [3]:

$$m_1 \rightarrow m_2 \rightarrow ... \rightarrow m_n$$

Logic virtual time order of Messages has relation:

$$m_1.lvt \leq m_2.lvt \leq ... \leq m_n.lvt$$

When process p failed and to be recovered, it will retrieve message set $\{m'_1, m'_2, ...m'_n\}$ from relevant processes to restore the state according to the protocol, the new message sequence has logic virtual time order

$$m'_1.lvt \leq m'_2.lvt \leq ... \leq m'_n.lvt$$

According the protocol assumption, the message with same LVT will be regarded as concurrency events, and then we can arrange messages with the same LVT an arbitrary order, then we get

$$m'_1 \rightarrow m'_2 \rightarrow ... \rightarrow m'_n$$

According to the PWD (Piecewise Deterministic) assumption [7] we can get

$$deterministic(m'_1, m'_2, ..., m'_n) = deterministic(m_1, m_2, ..., m_n)$$

So the new message sequence will get same execute result with history message sequence. □

Theorem 1. The recovery protocol can guarantee the system recover to the state before failure and all processes states are global consistent.

Proof: According lemma1, failure process can retrieve entire relevant messages; according lemma2, no survived processes become orphan; according lemma3, the recovery process can arrange messages sequence to right order, then after replaying the retrieval message queue the failure process will recover state to failure before, and all processes in system are consistent. □

4.2 Recovery Protocol Extend to *f* Concurrently Failures

Theorem 2. If updating backup entity by history message queue, the recovery protocol can tolerate *f* concurrent failures.

Proof: Assume there are *f* (*f*<*N*) processes failed concurrently in system, then backup processes take recovery action, each backup process may start recovery action from different logic virtual time, but there must exist a minimal time $T_{roll\min}$. All events and messages before the time $T_{roll\min}$ have been logged in system, then next

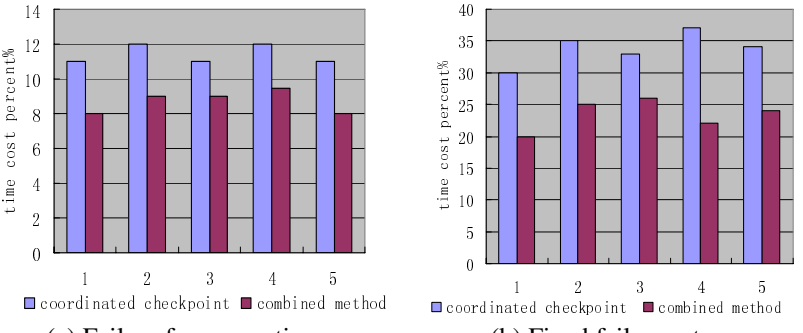
timestamp $T_{roll\min}+1$ the state of system is deterministic by state and output messages of time $T_{roll\min}$, so the system state can restore to $T_{roll\min}+1$ according to the PWD assumption, the survival processes can not push forward their local virtual time according to the synchronous mechanism of conservation time management of simulation. So failure processes can recover sequentially from $T_{roll\min}$ to $T_{roll\min}+1, T_{roll\min}+2$, … , until they recover to time $T_{\text{beforefailure}}$, then the system global virtual time can be pushed forward and continue simulations.

Corollary 1. The recovery protocol can solve single node failure.

When a simulation node failed, the simulation system can recover the failed process using backup process located in other node, so a single node failure can be solved.

5 Experiment and Results

Experiment was done to test the efficiency of proposed recovery protocol in Jcass. Jcass is a multi-agent complex simulation platform developed by national university of defense technology. The Fig.4(a) shows additional time cost in percent comparing the case of system employing the fault tolerance methods with the case of system not employing. The fig.4(b) shows the additional time cost comparing executing time in fixed events and migrating failure rate with executing time in failure free. Hardware platform is four PC with Celeron 800MHz, 256M SDRAM, 10M LAN. Test was done with 200 agents and 200 times migrations. Total 1000 events executed, and events failure rate was 10% migrating failure rate was 10%. System executed about 1 hour in failure free and employing proposed method.



(a) Failure free execution (b) Fixed failure rate

Fig. 4. Results of Experiment

Experiment results show that combined method has extra time cost 7% and method of checkpoint has extra time cost 11% in failure free execution. If we used fixed failure rate in execution, extra cost of combined method is about average 16%, while extra cost of checkpoint is about average 34%.

6 Conclusion

According to analysis of simulation framework and distributed recovery mechanism, we present a novel recovery scheme for simulation system. The scheme uses backup entity to monitor the prime entity and stores entity states, a recovery protocol is designed to solve the problem of system global state consistent. Correctness of recovery protocol is proved in theory. At last experiment was done to test the efficiency of recovery protocol; the experiment results show that proposed method reduces nearly half of time cost compared with checkpoint method whenever in failure free or fixed failure rate.

References

[1] Damani., "Fault -tolerant distributed simulation," presented at proceedings of the 12th workshop on parallel and distributed simulation(PADS'98), 1998.
[2] S. G. Johnnes Luthi, "F-RSS: A Flexible Framework for Fault Tolerant HLA Federations," presented at ICCS 2004.
[3] E. N. Elnozahy, D. B. Johnson, and Y. M. Wang. "A survey of rollback-recovery protocols in message-passing systems". Technical Report CMU-CS-96-181, Carnegie Mellon University, October 1996.
[4] D.Johansen, K.Marzullo, F.B.Schneider, K.Jacobsen, D.Zagorodnov."NAP: Practical Fault-Tolerance for Itinerant Computations". Technical Report TR98-1716. Department of Computer Science, Cornell University. USA. November, 1998
[5] L. Leslie, "Time, clocks, and the ordering of events in a distributed system," Commun. ACM, vol. 21, pp. 558-565, 1978.
[6] D. Agrawal, Agre,J.R., "Replicated objects in time warp simulations," presented at Proc. 1992 Winter Simulation Conference,SCS(1992), 1992.
[7] S. Rob and Y. Shaula, "Optimistic recovery in distributed systems," ACM Trans. Comput. *Syst.*, vol. 3, pp. 204-226, 1985.
[8] Michael R. Lyu, Xinyu Chen, Tsz Yeung Wong, "Design and Evaluation of a Fault-Tolerant Mobile-Agent System," IEEE Intelligent Systems, vol. 19, no. 5, pp. 32-38, Sept/Oct, 2004.
[9] F. Alan and D. Ralph, "Using Dynamic Proxy Agent Replicate Groups to Improve Fault-Tolerance in Multi-Agent Systems", AAMAS'03, July 14-18, 2003