Agent-based Data Management *

1K−05 Takao Komiya, Hiroyuki Ohshida, Katsuya Tanaka, and Makoto Takizawa †
Tokyo Denki University ‡
Email : {komi, ohsida, katsu, taki}@takilab.k.dendai.ac.jp

1 Introduction

In client-server database applications, application programs are performed on clients. Requests and responses are exchanged among clients and servers in networks. The more number of requests are issued to object servers by applications and the more number of responses are sent back to the applications, the more communication overheads are increased. In the three-tier client-server architecture, applications move to application servers from clients in order to decrease the communication overheads between clients and servers.

In database applications, transactions are required to manipulate objects in object servers so as to satisfy ACID. For example, objects in multiple object servers are required to be atomically manipulated and transactions are serializable. In the traditional systems, objects are locked to realize the serializability of transactions. In the locking protocol, multiple accesses to an object are coordinated based on a principle that only one transaction is a winner which can hold the object and the others are losers.

In another computation paradigm, programs named mobile *agents* move around data servers. First, an agent lands at a server and then is performed to manipulate data objects in the server. If the agent finishes manipulating the data objects in the server, the agent moves to another server which has data to be manipulated. Here, agents manipulate objects in object servers without exchanging messages in a network. Compared with traditional process-based applications like client-server applications, mobile agents have following characteristics;

1. Agents are autonomously initiated and performed.
2. Agents negotiate with other agents.
3. Agents are moving around computers.

In this paper, we discuss how to manipulate multiple object servers by using agents. Agents move around object servers without exchanging messages in the network. On the other hand, application programs and object servers are exchanging messages in the network. In addition, an agent negotiates with other agents if the agents manipulate objects in a conflicting manner. Through the negotiation, each agent autonomously makes a decision on whether the agent continues to hold the objects or gives up to hold the objects.

2 Object Servers

A system is composed of object servers $D_1, \ldots, D_m$ ($m \geq 1$), which are interconnected with reliable, high-speed communication networks. Each object server supports a collection of objects and methods for manipulating the objects. Objects are encapsulations of data and methods. Objects are manipulated only through methods supported by the objects.

Applications in clients initiate transactions in application servers. A transaction manipulates objects in one or more than one object server. A *transaction* T is an atomic sequence of methods for manipulating objects in object servers. A subsequence T_i of methods in T to manipulate objects in one object server D_i is referred to as *subtransaction* of T. A subtransaction T_i is also atomic sequence of methods in one object server D_i.

3 Computation model of agents

An agent is a procedure which can be performed on one or more than one object server. An agent issues methods to an object server to manipulate objects in an object server where the agent exists. Every object server is assumed to support a platform to perform agents.

First, an agent A is initiated by an application or is autonomously initiated on an object server. The procedure and data of an agent A are first stored in the memory of an object server D_i in order to perform the agent A on D_i. If enough resource like memory to perform the agent A is allocated for the agent A on the server D_i, the agent A can be performed. Here, D_i is referred to as *current* server of A and the agent A is referred to as *land at* the server D_j. Objects in the server D_i are manipulated by the agent A through methods. In result, state of object may be changed and a part of the state may be derived. Data derived from the server D_i may be stored in the agent A. Thus, an instance A_i of the agent A on the object server D_i shows a subtransaction, i.e. a sequence of methods for manipulating objects in the server D_i. Then, the agent A finds another server D_j which has objects to be manipulated by A. Then, the agent A moves to the server D_j. Here, the agent A may carry objects obtained from D_i as the data of A [Figure 1]. If enough resource like memory in the server D_j is allocated for the agent A, A lands at D_j.

A pair of agents A_1 and A_2 are referred to as *conflict* if A_1 and A_2 manipulate a same object through conflicting methods. For example, A_1 issues a method *reset* and A_2 issues *increment* to a *counter* object in a server D_j. Here, A_1 and A_2 conflict. The agent A is allowed to land at D_j if the following condition is satisfied:

[Landing conditions]

1. Enough resource to perform an agent A is allocated for the agent A in an object server D_j.
2. There is no agent on D_j which conflicts with A.

4 Consensus among Agents

An agent A manipulates objects in multiple object servers $D_1, \ldots, D_m$ ($m > 1$). After finishing manipulating objects in all the object servers, the agent A commits if some condition C on the servers $D_1, \ldots, D_m$ is satisfied. Otherwise, A aborts. For example, an atomic all-or-nothing condition is used to realize the atomicity of a transaction. That is, the agent commits

*エージェントにおけるデータ管理
†小宮 貴雄, 大信田 裕之, 田中 勝也, 滝沢 誠
‡東京電機大学

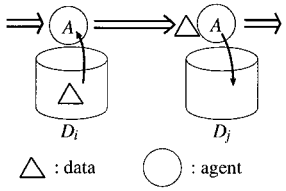

Figure 1: Agent.

only if all the object servers are successfully updated. Otherwise, the agent aborts, i.e. no update is done on the objects in any object server. The two-phase commitment (2PC) protocol is used to realize the all-or-nothing principle in distributed database systems. In another example, an application would like to book one hotel. The application issues a booking request to multiple hotel objects. Here, the application can commit if at least one hotel object is obtained. Thus, if at least one of the servers is successfully manipulated, the agent A commits. There are following consensus conditions;

[Consensus conditions]

1. *Atomic consensus*: an agent is successfully performed on all the object servers, i.e. all-or-nothing principle. This is a condition used in the traditional two-phase commitment protocol.
2. *Majority consensus*: an agent is successfully performed on more than half of object servers.
3. *At-least-one* consensus: an agent is successfully performed on at least one object server.
4. $\binom{n}{r}$ *consensus*: an agent is successfully performed on more than r out of n servers ($r \leq n$).

The atomic, majority, and at-least-one consensus conditions are shown in forms of $\binom{n}{n}$, $\binom{n}{\lceil (n+1)/2 \rceil}$, and $\binom{n}{1}$ consensus ones, respectively. More general consensus conditions are discussed in a paper [1].

5 Negotiation Strategies

Unless the landing conditions are satisfied, the agent A can not land at the server D_j. Here, the agent A can take one of the following ways:

1. The agent A waits in the current object server D_i.
2. The agent A finds another object server D_k which has objects to be possibly manipulated before D_j by A.
3. The agent A negotiates with other agents in D_j which hold resources.
4. The agent A *aborts*.

Suppose an agent lands at a current object server D_i. Here, there might be other agents B_1, ..., B_k which are being performed on the object server D_i. Each agent B_i is an agent or surrogate agent of an agent. If the agent A conflicts with some agent B_j on an object o, A negotiates with B_j with respect to which agent A or B_j holds the object o. There are following negotiation strategies:

1. The agent A blocks until the agent B_j commits.
2. The agent A takes over B_j, i.e. B_j releases the objects and blocks. Then A starts.
3. B_j aborts and A starts.

The first way is similar to the locking protocol. An agent A blocks if some agent B holds an object o in a conflicting way with the agent A. If B waits for release of an object held by A, A and B are deadlocked. Thus, deadlock among agents may occur. When an agent A blocks in a object server D_i, a timer is started. If the timer expires, the agent A takes one of the following ways:

1. The agent A retreats to an object server D_j which A has passed over.
2. Every surrogate A_j of A initiates a deadlock detection agent.

In the second way, an agent A takes over an agent B_j in an object server D_j if A conflicts with B_j and B_j holds an object. Here, A starts to do the negotiation with an agent B_j on D_j by using a following negotiation protocol :

[Negotiation protocol]

1. An agent A sends a *can-I-use* message $CIU(o, op)$ to an agent B_j on an object server D_j. This means that an agent A would like to manipulate an object o with a method op in an object server D_j.
2. On receipt of $CIU(o, op)$ from an agent A, an agent B_j sends OK to A if B_j can release the object o or B_j does not mind if A manipulates the object o. Here, there are two approaches to B_j's releasing the object o :
   a. B_j aborts if A precedes B_j.
   b. B_j rolls back to a checkpoint and then restarts if A precedes B_j. Otherwise, B_j sends No to A.
3. On receipt of OK from B_j, A starts manipulating the object o.
4. On receipt of No from B_j, there are following ways:
   a. A blocks until A receives OK/NO from B_j.
   b. A aborts.
   c. A triggers the second level negotiation protocol. □

If the agent B_j agrees with the agent A in the negotiation protocol, A can manipulate objects by taking over B_j. In the second way, the agent B_j not only releases the object but also aborts.

6 Concluding Remarks

This paper discussed an agent model for transactions which manipulate multiple object servers. An agent first moves to an object server and then manipulates objects. The agent autonomously moves around the object servers to perform the computation. If the agent conflicts with other agents, the agent negotiates with the other agents. The negotiation is done based on the commitment conditions and types of agents, i.e. ordered and an ordered.

References

[1] Shimojo, I., Tachikawa, T., and Takizawa, M., "M-ary Commitment Protocol with Partially Ordered Domain," *Proc. of the 8th Int'l Conf. on Database and Expert Systems Applications (DEXA'97)*, 1997, pp.397-408.