The Performance of Checkpointing and Replication Schemes for Fault Tolerant Mobile Agent Systems

Taesoon Park and Ilsoo Byun
Sejong University
Department of Computer Engineering
Seoul 143-747, KOREA
{tspark,widepis}@sejong.ac.kr

Hyunjoo Kim and Heon Y. Yeom
Seoul National University
Department of Computer Science
Seoul 151-742, KOREA
{hkim,yeom}@arirang.snu.ac.kr

Abstract

This paper evaluates the performance of checkpointing and replication schemes for the fault tolerant mobile agent system. For the quantitative comparison, we have implemented an experimental system on top of the Mole mobile agent system and also built a simulation system to include various failure cases. Our experiment aims to have the insight into the behavior of agents under two schemes and provide a guideline for the fault tolerant system design. The experimental results show that the checkpointing scheme shows a very stable performance; and for the replication scheme, some controllable system parameter values should be chosen carefully to achieve the desirable performance.

1. Introduction

A mobile agent is a software program which migrates from a site to another site to perform a task given by a user [4]. For the autonomous, asynchronous and continuous execution throughout the sites, the agent carries the program codes, the data and the intermediate execution states. Since the mobile agent gives a convenient way of accessing system resources for remote users, it has drawn attention as a new distributed computing paradigm.

For the mobile agent system to support the agents in more various application areas, the issues on the reliable execution should be considered. While executing on a site or migrating between the sites, an agent can be lost or its execution can be duplicated, due to a site failure. The reliable execution of a mobile agent is to guarantee the exactly-once execution even in the face of a system failure. Many fault-tolerance schemes have been suggested, which are either replicating the agents or checkpointing the agents.

An agent in the replication scheme is replicated and sent to several sites for each stage so that the agent can survive site failures. However, the replicas must agree on one execution site for each stage and the redundant execution of other sites should be undone. For the agreement procedure, a distributed transaction [11], the leader election protocol [3, 9] or the consensus protocol can be used [5, 6].

In the checkpointing scheme, the intermediate states of an agent are saved into a stable storage so that the agent can resume the execution from the saved state in case of a failure. To restrict the influence of failures to the other agents in the same communication group, the message logging [10] or the checkpointing coordination [2] can be employed. Since there is no agent replication, the communication cost for replication and agreement is not required.

Comparing the performance of two schemes, the fault tolerance degree has been debated. The replication scheme with $k$ replicas can tolerate up to $\left\lfloor \frac{k}{2} \right\rfloor$ failures, while the agents under the checkpointing scheme can be blocked even for a single failure. However, not much attention has been paid for the quantitative evaluation of the schemes, such as the cost of two schemes during the failure-free operation and in various failure cases.

For a replication scheme, the effect of the agent size has been measured in [6], however, the experiment assumed a simple failure environment and many of other system factors, such as the number of replicas and the timeout value, have not been considered. The performance of the replication scheme and the checkpointing scheme has been compared in [8], however, only the effects of the agent size and the replica number have been considered.

In this paper, we evaluate the performance of two fault tolerant schemes in various system environment. For the accurate experiment, we have implemented an experimental system on top of the Mole [1] system. To optimize the performance of the replication scheme, we have implemented three versions of replication, including the dedicated consensus agent and the asynchronous replica transfer. To consider various failure cases in the experiment, we also have
built a simulation system. Our experiment aims to have the insight into the behavior of agents under two fault tolerant schemes and provide a guideline for the fault tolerant system design.

2. The Mole System

To support execution and migration of an agent, the Mole system provides a Java platform called a location. The location is responsible for agent migration including preparing the migration of an agent, registering the agent to a new location, and starting up the execution of the agent. A RMI mechanism is used for agent migration and the Mole has adopted a weak-migration approach.

The execution of an agent performed in a location is called a stage and the agent execution can logically be viewed as a sequence of stages. For the agent execution, the location also serves as an access point to the local resources of the site and provides the inter-agent communication mechanisms, such as RMI or message-passing. A system site may provide one or more locations for the agents.

The Mole system provides two agent classes: one is the UserAgent and the other is the SystemAgent. The UserAgent class supports the application agents and the SystemAgent class is for the stationary system service agents of the location.

Failures considered in the system are the agent failure, the location failure and the system failure. For all of these failure types, the fail-stop [7] model is assumed. Agent and location failures are usually caused by incorrect agent states or location states; and hence the recovery information saved for the agent or location can survive the failure. However, the system failure is usually caused by a system crash which results in the loss of any recovery information saved in the volatile storage.

3. The Experimental System

To measure the cost of various fault-tolerant schemes, we have implemented a fault-tolerant mobile agent platform on top of the Mole system. In our experiments, the performances of the agents under two fault-tolerant schemes are evaluated: one is the checkpointing scheme with the reliable agent migration and the other is the replication scheme with the consensus. For the agents to access appropriate fault-tolerance methods, two agent classes have been created under the UserAgent class. One is the ChckpntAgent class and the other is the ReplicationAgent class.

Figure 1 shows the overview of our experimental system. A new method is denoted as a box and a existing Mole method is denoted as a dotted box. Each arrow denotes a call relation between the methods. The design of the experimental system basically follows the location-dependent approach and hence most of the new methods have been implemented inside the location of the Mole. A few methods for the consensus have been implemented inside the agent classes to better support the inter-agent consensus.

3.1. Checkpointing and Reliable Agent Migration

In the mobile system, to localize the recovery of an agent, it is desirable to checkpoint the agent’s state at the beginning of each stage. Then, for the recovery, the agent can resume the computation as if it has just arrived at the new location, without rolling back to an earlier stage. For an agent to be checkpointed, it has to inherit from the ChckpntAgent class. Then, the method checkpoint(), is called from the method handleMigration() when the agent migrates to a new location.

The handleMigration() method contains the codes for the agent migration and registration to a new location; and also the codes to start up the agent. The checkpoint() is called after the registration of an agent to the new location and right before the agent begins the execution. Hence, the state of an agent can be checkpointed into a disk at the beginning of each stage. The checkpoints of the agents executed in a location are maintained by a checkptTable of the location.

To support the fault tolerant execution of an agent, a reliable agent migration scheme should be provided in addition to the checkpointing. Otherwise, the agent may get lost or duplicated during the migration. In the Mole system,
an agent is duplicated before the migration and a copy of
the agent is transferred to a new location using a RMI call.
While the call is being processed, the receiver may fail and
this failure may results in the execution loss or duplicate.

To implement the reliable agent migration, a variation of
the two-phased commit protocol is used as shown in Figure
1. For the first phase of the reliable agent migration, the
existing RMI call from the method goTo() to the method
handleMigration() is used. The call and the call re-
turn are treated as a agentTransfer message and a transfer-
Ack message, respectively. The transferAck is returned after
a checkpoint is taken. For the second phase, a new method,
transferComplete(), has been implemented; and an-
other RMI call from goTo() to transferComplete() is
used as a transferComplete message.

The logging is performed by a new method log(),
which is called from goTo() on the receipt of the first
RMI call return and called from transferComplete() on
the receipt of the second RMI call. The sender considers
the migration to be aborted if any failure occurs before the
logging. The receiver also takes any recovery action only
for the logged agents, in case of a failure. However, for the
agents which have been checkpointed but not yet logged,
the receiver has to query the sender.

3.2. Agent Replication and Consensus

An agent in the replication scheme is replicated and mi-
grated to several sites. Among the replicas, one is called
a primary which is initially responsible for the execution.
Our experimental system basically follows the design of
FATOMAS [6] in which the priorities of the replicas are
predetermined. Hence, a replica takes over the execution,
only when all of the higher priority replicas fail. To detect a
possible failure, the time-out is used. When a primary does
not respond within a certain time-out period, the first replica
broadcasts the failure of the primary and starts up the agent
execution.

To implement the agent replication, the methods,
sendReplica() and handleReplica(), have been
implemented. The method sendReplica(), which is
called from the goTo() method before the migration of
an agent, first decides the locations to which the replicas
are sent and then transfers each replica by calling the re-

computeReplica(). The handleReplica(), on the receipt of an agent replica,
registers it into a backupTable without starting up the
execution and returns the call. After transferring every
replica, the primary is migrated to the new location.

Once the primary begins the execution, it may not re-
spond within a time-out period. Then, the first replica takes
over the execution after broadcasting Nack messages and re-
ceiving estimate messages from other replicas. However, it
is possible that the primary was too slow to respond within
the time-out period. In such a case, two agents may have
performed the same execution stage. To prevent such re-
dundant execution, the execution of a stage must be agreed
by the inter-agent consensus so that only one voted by the
majority of replicas can complete the execution and the oth-
ers should undo it.

For the consensus, two methods, consensus() and
handleConsensus(), have been implemented inside the
ReplicationAgent class. When an agent com-
pletes a stage and migrates to a next stage, it calls the mi-
grateTo() method, from which the method con-
ensus() is called. The consensus() method sends the
consensus message to the other replicas using the exist-
ing Mole method, message(), and waits for the replies.
On the receipt of the replies from the majority of replicas,
the consensus() returns the success. A method, han-
dleConsensus(), has been implemented to handle the
consensus message, which automatically replies back to the
sender agent on the receipt of a consensus message.

As shown in the protocol, the system with k replicas
is intended to tolerate \( \frac{k-1}{2} \) failures. In other words, only
the majority of replicas are for the possible execution and the
others are for attending the consensus. Since in this scheme,
the priorities of replicas are predetermined, the replicas
with lower priorities are sure to attend the consensus. Since in this scheme,
the priorities of replicas are predetermined, the replicas
with lower priorities are sure to attend the consensus.

Hence, in our experimental system, the sendReplica() method may send the replicas only to the majority of the lo-
cations. To the others, agent replicas containing the simple
codes to attend the consensus can be sent.

We have implemented another optimization; the asyn-
chronous replica migration. The primary, without waiting for
the other replicas to migrate to the new locations, be-
gins the execution as soon as it migrates. If the compu-
tation of the primary is long enough, the replication transfer
time can totally be masked by the asynchronous migration.
However, if the computation of a stage is very short, the
primary may begin the consensus before the majority of the
replicas arrive the new location. In this case, the consensus
messages may get lost and hence the primary resends the
consensus messages if it cannot receive the majority of the
replies within a certain time.

4. Performance Study

4.1. Experimental Results

A cluster of six Pentium IV 1 GHz PCs connected
through a 100 Mbps Ethernet was used for the experiments.
Each machine supported one location and an agent traversed
the locations in a predetermined order. The replicas were
also sent to the predetermined locations. Two types of
agents were used. One is the agent sleeping for one second
at each stage and the other is the agent reading and updating a counter value in a data file at each stage. Also, to control the size of an agent, an \(N\times N\) integer array is included in the agent.

Table 1 shows the time spent for the checkpointing and the reliable migration of an agent, when the size of the agent is 2KBytes and 200KBytes. As shown in the table, checkpointing does not put much overhead on the agent execution, while most of the extra overhead comes from the two-phased agent migration. Also, the overhead of reliable migration is heavily influenced by the size of an agent. The slow RMI compared to the fast disk access may cause these results. However, the overhead caused by checkpointing and two-phased agent migration is not affected by the computation length of the agent, since two types of agents show the similar performance.

<table>
<thead>
<tr>
<th>The Agent Size</th>
<th>Checkpointing Time</th>
<th>Reliable Migration Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2KBytes</td>
<td>5 ms.</td>
<td>11 ms.</td>
</tr>
<tr>
<td>200KBytes</td>
<td>16 ms.</td>
<td>113 ms.</td>
</tr>
</tbody>
</table>

**Table 1. The Cost of Checkpointing.**

Figure 2.(a) compares the performance of three replication schemes with the checkpointing scheme (denoted by CP). The execution time of an agent employing no fault-tolerant function is also shown (denoted by NO-FT). For the experiments, the agent accessing a disk file was used and three replication schemes are as follows:

- REP-Fk sends the actual copies of an agent to \(k\) locations.
- REP-k sends the actual copies of an agent to \(\lceil \frac{k}{2} \rceil\) locations and the consensus agents to the others.
- REP-Ak sends the replicas as in REP-k, however, the primary is sent first and then the other replicas are sent.

As shown in the figure, the replication optimization works well, for the small agent. However, for the large agents, the performance of REP-A is even worse than REP. The main reason of these results is that the computation time is too short to mask the agent migration time. Moreover, in our experimental system, the primary under REP-A sends out the consensus message again if it cannot receive the majority of the replies within 50 ms. As a result, the performance of REP-A becomes worse than REP when the agent size and the number of replicas are large.

However, for the long running application, the asynchronous migration works well, as shown in Figure 2.(b). For this result, we used the agent sleeping one second at each stage. As shown in the figure, the performance of REP-A is close to the one of the checkpointing scheme and it is even better when the agent size is 200 KBytes and three
Figure 3. Simulation Results.
replicas are used. However, this superior performance of REP-A cannot always be held for the large size agents, since as the agent size increases, the migration time of the replicas is also increased. Figure 2.(c) shows that the replication scheme is heavily affected by the agent size and the number of replicas.

4.2. Simulation Results

To measure the cost of two schemes in more various failure environments, we also have built a simulation system. The simulation system exactly follows our experimental system regarding the behavior of agents and the system parameters. For the simulation parameters, the unit costs for the 10 KBytes agent were obtained from our experimental system and used. The behavior of agents has been simulated with various values for the failure interval, the time-out and the blocking time, each of which is also assumed to follow an exponential distribution.

Figure 3.(a) and (b) show the performance of the checkpointing scheme with various failure rates and blocking time values. As shown in the figure, the performance is not sensitive to the change of the failure rate unless the blocking time is too large. An agent under the checkpointing scheme should redo the task of a current stage at most once after a failure, and hence, the increase in the execution time is not that large with the marginal blocking time. Figure 3.(b) also shows that the influence of blocking time may not be severe if the failure rate is low.

Figure 3.(c) shows the influence of the failure rate on the replication schemes when the timeout value is 3000 ms. Since in the replication scheme, the failure of a site causes the successive execution of the replicas, the overall execution time becomes increased. Figure 3.(d) shows the number of stages agreed on the first, the second and the third replicas for the system with seven replicas. As shown in the figure, when the failure rate is high, there is a high possibility that the majority of the replicas should try to perform the task.

In the replication scheme, the timeout value plays a more important role than the failure rate. For a short timeout period, the frequency of redundant execution may increases while the late failure detection may cause the delay in execution for a long timeout period. When the failure rate is high as shown in Figure 3.(f), the late failure detection causes the more severe problem. On the other hand, when the failure rate is low as in Figure 3.(e), the redundant execution can cause more delays in execution.

5 Conclusions

In this paper, we have evaluated the performance of the checkpointing and the replication schemes for the mobile agent system. For the evaluation, we have implemented an experimental system on top of the Mole system and also built a simulation system. From the experimental results, we can conclude the followings: Contrary to our expectation, checkpointing does not incur much overhead, however, the reliable migration and the replication incur much overhead. The checkpointing scheme also shows a very stable performance and it is sensitive only to the blocking time when the failure rate is high. The replication scheme is considerably affected by various system factors. However, by properly selecting some controllable parameter values, this scheme can achieve the performance as good as the one of the checkpointing scheme.

References