Fault Tolerance for Internet Agent Systems: in cases of stop failure and Byzantine failure

Tadashi Araragi
NIPPON TELEGRAPH AND TELEPHONE CORPORATION
NTT Communication Science Laboratories
2-4, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0237 Japan
araragi@cslab.kecl.ntt.co.jp

ABSTRACT

In this demo, we present our two fault-tolerant systems to overcome stop failure and Byzantine failure, respectively, for agent execution platforms such as JADE and Aglets. For both failures, we have extended traditional fault tolerance methods for intranet to make them applicable to Internet agent systems, which are huge, open, dynamic, autonomous, and unorganized distributed systems.

1. INTRODUCTION

When we utilize agents on the Internet for mission-critical tasks such as e-commerce, fault tolerance becomes crucial issue, because the Internet consists of unreliable hosts and is open to everyone. From the perspective of distributed systems, agent systems on the Internet are very different from traditional distributed systems on intranet: the number of working processes is very large and unbounded, and independently developed agents are continuously created and killed in an unorganized way. Therefore, the existing fault-tolerance methods are not directly applicable to these types of distributed systems. Furthermore, developers of agent systems have little familiarity with complicated fault tolerance algorithms in general. Consequently, we need to introduce a fault-tolerance method directly to agent execution platforms so that agent developers do not have to worry about it.

2. INTENDED USER AND SYSTEM REQUIREMENTS

Our method is intended for use by developers of agent execution platforms such as JADE and Aglets. If in their platforms agents communicate with each other only by message passing, not by shared memory, and we can explicitly monitor the progress of agent programs by a certain measure from outside the system, we can apply our methods to the platform. In this demo, we use JADE and our original FIPA-compliant platform "Erdoes."

3. STOP FAILURE

3.1 Problem and related work

Stop failure means that a host machine(s) suddenly crashes and some important data are lost. To guard against such loss, there are two representative algorithms among many existing rollback recovery algorithms. The Chandy Lamport algorithm [2] has the advantage that it does not block the execution of applications run by the agents when they are taking a global snapshot. The algorithm is not, however, applicable to dynamic environments in which agents are created and killed dynamically and no system knows the IDs of all agents currently running on the network. It is also difficult to cope with a huge distributed system, because the snapshot involves all agents. The Koo and Toueg algorithm [3], on the other hand, solves this problem by focusing on agents' relations created by communication. However, this algorithm could not avoid the blocking, and its consistency is weak; that is, a record of receiving messages can be lost. Our algorithm [4] extends the Chandy Lamport algorithm with Koo and Toueg's idea, and solves these problems at the same time: dealing with huge and dynamic systems, non-blocking, and achieving strong consistency.

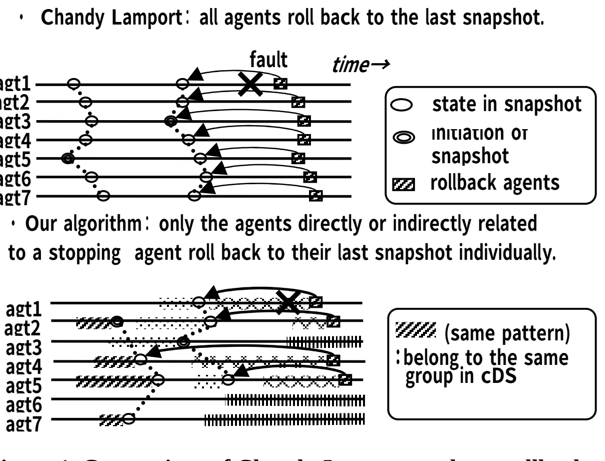
・Chandy Lamport: all agents roll back to the last snapshot.
fault
time→
agt1
agt2
agt3
agt4
agt5
agt6
agt7
state in snapshot
initiation of snapshot
rollback agents
・Our algorithm: only the agents directly or indirectly related to a stopping agent roll back to their last snapshot individually.
agt1
agt2
agt3
agt4
agt5
agt6
agt7
(same pattern)
: belong to the same group in cDS

Figure 1. Comparison of Chandy-Lamport and our rollbacks.

3.2 Technical points and discussion

Here we introduce the notion of communication dependency sets and partial snapshots. A communication dependency set (cDS for short) of an agent is the set of agent IDs with which the agent communicated after the last snapshot. When a snapshot initiator starts a snapshot, it dynamically decides the group of the snapshot by collecting the cDS of agents that may be involved in the snapshot, and a partial snapshot is taken among the group. At recovery, a group to be rolled back is decided in a similar way. As Fig. 1 shows, it is possible that agents roll back to different partial snapshots, but our algorithm guarantees consistency in any case.

Advantages: Because a snapshot group is decided dynamically, it is applicable to a dynamic environment and because the snapshot is partial, it can deal with a huge distributed system. Since it is based on the Chandy Lamport algorithm, it is non-blocking.

Disadvantages: The intersection of two snapshots initiated by different agents requires heavy communication between the initiators and makes the total procedure a little slow, while there is no such extra procedure in the Chandy Lamport algorithm.

3.3 Demo features

The fault-tolerant system is based on rollback recovery. Among the ten PCs, we assume eight of them are unreliable, while two specific ones are stable. On each of the unreliable PCs, eight agents are running, and they take partial snapshots from time to time. We can shut down any PC at any time. Then it is shown that the agents running on the PC are recovered on one of the stable PCs consistently. By displaying the messages sent and received by the recovered agents, we can show there is no inconsistency in the communications. That is, the messages recorded as sent are also recorded as received, even if messages were still in the link when the unstable PCs crashed. This can be seen clearly in a graphical representation, and the time performance is practically acceptable.

4. BYZANTINE FAILURE

4.1 Problem and related work

Byzantine failure means that a host is taken over by a malicious intruder, and the agents running there are completely controlled in an unwanted way. It had been believed that a practical Byzantine fault-tolerant system is difficult to realize, and it is also proved that there is no algorithm that completely solves Byzantine agreement in asynchronous systems such as Internet agent systems. However, Castro and Liskov [1] introduced a practical Byzantine fault-tolerant system to asynchronous systems under the allowable assumption of message delay. Unfortunately, this algorithm essentially addresses server client-type systems, which are different from homogeneous systems like agent systems. We extended Castro and Liskov's system so that it works for homogeneous and autonomous systems.

4.2 Technical points and discussion

Castro and Liskov's algorithm assumes the client is honest and makes only replicas of servers. In the homogeneous case, we cannot tell which is the client and which is the server. Thus, we must make replicas of both sides of any communication (Fig. 2), and for this we require a more complicated agreement procedure. Moreover, autonomy of an agent implies that in many cases, agents do not wait for specific messages, that is, they behave based on the messages currently received by chance. Therefore, to make the behavior of replicas the same, we need to introduce a procedure of agreement on when and which messages they process. We have introduced these extensions to the Castro and Liskov algorithm.

Advantages: Our method can cope with the properties of agent systems: homogeneity and autonomy while retaining practically acceptable speed.

Disadvantages: Because of the replicas on both sides and the additional agreement on a set of messages to be processed, message complexity becomes very high.

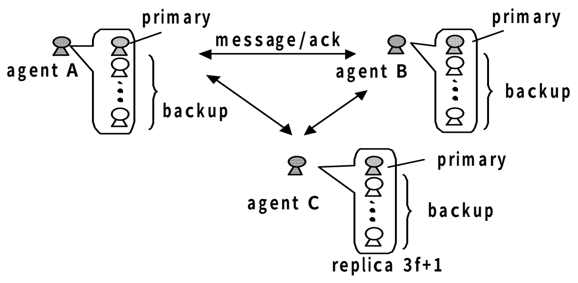

Figure 2. Replica models for Byzantine fault tolerance.

4.3 Demo features

The fault-tolerant system is based on replica models. For each agent, we create four replicas (= 3f+1, where the number of attacked hosts f is 1), including the original one on different hosts. These replicas are continuously making agreements on the order and timing of processing the received messages so that their behavior coincides at the common program. The agreement algorithm is designed so that when an intruder takes a host, and a replica on the host is controlled to behave differently from the other replicas, the remaining honest replicas can reach valid agreement to behave in an expected manner, even if the malicious one disturbs the agreement process in any way. In our demo, we can show the replicas reordering the received messages in a common order and decide the timing with which they are processed by agreement.

5. Future works

To improve efficiency, we plan to investigate good timing of snapshots, and as another base for our Byzantine fault tolerance, we should consider the randomization method [5].

6. REFERENCES

[1] M. Castro, B. Liskov: Practical Byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. 20(4): 398-461 (2002).

[2] K.M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst., 3(1):63-75 (1985).

[3] R. Koo, S. Toueg: Checkpointing and Rollback-Recovery for Distributed Systems. IEEE Trans. Software Eng. 13(1): 23-31 (1987).

[4] S. Moriya, T. Araragi: Dynamic Snapshot Algorithm and Partial Rollback Algorithm for Internet Agents, DISC 2001 Brief Announcement, DI-FCUL TR-01-7 J. Welch (ed.): 23--28(2001).

[5] C. Cachin, K. Kursawe, V. Shoup: Random Oracles in Constantinople: practical asynchronous Byzantine agreement using cryptography. PODC 2000: 123-132 (2000).