A Concurrency Control for Transactional Mobile Agents

Jeong-Joon Yoo and Dong-Ik Lee
Department of Information and Communications,
Kwang-Ju Institute of Science and Technology (K-JIST)
Puk-Gu Oryong-Dong 1 Kwangju, Korea
e-mail:{jjyoo, dilee}@csrl.kjist.ac.kr

Abstract

Mobile agents are autonomous objects that can migrate from one host to another within a computer network. Concurrent execution of multiple mobile agents requires a concurrency control for data consistency. Some mobile agent applications, such as information retrieval, do not need a concurrency control, but most applications, for example workflow and electronic commerce, require such a control on their mobile agents because a task dependency exists between them. In this paper, we define the requirement of transactional mobile agents and compare the performance of some concurrency control methods to select one having the highest concurrency ratio for mobile agents. According to the results of our experiments, we suggest using a timestamp-ordering protocol having alternative tasks for concurrency control on mobile agents. If long-lived and short-lived transactions coexist, it is much better to use a timestamp-ordering protocol having alternative tasks than a locking-based protocol according to performance aspects. Since the timestamp-ordering protocol having alternative tasks assures global serializability, we can maintain the data consistency across the system. To use the timestamp-ordering protocol having alternative task as a concurrency control for mobile agents in distributed environments, we also describe how to solve the cascading abort and global time synchronization problems.

1. Introduction

Mobile agent technology has been proposed for various application areas including workflow, electronic commerce and network management. Many of these applications require agents to guarantee the exactly-once semantic regardless of node failures. For this reason, some research on this exactly-once property for mobile agents was done by Rothermel [1]. It may not be sufficient, however, to guarantee global correctness of concurrent and the interleaved execution of agents (transactions). In real applications, when many mobile agents are concurrently executed, they need not only to preserve the exactly-once property but also to provide global serializability. There is a need to ensure that the concurrent execution of multiple agents does not violate the consistency of the data accessed. To enforce the concurrent execution of transactions in a manner that does not violate the consistency of data, the most common approach used is to ensure that the corresponding schedule is conflict-serializable, or conflict equivalent to a serial schedule. Assuring such equivalences requires enforcement of global serializability and global commitment of mobile agents. If all tasks belonging to different mobile agents are commutative (in other words, no dependency exists), the need to provide the global concurrency control does not exist. Most mobile agent systems were developed under this assumption. However, in most of the mobile agents based applications, especially workflow systems, there are dependencies among tasks belonging to different mobile agents. A mobile agent system for a workflow application therefore must provide a global concurrency control mechanism on mobile agents for their correct execution.

The features of a transaction model have effects on its concurrency control policies. To decide the best concurrency control method for mobile agents, we must define the requirements of the transaction models for mobile agents, referred to as transactional mobile agents throughout this paper. With these requirements laid out, we can then design a transaction model for transactional mobile agents and suggest the preferable concurrency control method that ensures correct execution of mobile agents.

The remainder of this paper is organized as follows. The next section includes a brief review of related work on transaction models. Section 3 describes the specification of transactional mobile agents. As well, this section also defines the requirements of a transaction model for transactional mobile agents. Section 4 compares the performance of concurrency control methods for transactional mobile agents and Section 5 discusses some problems to be solved when we use the preferable
concurrency control method in distributed environments. Finally we conclude with the future work in section 6.

2. Related Transaction Model

Several transaction models were proposed for the fulfillment of the requirements. The traditional transaction model refers to those transactions endowed with the atomicity, consistency, isolation and durability (ACID) properties. The extended and relaxed transaction model refers to transactions that extend and relax the ACID properties of the traditional transaction model.

2.1. Traditional Transaction Model

This model requires the properties of atomicity, durability, consistency and isolation. A transaction is atomic if either all actions are completed or none is completed. The durability of the transaction refers to the inability to abort committed transactions. Transactions must preserve the consistency of data item even in concurrent executions. Isolation refers to the intermediate state of a transaction not being visible to other concurrent transactions. This is a basic transaction model and is used in widespread applications.

2.2. Extended and Relaxed Transaction Model

This model is based upon the traditional transaction model. A Nested Transaction and Saga Transaction are belonging to this type of model as well [2,3]. They extend and relax the properties of traditional transaction model. A Nested Transaction is a set of subtransactions that may recursively contain other subtransactions, thus forming a transaction tree. This transaction provides full isolation on the global level, but they permit increased modularity, finer granularity of failure handling and a higher degree of intra-transaction concurrency than the traditional transactions. The Saga Transaction was developed to deal with long-living transactions. In this model, if one of the subtransactions fails, then committed subtransactions are undone by executing compensating subtransactions. This model relaxes the full isolation requirements and increases inter-transaction concurrency.

2.3. Distributed Transaction Model based on Mobile Agents

This study presents a transaction model that supports the correct and reliable execution of complex activities that may autonomously and asynchronously roam a network of autonomous and heterogeneous systems. Associated with adequate concepts for high level transaction specification which provide transparencies and abstractions for the transaction designer, this transaction model represents a powerful basis for supporting the flexibility needed for modeling and executing activities in a very dynamic and massively distributed environment, including mobile devices. This model considers only transactions that are implemented by a single mobile agent. It supports local consistency maintenance, but in the case of concurrent executions by multiple mobile agents, it has difficulty in maintaining global consistencies that ensure the correct execution of the multiple mobile agents [4].

3 Transactional Mobile Agent Model

The feature of a transaction model has direct effects on its concurrency control policies. In this section, we describe the specification and the requirement of transactional mobile agents, which are the fundamental aspects in deciding which concurrency control mechanism to use for mobile agents to ensure the correct execution of multiple agents.

3.1. Specification

Mobile agents are software programs that are responsible for executing a series of tasks and autonomously move into several hosts through a network. All tasks belonging to one mobile agent have a data dependency on other tasks belonging to other mobile agents. Whenever mobile agents visit one host, they cannot execute each task immediately to maintain data consistencies. The results of task are preserved in the execution state of the mobile agents. Mobile agents create new agents, referred to as sub agents, to increase the concurrency and are able to communicate with other agents, including sub agents. As well, whenever task errors occur, they do not always roll back all tasks previously completed but execute alternative tasks.

3.2 Requirements

Transactional mobile agent models are comparable with extended and relaxed transaction models. A transactional workflow model is one of the extended and relaxed transaction models [5]. This has similarity with the transactional mobile agent model. They require the relaxation of the atomicity and isolation properties. There are, however, some distinctive features between the two models. While transactional workflow models require the partial reduction of the isolation requirement, transactional mobile agent models require the full reduction of this property. Another distinctive feature is the migration of transactions.
Relaxed Atomicity
A transaction is atomic if either all actions are completed or none is completed. Because a transactional mobile agent sometimes lasts for a long time, the atomic property is not acceptable. If some action is not completed perfectly but has some errors, the cost of rolling back all other completed action is so expensive. The traditional notion of atomicity would require that a failure of any task result in the failure of the transactional mobile agent. However, a transactional mobile agent can survive the failure of one of its tasks, for example, by executing a functionally equivalent task (namely, an alternative task) at another site. Alternatives and compensating tasks must be defined in the transactional mobile agent model to relax the atomicity requirements. In the case where physical failures occur, each of mobile agents performs forward recovery, namely, the agent recovers its state and continues executing the transactional mobile agent from where it was interrupted.

Consistency
Transactions must preserve the consistency of a data item even in concurrent executions of the transactional mobile agents. Therefore, scheduling mechanisms for preserving these consistencies are required. In order to enforce the concurrent execution of multiple agents without violating the consistency of the data, the most common approach used is to ensure that the corresponding schedule is conflict-serializable, that is, conflict equivalent to a serial schedule. Two schedules are conflict equivalent, if they consist of the same tasks and order their conflicting operations in the same manner. This is based upon the assumption that each agent maintains data consistency if executed alone.

Relaxed Isolation
Isolation refers to the intermediate state of the transaction not being visible to other concurrent transactions. Permitting the visibility of states of ongoing transactions increases the concurrency but also increases the complexity of scheduling. A schedule correcting for these transaction was presented by Pitoura [6]. In that paper, the structure of an agent is defined through the dependencies between the execution state of its methods. The enforcement of the structural properties of an agent and the control of the interaction of agents with other agents and remote resources is assigned to agent managers that accomplish this task through a well-defined small set of primitives.

Mobility
Transactional mobile agents migrate from one host to another through networks. The place a transactional mobile agent begins to act is not always the same with one it would be destroyed. The mobility requires for the concurrency control to ensure not only a local serializability but also a global serializability. The guarantee of a local serializability is not sufficient to ensure a correct execution of transactional mobile agents. Therefore global serializability is a mandatory for correct execution of multiple mobile agents.

4 Performance Comparison of Concurrency Control for Transactional Mobile Agents

One way to ensure serializability is to require that access to data items be done in a mutually exclusive manner, that is, while one transaction accesses a data item, no other transaction can modify that data item. The most common method used to implement this is to allow a transaction to access a data item only if it is currently holding a lock on that item. In the lock-based protocol, the order between every pair of conflicting transactions is determined at the time of execution by the first lock they both request that involves incompatible modes. Another method for determining the serializability order is to select an ordering among transactions in advance. The most common method for doing so is to use a timestamp-ordering protocol. The timestamp-ordering protocol ensures conflict serializability. This follows from the fact that conflicting operations are processed in timestamp order, thus ensuring freedom from deadlock since no transaction ever waits.

Concurrency control must not only preserve the consistency of the data, but also increase the concurrency of multiple transactions. In this section, we compare three concurrency control methods, the lock-based protocol, timestamp-ordering protocol, and timestamp-ordering protocol having alternative tasks from the perspective of response times, i.e., elapsed times for completing all given transactional mobile agents.

4.1 Elapsed Time

We compared the elapsed time for concurrency control for the case where only long transactions exist and for the case where both short and long transactions coexist. Two concurrency control methods, the timestamp-ordering protocol (referred to as pure timestamp-ordering in Figure 1) and the lock-based protocol, and the timestamp-ordering protocol with alternative tasks are compared.

We used GSPN [7] for modeling the above two concurrency control methods. The degree of concurrency, i.e. the elapsed time, for each method was determined by increasing the number of mobile agent instances. The important parameters for simulation are as follows.
Simulation Parameters

The performance of the concurrency control is determined by several parameters. Some of them are determined by the hardware, operating system architecture and network conditions. However, in this case, more important parameters include the conflicting probability, the number of instances of transactional mobile agents and the duration of transactions life-cycle. To apply the simulation’s results to real world, these parameters should be included in the model. The parameters being constant for all simulations are summarized in Table 1.

The time between agent generation is a non-deterministic, exponentially distributed function with $\lambda = 1$ and 1-server. Other stochastic transitions are fired deterministically. These concurrency control methods were simulated with these parameters within LAN environments. The definitions for these parameters are as follows.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time between agent generation</td>
<td>Exponential function with $\lambda = 1$</td>
</tr>
<tr>
<td>Agent transmission time</td>
<td>0.3 sec</td>
</tr>
<tr>
<td>Locking duration of long-transaction</td>
<td>3600 sec</td>
</tr>
<tr>
<td>Locking duration of short-transaction</td>
<td>0.5 sec</td>
</tr>
<tr>
<td>The number of alternative tasks</td>
<td>1</td>
</tr>
<tr>
<td>Conflict probability</td>
<td>0.2</td>
</tr>
<tr>
<td>The length of travel path</td>
<td>3</td>
</tr>
<tr>
<td>Long transaction probability</td>
<td>0.2 (for the case of short and long transaction coexist)</td>
</tr>
<tr>
<td>Restart delay time</td>
<td>5 sec (for the case of short transaction) or 4000 sec (for the case of long transaction)</td>
</tr>
</tbody>
</table>

4.2 Data Consistency

A global schedule $S_I$ is serializable if and only if it is conflict equivalent to a serial global schedule $S_0$, i.e., it has the following property: for all pairs of conflicting operations $o_i$ and $o_j$, if $o_i$ precedes $o_j$ in a schedule $S_I$, then $o_j$ precedes $o_i$ in $S_0$. In order to obtain consistent scheduling of the operations in the system, we need first to make sure that each local schedule is serializable and that the equivalent serial schedules at all sites are compatible. Thus, if at one site $T_i < T_j$ exists, at no other site should we have $T_j < T_i$. We can determine if a schedule is globally serializable by checking if the corresponding serialization graph contains a cycle. Transactions in a timestamp-ordering protocol are totally ordered. So the above local schedule and global schedule using timestamp-ordering protocol is conflict serializable.
Therefore we can assure that timestamp-ordering protocol preserve the data consistency at all sites.

Until now, we have shown that the timestamp-ordering protocol having alternative tasks not only outperform the other lock-based protocols but also preserve the data consistency at all sites. Therefore, the timestamp-ordering protocol having alternative tasks as a concurrency control method for transactional mobile agent models would be the best implement. However, some difficulties still need to be addressed, such as the cascade aborting problem and global time synchronization, before using these timestamp-ordering protocols.

5 Concurrent Execution of Transactional Mobile Agents

When multiple mobile agents are concurrently executed, we can control the concurrency with the timestamp-ordering protocol having alternative tasks. To use this protocol in real life, however, we must use a two-phase commit protocol to avoid cascading abort problems. Also global time must be synchronized for fairness in timestamp generation for each host.

5.1 Two-Phase Commit Protocol

In order to ensure atomicity, all the sites in which a transaction is executed must agree on the final outcome of the execution. The transaction must either commit at all sites or it must abort at all sites in distributed environments. In order to ensure this property, the transaction coordinator must execute the two-phase commit protocol. When a transaction completes its execution at all sites, the transaction can commit. To eliminate a cascade aborting, we can combine the timestamp-ordering scheme with the two-phase commit protocol.

Fig. 1. Performance comparison: (a) Both short-lived agents (transactions) and long-lived agents (b) Only long-lived agents
5.2 Global Time Synchronization

In distributed environments, a timestamp must be unique and must be synchronized to ensure that the global timestamps generated at one site are not always greater than those generated in another site. If one site generates local timestamps at a faster rate than other sites, a transaction generated at these other sites always have a lower timestamp that always results in a rolling back and restart of the transaction. To solve this problem we must ensure that local timestamps are generated fairly across the distributed host. To accomplish this we can use the method suggested by Lamport[8]. To ensure that the various logical clocks are synchronized, we require that a site, referred to as $S$, advances its logical clock whenever a transaction $T$ with timestamp $x$ visits that site and $x$ is greater than the current value of logical clock. In this case, site $S$ advances its logical clock to the value $x+1$. Therefore $S$ does not always have a logical clock greater (or less) than any other sites. As a result, we can provide fairness in timestamp generation for distributed environments.

6 Conclusion and Future Work

Most of mobile agents based applications, especially workflow and electronic commerce, require a concurrency control on mobile agents, because in such applications, tasks belonging to mobile agents have dependencies on tasks belonging to other mobile agents. To select a concurrency control method for transactional mobile agents, we defined the requirement of the transactional mobile agent model. Relaxed atomicity, reduced isolation and migration are distinguished features from the traditional transaction model. For relaxed atomicity, transactional mobile agents require alternative tasks. Alternative tasks and migration features effect the performance of the concurrency control. According to the results of our experiments, the timestamp-ordering protocol having alternative tasks outperforms other lock-based protocols. This work is a part of several research projects, in progress at K-JIST, on the development of a mobile agent platform for workflow systems. We are currently studying methods for recovery in the case a failure occurs. It will provide reliability for mobile agents, which is important for reliable workflow systems.

References