Implementation and Evaluation of Transactional Agents for Distributed Systems

Youhei Tanaka[1], Naohiro Hayashibara[1], Tomoya Enokido[2], and Makoto Takizawa[1]
[1]Dept. of Computers and Systems Engineering Tokyo Denki University, Japan
{youhei, haya, taki}@takilab.k.dendai.ac.jp
[2]Faculty of Business Administration Rissho University, Japan
eno@ris.ac.jp

A transactional agent is a mobile agent to manipulate objects distributed on computers with some commitment condition. In this paper, we present a computation model of transactional agent and discuss how to implement a transactional agent to manipulate objects on multiple database servers. In order to reduce the communication overhead to transfer a transactional agent from a computer to another computer in networks, a transactional agent is decomposed into a pair of routing and manipulation subagents. We evaluate the transactional agent model in terms of access time compared with the traditional client-server model.

1 Introduction

Various types of objects are distributed on multiple computers in networks. An object is an encapsulation of data and methods for manipulating the data. An object can be manipulated only through the methods. An application program manipulates objects distributed in computers. A transaction [2] of the application program is modeled to be an atomic sequence of methods. Transactions are traditionally realized in the client-server model [6]. Here, servers can be made more reliable by using multiple replicas of the servers. However, applications cannot be performed if the clients are faulty. Mobile agents [3] are programs which move from computers to computers and then locally manipulate objects in the computers. If an application program is realized in a mobile agent, the application program can be performed on operational computers by escaping from faulty computers. We discuss how to realize an application program on distributed objects in a mobile agent. In this paper, a *transactional agent* is defined to be a mobile agent which autonomously moves around computers in networks and locally manipulates objects in each of the computers [7]. In addition, a transactional agent is specified with one of commitment conditions [5].

In section 2, we discuss a model of transactional agent. In section 3, we discuss the implementation of the transactional. In section 4, we evaluate the transactional agent in terms of access time compared with the client-server model.

2 Transactional Agents

2.1 Transactional Agent

A system is composed of computers $D_1, ..., D_n$ ($n \geq 1$) interconnected in a reliable network. Messages are delivered in a sending order without message loss in the network. Each computer D_i is equipped with a class base CB_i and an *object base* OB_i. In the class base OB_i, classes are stored. The object base OB_i is a collection of persistent objects. On receipt of a method request, the method is performed on an object in the object base OB_i.

In the client-server model, a transaction is performed on a client or an application server. The transaction issues methods to servers. Methods are performed on objects in the servers and the responses of the methods are sent to the transaction. Even if a server is faulty, a transaction can be operational if the server is replicated. However, a transaction cannot be performed if the client or application server is performed is faulty.

A *mobile agent* is an object-based program, which moves around computers in networks and locally manipulates objects in each computer [3]. A class c is stored in a home computer $home(c)$. Here, $home(A)$ shows a home computer of the class of a mobile agent A. A mobile agent A is initiated on a *base* computer $Base(A)$ by loading classes of the mobile agent from the home computer. If a method on some class c is invoked by a mobile agent on a computer, the class c is loaded from the home computer $home(c)$.

In this paper, a transaction is realized in a mobile agent [Figure 1]. A transaction can be operational if it

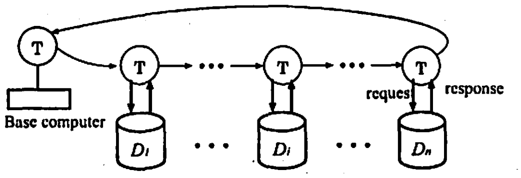

Figure 1. Transactional agent (TA) model.

is performed on an operational computer even if a base computer is faulty. Suppose a transaction is moving to a computer D_i. Here, if D_i is detected to be faulty, the transaction can move to another computer.

A *transactional agent* is defined to be a mobile agent which satisfies the following properties:

1. Autonomously makes a decision on what computer to visit in presence of faults of computers and change of service supported by computers and networks.
2. Moves from computers to computers in networks and locally manipulates objects in a computer.
3. Commits only if some commitment condition intrinsic to the transactional agent, otherwise aborts.

Target objects are objects to be manipulated by a transactional agent. A computer with target objects is a *target* computer of a transactional agent. A domain $Dom(A)$ is a set of *target* computers D_1, ..., D_n of a transactional agent A. Classes of a transactional agent is transferred from computers to computers while the program is fixed on a client or application server in a client-server model. In order to reduce the communication overhead to transfer classes among computers, a transactional agent A is decomposed into a pair of routing subagent and manipulation subagent.

According to the traditional concurrency control theories, a transaction commits only if objects in all the target computers are successfully manipulated. This is the atomic commitment condition. In addition, we consider other types of commitment conditions [5] which are discussed later.

2.2 Routing subagent

A routing subagent $RA(A)$ makes a schedule to visit target computers. An object x flows from a computer D_i to another computer D_j in a transactional agent A ($D_i \stackrel{x}{\Rightarrow} D_j$) iff a manipulation subagent $MA(A, D_i)$ on a computer D_i outputs an intermediate object x and a $MA(A, D_j)$ on another computer D_j uses the intermediate object x as an input.

An output-input relation among manipulation subagents is shown in a navigation map $Map(A)$ as shown in Figure 2. A path $D_i \rightarrow D_j$ shows an output-input relation $D_i \stackrel{x}{\Rightarrow} D_j$ from a computer D_i to another computer D_j. In Figure 2, $MA(A, D_1)$ of the transactional agent A derives an intermediate object w from a computer D_1. $MA(A, D_3)$, $MA(A, D_4)$, and $MA(A, D_5)$ use the intermediate object w as the input objects in the computers D_3, D_4, and D_5, respectively.

A pair of computers D_i and D_j are *independent* ($D_i \parallel D_j$) if neither $D_i \rightarrow D_j$ nor $D_j \rightarrow D_i$ in an output-input graph. Here, $RA(A)$ can visit a pair of the computers D_1 and D_2 in any order. Figure 2 shows a navigation map $Map(A)$. Here, $RA(A)$ is required to visit the computer D_3 after visiting the computer D_1. On the other hand, $RA(A)$ can visit a pair of the computers D_1 and D_2 in any order since the computers D_1 and D_2 are *independent* ($D_1 \parallel D_2$). A node which does not have any incoming edge is referred to as *initial*. In Figure 2, D_1 and D_2 are initial.

In Figure 2, the intermediate object w derived from the source computer D_1 is required to be brought into the destination computers D_3, D_4, and D_5. Intermediate objects are required to be efficiently transferred to destination computers.

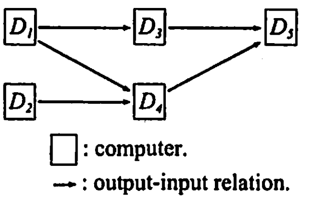

Figure 2. Navigation map.

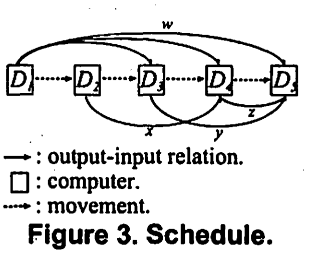

Figure 3. Schedule.

A schedule to visit target computers is obtained in $RA(A)$ by the topological sort [4] of nodes in the navigation map G (= $Map(A)$). The intermediate object w from the computer D_1 is used by D_3, D_4, and D_5 but the intermediate object x from D_2 is used by one computer D_4. Hence, D_1 is selected. A schedule for the navigation map G shown in Figure 2 is obtained as shown in Figure 3. Here, a dotted arc shows the visiting order of the computers. A straight arc shows an output-input relation among computers.

A transactional agent locally manipulates objects in each computer by moving around computers. If $MA(A, D_i)$ finishes manipulating objects in each computer D_i, $RA(A)$ checks the *commitment* condition $CC(A)$ by communicating with the other sibling $MA(A, D_1)$, ..., $MA(A, D_n)$ on D_1, ..., D_n. The commitment condition $CC(A)$ shows which computers have to be successfully manipulated:

1. *Atomic commitment*: all the computers.
2. *Majority commitment*: more than half of computers.
3. *At-least-one commitment*: at least one computer.
4. $\binom{n}{r}$ *commitment*: more than r out of n computers.

2.3 Manipulation subagents

On arrival of a routing subagent $RA(A)$ on a computer D_i, classes of a manipulation subagent $MA(A,D_i)$ are loaded. Objects are manipulated in the $MA(A, D_i)$. $MA(A, D_i)$ is an application program to locally manipulate objects in a computer D_i. In a manipulation subagent, a method of class is invoked. In this implementation, each time a method is invoked, the class is loaded to the subagent.

3 Implementation

A transactional agent A is implemented in a mobile agent of Aglets [3]. A routing subagent $RA(A)$ of the transactional agent A carries a navigation map object G. $RA(A)$ makes a decision on which computer to visit by using the navigation map G. $RA(A)$ selects a computer and then moves to the destination computer.

An object base OB_i is realized in an object agent OBA_i and database DB_i. A database DB_i is a relational database system or XML database system. The *object subagent* OBA_i supports a manipulation subagent $MA(A, D_i)$ with an object-based interface independent of types of database management systems.

$RA(A)$ leaves a computer D_i after objects are manipulated in $MA(A, D_i)$ and the object subagent OBA_i. However, OBA_i still holds the objects manipulated by OBA_i even if $RA(A)$ leaves the computer D_i. OBA_i is realized in a local transaction on the object base OB_i. OBA_i does not terminate.

In summary, a transactional agent A behaves as follows:

1. A $RA(A)$ initiates $MA(A, D_i)$ and OBA_i by loading their classes to a current computer D_i from the home computers of the classes.
2. If $MA(A, D_i)$ issues a method to OBA_i, OBA_i translates the method to SQL/XSQL commands to the database system in the current computer D_i.
3. Even if $RA(A)$ leaves the computer D_i, OBA_i still holds locks on the objects manipulated. $MA(A, D_i)$ does not terminate either. $MA(A, D_i)$ negotiates with other routing subagents while waiting for the final decision for $RA(A)$.
4. $RA(A)$ eventually makes a decision on commit or abort according to the commitment condition $CC(A)$ of the transactional agent A. $RA(A)$ notifies the sibling $MA(A, D_1)$, ..., $MA(A, D_n)$ of the decision, commit or abort.
5. On receipt of the commitment decision from $RA(A)$, each $MA(A, D_i)$ forwards the decision to OBA_i. OBA_i commits and aborts on receipt of *commit* and *abort*, respectively, from $RA(A)$. $MA(A, D_i)$ and OBA_i terminate here.

Suppose a routing subagent $RA(A)$ of a transactional agent A is on a current computer D_n after leaving manipulation subagents $MA(A, D_1)$, ..., $MA(A, D_n)$ on computers D_1, ..., D_n, respectively. Here, the transactional agent A can commit if all or some of the manipulation subagents commit depending on the commitment condition $CC(A)$ by using the two-phase commitment (2PC) protocol [8].

4 Evaluation

We measure how long it takes to perform a transactional agent (TA) model compared with the client-server (CS) model for the same application. Here, three computers D_1, D_2, and D_3 support Oracle database systems as object bases and one computer H is a home computer of the classes of manipulation subagents and object subagents. Another computer is a base computer C of a transactional agent A. The computers are interconnected in the 1Gbps Ethernet.

In the CS model, a same application program as the TA model is performed on an application server. The application program issues SQL methods to relational database systems in the computers D_1, D_2, and D_3. We consider the following types of applications P and Q for the TA and CS models:

1. Application P: An intermediate object I is derived from the object base OB_1. The object bases in the computers D_2 and D_3 are updated by inserting I [Figure 4a)].
2. Application Q: I_1 and I_2 are derived from the object bases OB_1 and OB_2 in the computers D_1 and D_2, respectively. Then, the object base OB_3 in D_3 is updated by inserting I_1 and I_2 [Figure 4b)].

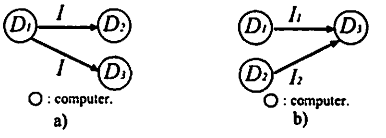

Figure 4. Navigation maps for application programs P and Q.

In the TA model, a routing subagent $RA(A)$ is first initiated in a base computer C. $RA(A)$ finds in which order $RA(A)$ visits the computers D_1, D_2, and D_3. On arrival of $RA(A)$ on a computer D_i, the classes of the manipulation subagent $MA(A, D_i)$ and the object subagent OBA_i are loaded to the computer D_i from the home computer H. Then, the object base OB_i is manipulated through the object subagent OBA_i by $MA(A, D_i)$. On completion, the $RA(A)$ moves to another computer. Here, $MA(A, D_i)$ and OBA_i still exist. OBA_i holds objects manipulated in the computer D_i.

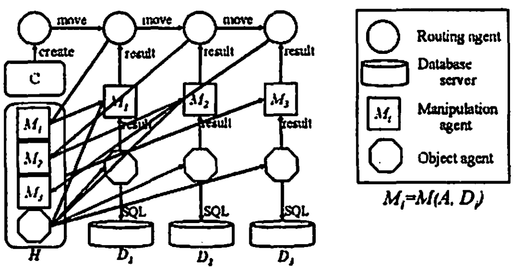

Figure 5. The transactional agent (TA) model.

In the CS model, each of the applications P and Q is performed on the base computer. Then, the application program manipulates the databases in the computers D_1, D_2, and D_3 in this order.

In the TA model, intermediate objects have to be transferred to other computers where manipulation subagents of the transactional agent are to be performed. We consider three ways to deliver I from a source computer D_i to another destination D_j:

1. $RA(A)$ carries I to the destination computer D_j.
2. After $RA(A)$ arrives at the computer D_j, $RA(A)$ requests the source computer D_i to send I to the computer D_j.
3. $RA(A)$ transfers I to the destination computer D_j before leaving source computer D_i.

The total processing time T is measured for number of intermediate tuple objects. The processing time T is composed of the following times:

1. T_0 = time to load and initialize $RA(A)$ at the base computer c.
2. T_{1i} = time to transfer $RA(A)$ from a computer D_i to another computer.
3. T_{2i} = time for $RA(A)$ to invoke $MA(A, D_i)$ on a computer D_i.
4. T_{3i} = time for $MA(A, D_i)$ to invoke an object subagent OBA_i on a computer D_i.
5. T_{4i} = time for OBA_i to manipulate objects by issuing SQL commands to the object base OB_i.
6. T_5 = time to do the commitment of all sibling manipulation subagents and for $RA(A)$ to return to the base computer C.

The total processing time T is given as $T = T_0 + \sum_{i=1,\ldots,n}(T_{1i} + T_{2i} + T_{3i} + T_{4i}) + T_5$ where n is the number of target comptuers of the transactional agent A. The size of the routing subagent is 7kBytes. The manipulation subagent is 4kBytes. The size of the manipulation subagent depends on the application. The object subagent is 5kBytes.

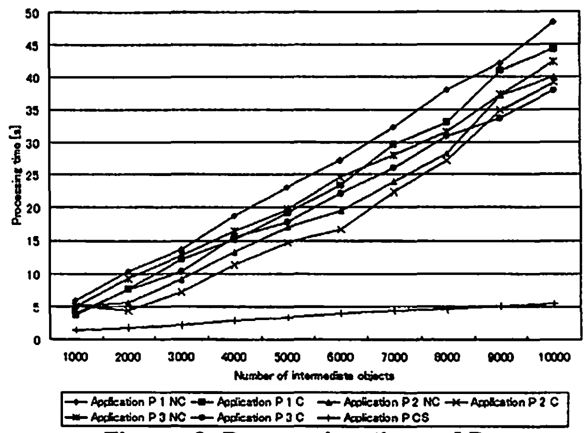

Figure 6. Processing time of P.

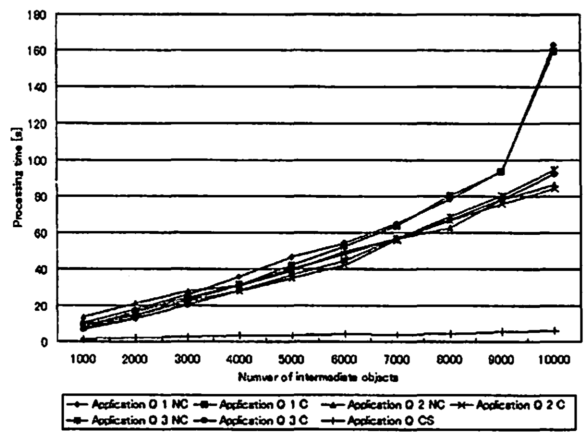

Figure 7. Processing time of Q.

The computation of a transactional agent is composed of steps: initialization moving of the routing subagent, loading of manipulation subagents, loading of object subagents, manipulation of objects, and commitment. Tables 1 and 2 show how long it takes to perform each step for the applications P and Q, respectively. 96% – 98% of the total access time in the transactional agent is spent by the manipulation subagent and object subagent, the manipulation time in the TA model is longer than the CS model.

5 Concluding Remarks

We discussed how to realize a transaction to manipulate objects distributed in multiple computers in a

Table 1. Processing time P.

[s]

	1 (Carry)	2 (Request and send)	3 (Independently transfer)	4 (Client-server model)
Initialization	0.012 (0.015 %)	0.012 (0.016 %)	0.012 (0.016 %)	0.157 (2.879 %)
Moving	1.962 (2.459 %)	1.085 (1.481 %)	0.892 (1.182 %)	0
MA loading	0.029 (0.036 %)	0.032 (0.043 %)	0.083 (0.22)	0.037 (0.678 %)
OBA loading	0.019 (0.023 %)	0.018 (0.025 %)	0.174 (0.023 %)	0
Manipulation	77.594 (97.245 %)	72.029 (98.337 %)	74.18 (98.319 %)	5.235 (96.02 %)
2PC	0.176 (0.22 %)	0.071 (0.097 %)	0.107 (0.422 %)	0.023 (0.422 %)
Total time	79.792	73.247	75.448	5.452

Table 2. Processing time Q.

[s]

	1 (Carry)	2 (Request and send)	3 (Independently transfer)	4 (Client-server model)
Initialization	0.012 (0.01 %)	0.013 (0.015 %)	0.012 (0.0127 %)	0.156 (2.6536 %)
Moving	2.235 (1.91 %)	1.093 (1.271 %)	0.971 (1.003 %)	0
MA loading	0.03 (0.026 %)	0.038 (0.044 %)	0.035 (0.037 %)	0.041 (0.693 %)
OBA loading	0.017 (0.015 %)	0.017 (0.02 %)	0.018 (0.019 %)	0
Manipulation	114.6 (97.95 %)	84.72 (98.54 %)	92.825 (98.788 %)	5.691 (96.18 %)
2PC	0.11 (0.094 %)	0.09 (0.105 %)	0.102 (0.109 %)	0.029 (0.49 %)
Total time	116.99	85.973	93.963	5.917

mobile agent. A transactional agent is a mobile agent which autonomously decides on which computer to visit, moves to a computer, and then locally manipulates objects. A transactional agent has its own commitment condition. We discussed how to implement transactional agents in Aglets. We evaluated the transactional agent model in terms of processing time compared with the client-server model. The client-server model implies shorter responce time than the transactional agent model because the manipulation time of the transactional agent model is too long. We are now discussing how to reduce the manipulation time of the transactional agent manipulating objects at shorter time.

Acknowledgment

This research is partially supported by Research Institute for Science [Q05J-04] and Technology and Frontier Research and Development Center [16-J-6], Tokyo Denki University.

References

[1] American National Standards Institute. *The Database Language SQL*, 1986.

[2] J. Gray and A. Reuter. *Transaction Processing : Concepts and Techniques*. Morgan Kaufmann, 1993.

[3] IBM Corporation. *Aglets Software Development Kit Home*. http://www.trl.ibm.com/aglets/.

[4] D. E. Knuth. *The Art of Computer Programming, Vol. 2*. Auerbach Publications, 1998.

[5] T. Komiya, T. Enokido, and M. Takizawa. Mobile agent model for transaction processing on distributed objects, 2003.

[6] N. A. Lynch, M. Merritt, A. F. W. Weihl, and R. R. Yager. *Atomic Transactions*. Morgan Kaufmann, 1994.

[7] M. Shiraishi, T. Enokido, and M. Takizawa. Fault-Tolerant Mobile Agent in Distributed Objects Systems. In *Proc. of the 9th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003)*, pages 145–151, 2003.

[8] D. Skeen. Nonblocking Commitment Protocols, 1982.

[9] Y. Tanaka, N. Hayashibara, T. Enokido, and M. Takizawa. Design and Implementation of Transactional Agents for Manipulating Distibuted Objects. In *Proc. of the 19th Advanced Information Networking and Applications*, pages 368–373, 2005.