社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2006－DPS－127　(10)
2006／6／6

A Fault-Tolerant Transactional Agent Model on Distributed Object 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*
[2]*Faculty of Business Administration, Rissho University, Japan*
[1]*{youhei, haya, taki}@takilab.k.dendai.ac.jp*, [2]*eno@ris.ac.jp*

Abstract

A *transactional agent* is a mobile agent to manipulate objects with some type of commitment condition. We assume computers may stop by fault while networks are reliable. In the client-server model, servers are fault-tolerant according to traditional replication and checkpointing technologies. However, an application program on a client cannot be performed if a client computer is faulty, e.g. blocking problems in the two-phase commitment protocol. An application program can be performed on another operational computer even if a computer is faulty in the transactional agent (TA) model. There are kinds of faulty computers for a transactional agent; *current*, *destination*, and *sibling* computers where a transactional agent now exists, will move, and has visited, respectively. We discuss how the transactional agent is tolerant of the types of computer faults.

分散オブジェクトシステムにおける耐障害トランザクショナルエージェントモデル

田中 洋平[1]　林原 尚浩[1]　榎戸 智也[2]　滝沢 誠[1]
[1] 東京電機大学大学院 理工学研究科 情報システム工学専攻
[2] 立正大学 経営学部

クライアント・サーバモデルでは，サーバの信頼性は多重化，チェックポイント技法により高められているが，クライアントの故障によりアプリケーションが動作しなくなる問題点がある．本研究では，モバイルエージェントを用いてアプリケーションを高信頼に実行させせることを考える．トランザクショナルエージェントとは，コンピュータ上を移動し、移動先のコンピュータ内のオブジェクトを操作するモバイルエージェントである．従って，コンピュータが障害しても他のコンピュータに移動することにより，アプリケーションを実行させせることができる．本研究では，トランザクショナルエージェントを用いて、耐障害アプリケーションを実現するトランザクショナルエージェントモデルを新たに提案する．

1 Introduction

Various types of objects [7] like databases [9, 11] are distributed on multiple servers. An application program manipulates objects distributed in computers. A *transaction* [3] of the application program is an atomic sequence of methods issued to objects. In the client-server (CS) model [6], a transaction on a client issues access requests like SQL [1] to servers. Servers can be made more reliable and available by using multiple replicas of the servers [15] and taking checkpoints [10]. However, application programs cannot be performed if the clients are faulty. For example, servers might block if a client is faulty in the two-phase commitment protocol [12]. On the other hand, mobile agents [4] are programs which move around computers and then locally manipulate objects in each computer. Here, an application program moves to objects while data objects are transmitted in the CS model. In a mobile agent model, the application program can be performed on another operational computer by escaping from faulty computers. In this paper, we assume computers may stop by fault while networks are reliable. We discuss how to reliably realize an application program manipulating objects in a mobile agent in presence of computer faults. In this paper, a *transactional agent* (TA) is a mobile agent which autonomously moves around computers in networks and locally manipulates objects in each computer. In addition, a transactional agent is specified with one of commitment conditions like atomic and at-least-one conditions. For example, a transactional agent with the majority condition can commit only if objects in more than a half of the objects could be successfully manipulated.

In order to reduce the overhead for transfering classes of a mobile agent, a transactional agent is decomposed into smaller pieces, a *routing* subagent and a collection of *manipulation* subagents. The routing subagent makes a decision on what computer to visit next and then moves to the computer in networks. A manipulation subagent is only a part of an application program to locally manipulate objects in each computer. On arrival of a routing subagent at a computer, a manipulation subagent is loaded to the computer. In order to resolve the *unrecoverable abort*, a manipulation subagent still holds locks on objects even after the routing subagent leaves the computer. In the TA model, there are types of computers which might be faulty, i.e. *destination*, *sibling*, and *current* computers where a transactional agent moves, has passed and a manipulation subagent exsits, and currently exists, respectively. We discuss how a transactinal agent to be tolerant of faults of the types of computers. An application program manipulating objects in multiple computers like database applications are reliably realized in the TA model.

In section 2, we discuss the TA model. In section 3, we discuss how fault-tolerant model of the transactional agent. In section 4, we evaluate the TA model.

2 A Transactional Agent (TA) Model

2.1 Transactional agents

A *mobile agent* is an object-based program composed of classes like Java [13] programs which moves around computers in networks and locally manipulates objects in each of the computers [4] as follows:

1. A class c is stored in a home computer $Home(c)$.
2. If a method on a class c is invoked on a computer D. c is loaded to D from $Home(c)$. If c is cached in D, c in the chache is invoked.
3, A mobile agent A is initiated on a *base* computer $Base(A)$ by loading classes from $Home(A)$.

An application program which manipulates objects is realized in a mobile agent. A transaction of an application can be operational in a mobile agent model as long as the transaction is performed on an operational computer. A *transactional agent* is a mobile agent with the following properties:

1. Autonomously makes a decision on what computer to visit in presence of computer faults and change of service supported by computers.
2. Moves from computers to computers and locally manipulates objects in each computers.
3. Negotiates with other transactional agents with respect to which one takes conflicting objects.
4. Commits only if a commitment condition is satisfied, otherwise aborts.

Target objects are objects to be manipulated by a transactional agent. Target objects are stored in a *target* computer. A program, i.e. classes are transferred to a target computer in the transactional agent (TA) model while the program is fixed on a client in the client-server (CS) model. In order to reduce the communication overhead, a transactional agent A is decomposed into smaller pieces, a *routing* subagent $RA(A)$ and *manipulation* subagents $MA(A, D_1)$, ..., $MA(A, D_n)$ ($n \geq 1$), where D_i stands for a target computer. Each $MA(A, D_i)$ is a part of an application program to locally manipulate objects in D_i. $RA(A)$ moves around computers. On arrival of $RA(A)$ at D_i, classes of $MA(A, D_i)$ are loaded to D_i from $Home(A)$ [Figure 1]. In the TA model, only a part of an application program is loaded to each target computer, which is required to manipulate objects in the computer.

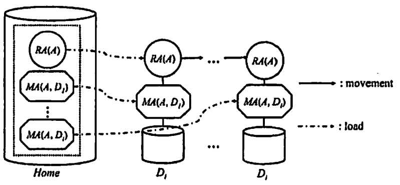

Figure 1. TA model.

2.2 Routing subagents

A transactional agent A is initiated on a base computer D_1 (= $Base(A)$). Here, the routing subagent $RA(A)$ is loaded to D_1 from the home computer $Home(A)$. $RA(A)$ makes a schedule $Sch(A)$ to visit target computers and moves to another computer D_2 from D_1. Thus, $RA(A)$ moves to a computer D_i from D_{i-1}. On arrival at D_i, $RA(A)$ loads classes of a manipulation subagent $MA(A, D_i)$ from $Home(A)$. Objects are locally manipulated in the current computer D_i through $MA(A, D_i)$. Then, $MA(A, D_i)$ may output intermediate objects $Out(A, D_i)$. In turn, $MA(A, D_i)$ may use the intermediate objects $In(A, D_i)$ output by another $MA(A, D_h)$ ($h < i$). An object x flows from a computer D_i to D_j ($D_i \overset{x}{\Rightarrow} D_j$) iff $MA(A, D_i)$ outputs an intermediate object x and $MA(A, D_j)$ uses the object x as an input. Here, D_i and D_j are *source* and *destination* computers of the object x, respectively. $D_i \overset{x}{\Rightarrow} D_j$ means that a transactional agent A has to visit D_i before D_j and x is delivered from D_i to D_j. The authers discuss how to deliver intermediate objects [14]. A *navigation map* $Map(A)$ is a directed graph where each node D_i shows a manipulation subagent $MA(A, D_i)$ [Figure 2]. A directed edge $D_i \rightarrow$

D_j shows that $MA(A, D_j)$ is performed after $MA(A, D_i)$, i.e. $D_i \overset{x}{\Rightarrow} D_j$ for some object x.

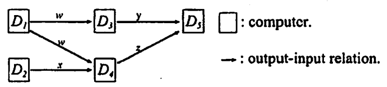

Figure 2. Navigation map $Map(A)$.

A schedule $Sch(A)$ to visit target computers is obtained by using the topological sort [5] of nodes in $Map(A)$. A node without any incoming edge is *initial*. One of initial nodes D_1 and D_2, say D_1 is arbitrarily selected in Figure 2. D_1 is removed from $Map(A)$. D_2 is taken. Finally, $Sch(A)$ is a sequence of computers $\langle D_1, D_2, D_3, D_4, D_5 \rangle$.

2.3 Manipulation subagents

A manipulation subagent $MA(A, D_i)$ is a part of an application program of a transactional agent A, which locally manipulates objects in a computer D_i. Even if a routing subagent $RA(A)$ leaves D_i, $MA(A, D_i)$ still holds objects manipulated in D_i. $MA(A, D_i)$ commits only according to the indication of $RA(A)$. Here, suppose $RA(A)$ is now on a computer D_i after visiting computers D_1, ..., D_{i-1}. Here, D_i is *current* and $MA(A, D_1)$, ..., $MA(A, D_{i-1})$ are *sibling* manipulation subagents. Suppose a routing subagent $RA(A)$ is now on a current computer D_n after visiting D_1, ..., D_{n-1}. If a manipulation subagent $MA(A, D_n)$ finishes manipulating objects in D_n, $RA(A)$ checks if the *commitment* condition $CC(A)$ is satisfied. There are the following types of the commitment conditions:

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

Suppose $MA(A, D_i)$ holds objects and a routing subagent $RA(B)$ of another transactional agent B would like to manipulate objects in a conflicting way. Here, $RA(B)$ negotiates with $MA(A, D_i)$ and it is decided which transactional agent A or B holds the objects based on the commitment conditions [11]. For example, if $CC(A)$ is an at-least-one type, $MA(A, D_i)$ can abort and release the objects.

3 A Fault-Tolerant Model

3.1 Types of faults

A routing subagent $RA(A)$ of a transactional agent A moves in a network. First, $RA(A)$ is initiated on a base computer D_1 (= $Base(A)$). Then, $RA(A)$ moves to a computer D_2 from D_1 and a manipulation subagent $MA(A, D_2)$ locally manipulates objects in D_2. Thus, $RA(A)$ visits computers D_1, D_2, ..., D_i. Then, $RA(A)$ moves to D_{i+1} from D_i. D_{i-1} and D_{i+1} are the *direct predecessor* and *successor* of D_i, respectively. D_{i-h} and D_{i+h} ($h > 0$) are the *predecessor* and *successor* of D_i, respectively. A transactional agent A itself is assumed to correctly behave according to the specification. We assume a computer may stop. There are the following types of faults [Figure 3]:

1. A *destination* computer D_{i+1} on a current computer D_i is faulty.
2. A *sibling* computer D_k ($k < i$) which $RA(A)$ has visited is faulty.
3. A *current* computer D_i is faulty.
4. A home computer $Home(A)$ is faulty.

If $Home(A)$ is faulty, another replica of $Home(A)$ delivers the classes. We discuss the other three types of computer faults.

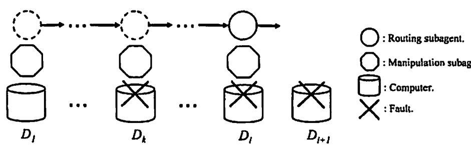

Figure 3. Faults of computers.

3.2 Faults of destination computer

First, suppose a routing subagent $RA(A)$ finds a destination computer D_j from the navigation map $Map(A)$ on the current computer D_i. However, suppose D_j is faulty. $RA(A)$ first tries to find another operational destination computer from D_i. If not found, $RA(A)$ backs to the direct precedessor D_h or waits on the current computer D_i. Suppose $RA(A)$ desides to wait on D_i. If $RA(A)$ waits for some time units, $RA(A)$ backs to the direct predecessor or aborts if D_j is still faulty. Next, suppose $RA(A)$ backs to the direct predesessor D_h. Here, RA has to find another operational destination computer.

There are following ways for a routing subagent $RA(A)$ on a current computer D_i to find another destination D_k if D_j is detected to be faulty:

1. $RA(A)$ finds an *independent* computer D_k of D_j.
2. $RA(A)$ finds a *replica* computer D_k where $MA(A, D_j)$ can be performed.

Independent and replica computers of D_j are *candidate* ones from a current computer D_i. If a candidate

D_k is found, *RA*(A) moves to D_k. If another destination from D_i is not found, *RA*(A) backs to the direct predecessor D_h. Here, $MA(A,\ D_i)$ is aborted on D_i. Then, a node D_j is marked "faulty" in $Map(A)$.

On backing to the predecessor D_h, *RA*(A) brings information on faulty computers. If *RA*(A) on D_h finds another candidate D_k ($k \neq i$) as discussed here, *RA*(A) moves to D_k. Otherwise, *RA*(A) furthermore backs to the direct predecessor. If D_i is $Base(A)$, the transactional agent A aborts.

3.3 Detection of faulty computer

In the TA model, a routing subagent *RA*(A) leaves manipulation subagents on computers. Hence, sibling computers are chached as shown on Figure 4. Each sibling manipulation subagent $MA(A,\ D_i)$ exchanges control messages with $MA(A,\ D_{i-1})$ and $MA(A,\ D_{i+1})$ as shown in Figure 4. If $MA(A,\ D_i)$ is faulty, neither $MA(A,\ D_{i-1})$ nor $MA(A,\ D_{i+1})$ receives any control message from $MA(A,\ D_i)$. Here, $MA(A,\ D_{i-1})$ and $MA(A,\ D_{i+1})$ find D_i to be faulty by the time-out mechanism.

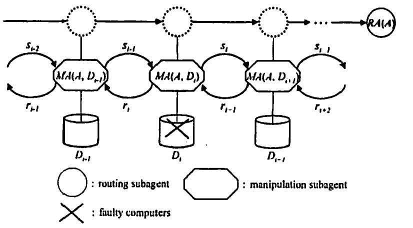

Figure 4. Communication among manipulation subagents.

Suppose a routing subagent *RA*(A) moves to a computer D_i from D_{i-1}. *RA*(A) carries a log information *Log* to D_i, which shows a history of computers D_1, ..., D_{i-1} which *RA*(A) has so far visited. Hence, $MA(A,\ D_i)$ knows what computers are its predecessors D_1, ..., D_{i-1}. On time *RA*(A) leaves a computer D_i for D_{i+1}, *RA*(A) gives information of the destination D_{i+1} to $MA(A,\ D_i)$. Here, a variable $Pred_i$ denotes a sequence of the predecessors $\langle D_1$, ..., $D_{i-1}\rangle$ and $Succ_i$ indicates a sequence of its successors $\langle D_{i+1}\rangle$. Variables $DSucc_i$ and $DPred_i$ show the direct succesor and predecessor of $MA(A,\ D_i)$, respectively. A notation "$\langle a_1, ..., a_n\rangle + b$ " shows $\langle a_1$, ..., a_n, b $\rangle$. a_1 and a_n are the *top* and *last* elements of $\langle a_1$, ..., $a_n\rangle$. D_h and $MA(A,\ D_h)$ in $Succ_i$ $\langle$ D_1, ..., D_h $\rangle$ are referred to as the *last* successor of D_i and $MA(A,\ D_i)$, respectively. Sibling manipulation subagents $MA(A,\ D_1)$, ..., $MA(A,\ D_i)$ communicate with each other to detect faulty computers as follows [Figure 4]:

1. A manipulation subagent $MA(A,\ D_i)$ is created on a computer D_i on arrival of *RA*(A) with the log *Log* from D_{i-1}. Here, $Pred_i := Log$, $DPred_i := D_{i-1}$, and $Succ_i := DSucc_i := \phi$. $MA(A,\ D_i)$ sends a *State* message s_j to $MA(A,\ D_{i-1})$ where $s_j.succ_i = Succ_i$.
2 On receipt of a *State* message s_{i+1} from $MA(A,\ D_{i+1})$, $MA(A,\ D_i)$ manipulates a valiable $Succ_i$ as $Succ_i := s_{i+1}.succ + D_{i+1}$ and send a *State* message s_i to $MA(A,\ D_{i-1})$ where $s_i.succ :=$ $Succ_i$ and a *State-response* message r_i to $MA(A,\ D_{i+1})$ where $r_i.pred := Pred_i$.
3. On reciept of a *State-responce* message r_{i-1} from D_{i-1}, $Pred_i := r_{i-1}.pred + D_{i-1}$ in $MA(A,\ D_i)$. $MA(A,\ D_i)$ sends a *State-responce* message r_i to $MA(A,\ D_{i+1})$ where $r_i.pred := Pred_i$.
4. When *RA*(A) leaves D_i for D_{i+1}, $Log := Prec_i + D_i$ and $Dsucc_i := D_{i+1}$. *RA*(A) carries *Log*.

If $MA(A,\ D_i)$ does not receive any messsage from $MA(A,\ D_{i-1})$ and $MA(A,\ D_{i+1})$ for some time units, $MA(A,\ D_i)$ sends a *State-responce* message r_i and *State-message* s_i to $MA(A,\ D_{i-1})$ and $MA(A,\ D_{i+1})$, respectively. After sending some number of *State-response* and *State* messages, if $MA(A,\ D_i)$ does not receive any message, $MA(A,\ D_i)$ perceives $MA(A,\ D_{i-1})$ and $MA(A,\ D_{i+1})$ to be faulty. Thus, $MA(A,\ D_i)$ obtains information $Pred_i$ of the predecessors D_1, ..., D_{i-1} from *RA*(A). $MA(A,\ D_i)$ obtains information on what computers are the successors on receipt of the *State-responce* messages from $MA(A,\ D_{i+1})$. In addition, $MA(A,D_i)$ forwards a *State* message from the direct predecessor to the direct successor. The change of the predecessors is propagated up from the predecessors to the current computer. *RA*(A) is moving, i.e. the successors are increasing. The change of the successors is also propagated down from successors to the base computer.

3.4 Fault of current computer

A routing subagent *RA*(A) is faulty only if the current computer D_i is faulty due to the fault of D_i. Suppose that *RA*(A) comes from D_{i-1} to D_i. Suppose the direct prodecessor $MA(A,\ D_{i-1})$ detects D_i to be faulty, i.e. *RA*(A) is faulty on D_i. Here, $MA(A,\ D_{i-1})$ recreates a new incarnation of *RA*(A). The new incarnation tries to take another destination D_k in the navigation map $Map(A)$ as discussed before. Here, $MA(A,\ D_{i-1})$ sends a *State* message s_{i-1} to its direct predecessor $MA(A,\ D_{i-2})$, where $s_{i-1}.succ =$ $\langle D_{i-1}\rangle$. If another destination D_k is found, *RA*(A)

moves to D_k. If not found, $RA(A)$ further backs to $MA(A, D_{i-2})$ from D_{i-1} and $MA(A, D_{i-1})$ aborts.

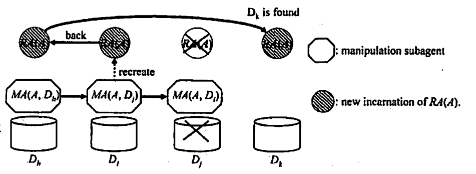

Figure 5. Reincarnation of routing subagent.

3.5 Fault of manipulation subagents

A sibling manipulation subagent $MA(A, D_i)$ may be faulty. $MA(A, D_{i-1})$ and $MA(A, D_{i+1})$ detect the fault of $MA(A, D_i)$ by the time-out mechanism as discussed before. If the commitment condition $CC(A)$ is not an atomic type, $RA(A)$ can continue the computation even if $MA(A, D_i)$ is faulty. $MA(A, D_{i+1})$ knows every predecessor $MA(A, D_h)(h < i)$ while $MA(A, D_{i-1})$ knows $MA(A, D_i)$ but may not know every successor $MA(A, D_h)$ $(h > i)$. Hence, $MA(A, D_{i+1})$ sends a *State* message s_{i+1} to $MA(A, D_{i-1})$. Here, $MA(A, D_{i+1})$ and $MA(A, D_{i-1})$ are now neghbouring.

Secondly, the predecessor $MA(A, D_{i-1})$ of the faulty $MA(A, D_i)$ creates a new incarnation of $RA(A)$ on D_{i-1}. The new incarnation finds another operational destination than the faulty computer D_i as discussed here. Here, $MA(A, D_{i-1})$ sends a *State* message s_{i-1} where $s_{i-1}.succ = \phi$ to $MA(A, D_{i-2})$ to inform the fault of D_i. Thus, the predecessors of $MA(A, D_{i-1})$ eventually know that D_i is faulty and the successors D_{i+1}, D_{i+2}, ... are not sibiling ones. The direct successor $MA(A, D_{i+1})$ also detects $MA(A, D_i)$ to be faulty. $MA(A, D_{i+1})$ sends an *Abort* message to $MA(A, D_{i+2})$ and then aborts. On receipt of *Abort*, $MA(A, D_{i+2})$ forwards *Abort* to $MA(A, D_{i+3})$ and then aborts. Eventually, *Abort* is delivered to the current manipulation subagent and catches up with the old incarnation of $RA(A)$. On receipt of *Abort*, the old incarnation aborts.

If each $MA(A, D_i)$ forwards an *Abort* message to only its direct successor $MA(A, D_{i+1})$, it takes time to inform the old incarnation $RA(A)$ of *Abort*. We take the following ways to reduce the time [Figure 6]:

1. On receipt of *Abort* from $MA(A, D_{i-1})$, $MA(A, D_i)$ finds the last successor $MA(A, D_h)$ in the log $Succ_i = \langle D_{i+1}, D_{i+2}, ..., D_h \rangle$.
2. $MA(A, D_i)$ forwards *Abort* to not only $MA(A, D_{i+1})$ but also the last successor $MA(A, D_h)$. Then, $MA(A, D_i)$ aborts.
3. On receipt of *Abort* from a predecessor $MA(A, D_j)$ $(j < i - 1)$, $MA(A, D_i)$ forward *Abort* to both $MA(A, D_{i+1})$ and $MA(A, D_{i-1})$. Then, $MA(A, D_i)$ aborts. If D_i is current, the incarnation of $RA(A)$ on D_i aborts.

A manipulation subagent $MA(A, D_i)$ delivers an *Abort* message directly to the last successor D_h in $Succ_i = \langle D_{i+1}, ..., D_{h-1}, D_h \rangle$ by skipping the other successors D_{i+1}, ..., D_{h-1}. $MA(A, D_h)$ also forwards an *Abort* message to not only $MA(A, D_{h-1})$ and $MA(A, D_{h+1})$ but also its last successor. The *Abort* message can be earlier delivered to the old incarnation.

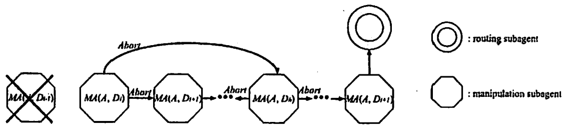

Figure 6. Transmission of Abort.

4 Evaluation

We evaluate the transactional agent (TA) model in terms of how log it takes to deliver an *Abort* message to an old incarnation of a routing subagent after a sibling computer is detected to be faulty. In addition, a routing subagent is moving even after a sibling computer is faulty. We measure has many computers a routing subagent visits after a faulty computer is detected.

We assume that it takes 160 [msec] to perform a transactional agent on each computer, i.e. move a routing subagent, load classes of a manipulation subagent from the base computer, and perform the manipulation subagent which manipulates objects in a database. We also assume it takes 30[msec] to deliver a message from a computer to another computer in a network. In this evaluation, each subagent is implemented as thread in AMD Opteron Processor 248 2GHz and 2 GBytes memory. The movement of a routing subagent is realized as creating a thread. That is, each time a routing subagent moves to another computer, a thread is created and show a manipulation subagent on the computer.

Initially, there are a sequence of fifty manipulation subagents D_0, ..., D_{49}. D_{49} is current and D_0 is a base computer. Every 160 [msec], one subagent D_i is created (i = 50, 51, ...). In the evaluation, each D_i is referred to as *node*. We assume one node out of fifty nodes D_0, ..., D_{49} is faulty. Then, *Abort* messages are exchanged among nodes as discussed in the preceding subsection. In one way, an *Abort* message is sent to only neighboring nodes, i.e. direct successor and precedessor. On the other hand, *Abort* is sent to not only neighboring nodes but also the last successor node. The former way is referred to as *traditional* one. The latter way is a *new* one. We measure how long it takes to deliver *Abort* to the old incarnation of the routing subagent after a faulty node is detected. Figure 7 shows how long it takes to abort the old incarnation for which node D_i is faulty (i = 0, ..., 49). For example, if the 10th node D_{10} is faulty, it takes 1585 [msec] and 857 [msec] to deliver *Abort* in the traditional and new ways, respectively. Until *Abort* catches up with the old incarnation of the old incarnation of the routing subagent, the old incarnation moves in networks, i.e. manipulation subagents are created. Figure 8 shows how many nodes the old incarnation visits after some computer is faulty. In the new way, only one node is created until the old incarnation recieves *Abort*. In the tranditional way, 16 nodes are created for the fault of the node.

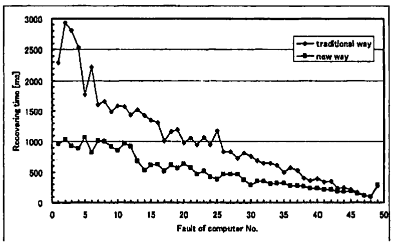

Figure 7. Recovering time.

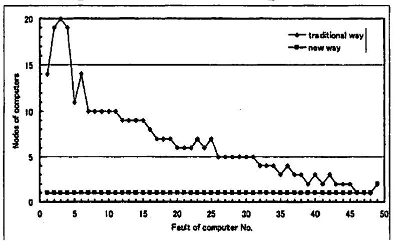

Figure 8. Number of nodes.

5 Concluding Remarks

We discussed how to realize a fault-tolerant application to manipulate objects distributed on multiple computers with a mobile agent in presence of computer faults. There are types of computer faults in the transactional agent, *home*, *destination*, *sibling*, and *current* computers. We discussed how to make a transactional agent tolerant of the types of computer faults through the cooperation of the sibling manipulation subagents. In the traditional client-server model, applications cannot be performed if the clients are faulty. In the transactional agent (TA) model, a transactional agent autonomously finds another destination computer even if computers are faulty. Thus, application programs, especially database application programs can be reliably realized in the transactional agent model. We evaluated how long it takes to abort the old incarnation of the routing subagent.

References

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

[2] L. Gong. *JXTA: A Network Programming Environment*, pages 88–95. IEEE Internet Computing,, 2001.

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

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

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

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

[7] Object Management Group Inc. The Common Object Request Broker : Architecture and Specification. *Rev. 2.1*, 1997.

[8] A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf. *Coordination of Internet Agents*. Springer-Verlag, 2001.

[9] Oracle Corporation. *Oracle8i Concepts Vol. 1 Release 8.1.5*, 1999.

[10] R. S. Pamula and P. K. Srimani. Checkpointing Strategies for Database Systems. In *Proc. of the 15th Annual Conf. on Computer Science, IEEE Computer Society*, pages 88–97, 1987.

[11] 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.

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

[13] Sun Microsystems Inc. *The Source for Java (TM) Technology*. http://java.sun.com/.

[14] Y. Tanaka, N. Hayashibara, T. Enokido, and M. Takizawa. Design and Implementation of Transactional Agents for Manipulating Distibuted Objects. In *Proc. of the IEEE 19th Advanced Information Networking and Applications (AINA2005)*, pages 368–373, 2005.

[15] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding Replication in Databases and Distributed Systems. In *Proc. of IEEE ICDCS-2000*, pages 264–274, 2000.