A Hierarchical Navigation Map for A Fault-Tolerant Mobile Agent Model

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

ABSTRACT

An application program on a faulty computer can be performed on another operational computer by moving the program in the mobile agent model. In this paper, we discuss a transactional agent (TA) model where a reliable and efficient application for manipulating objects in multiple computers is realized in the mobile agent model. Here, only a small part of the application program named routing (R) subagent moves around computers. A routing subagent autonomously finds a destination computer. We discuss a hierarchical navigation (HN) map which computer should be visited. A manipulating (M) subagent programs manipulating objects in a computer are loaded to the computer on arrival of R subagent in order to reduce the communication overhead. 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. The types of faults are detected by neighboring M subagents by communicating with each other. If some of the M subagents are faulty, the R subagent has to be aborted. However, the R subagent is still moving. We discuss how to efficiently deliver the abort message to the moving R subagent. We evaluate the TA model in terms of how long it takes to abort the R subagent if some computer is faulty.

1: INTRODUCTIONS

Various types of objects like databases [11, 13] are distributed on multiple servers in networks. A transaction [3] of an application program is an atomic sequence of methods issued to objects. In the client-server model [6], a transaction is performed on a client and issues access requests like SQL [1] to servers. Servers can be made more reliable and available by using multiple server replicas [14] and taking checkpoints [9] in the servers. However, application programs cannot be performed on clients if the clients are faulty. For example, multiple servers might block if a client is faulty in the two-phase commitment protocol [11]. A process of an application program can be actively, passively, semi-actively, and semi-passively replicated [14]. However, it is not easy to actively and semi-actively perform multiple replicas of an application program on databases since multiple replicas issue update requests to each of the databases [2]. On the other hand, mobile agents [4] are programs which move in networks. Here, an application program on a faulty computer can move to another operational computer. We discuss how to reliably realize an application program manipulating distributed objects in a mobile agent in presence of computer faults. A *transactional agent* (*TA*) is a mobile agent which manipulates objects with some commitment condition.

In order to reduce the communication overhead, a transactional agent is decomposed into a *routing* (*R*) subagent and a collection of *manipulation* (*M*) subagents. The *R* subagent autonomously moves in the network. We introduce *a hierarchical navigation* (*HN*) map in this paper. Here, computers with replicas of an object are collected into a *group* and the precedent relations among components in the group are specified based on the output-input relation [14]. A commitment condition of a transactional agent is specified for each group, e.g. atomic and majority ones. An *M* subagent is only a part of an application program to locally manipulate objects in each computer. On arrival of an *R* subagent, classes of the *M* subagent are loaded to 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 (*M*) subagent exists, and currently exists, respectively. Sibling *M* subagents are linearly chained to show a sequence of computers which the transactional agent has visited. An *M* subagent periodically exchanges messages with its neighboring ones. An *M* subagent is detected to be faulty by the neighboring *M* subagents with the time-out mechanism. We evaluate the TA model in terms of how long it takes to abort the *R* subagent since the *R* subagent is still moving.

In section 2, we discuss the TA model. In section 3, we present the HN map. In section 4, we discuss how a transactional agent can be tolerant of computer faults. In section 5, we evaluate the TA model.

2: A TRANSACTIONAL AGENT MODEL

A class *c* is stored in a home computer *Home*(*c*). If a method of a class *c* is invoked by a mobile agent on a computer *D*, *c* is searched in the local cache. If found, *c* in the cache is invoked. Otherwise, *c* is loaded to *D* from *Home*(*c*). A mobile agent *A* is initiated on a *base* computer *Base*(*A*) by loading classes from *Home*(*A*). A *transactional* agent is a mobile agent which autonomously makes a decision on what computer to visit in presence of computer faults, moves in networks and locally manipulates objects in each computer, negotiates with other transactional agents with respect to which one takes conflicting objects, and commits only if a commitment condition is satisfied, otherwise aborts. *Target* objects are objects to be manipulated by a transactional agent. In order to reduce the communication overhead, a transactional agent A is decomposed into a *routing* (R) subagent $RA(A)$ and *manipulation* (M) 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. Only $RA(A)$ moves around computers while autonomously making a schedule to visit target computers. On arrival of $RA(A)$ at a computer D_i, classes of $MA(A, D_i)$ are loaded to D_i from the home computer $Home(A)$ as shown in Figure 2.

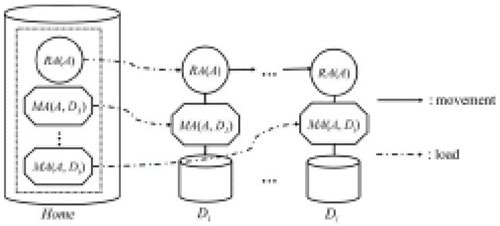

Figure 1. Movement of a transactional agent.

A transactional agent A is initiated on a base computer D_1 (= $Base(A)$) by loading the classes from $Home(A)$. $RA(A)$ makes a schedule $Sch(A)$ to visit target computers. Then, $RA(A)$ moves to another computer D_2 from D_1 according to the schedule $Sch(A)$. Thus, $RA(A)$ moves to a computer D_i from D_{i-1}. Here, D_i is *current*. Then, the M subagent $MA(A, D_i)$ is initiated in D_i. Objects are locally manipulated in the current computer D_i through $MA(A, D_i)$. Then, $MA(A, D_i)$ may output intermediate objects from D_i. In turn, $MA(A, D_i)$ may use the intermediate objects output by another $MA(A, D_h)$ ($h < i$). Even if $RA(A)$ leaves a computer 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 *current* computer D_i after visiting D_1, ..., D_{i-1}. Here, $MA(A, D_1)$, ..., $MA(A, D_{i-1})$ are *sibling* M subagents. Suppose $MA(A, D_i)$ holds objects and $RA(B)$ of another transactional agent B would like to manipulate the objects in a conflicting way. Here, $RA(B)$ negotiates with $MA(A, D_i)$ to decide which one A or B holds the objects based on the commitment conditions [10].

3: A HIERARCHICAL NAVIGATION MAP

Objects are distributed to computers in networks. Suppose an object o_i is replicated in multiple computers D_{i1}, ..., D_{il_i} ($l_i \geq 1$). Let o_{ij} be a replica of an object o_i in D_{ij}. If a transactional agent A derives some values of the object o_i, A can visit any computer D_{ik} with a replica o_{ij}. On the other hand, A has to visit every computer D_{ih} to update the object o_i,. Here, a collection of computers D_{i1}, ..., D_{il_i} is a *group* G_i of o_i. Here, suppose a manipulation (M) subagent $MA(A, D_i)$ derives some object x from a computer D_i and $MA(A, D_j)$ uses x in another D_j [Figure 3]. Here, there is an output-input relation "$D_i \rightarrow D_j$." The output-input relation $\rightarrow$ shows a sequence of computers in which the R subagent $RA(A)$ visits. If $D_i \rightarrow D_j$, $RA(A)$ has to visit D_i prior to D_j. D_i and D_j are *independent* ($D_i \parallel D_j$) iff neither $D_i \rightarrow D_j$ nor $D_j \rightarrow D_i$.

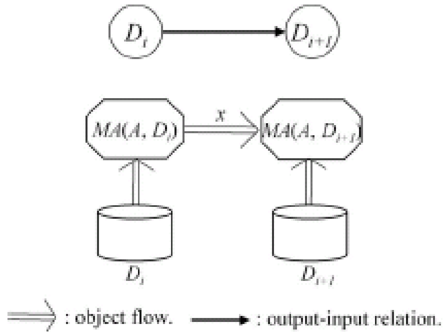

Figure 2. Input-output relation.

A hierarchical navigation (HN) map $HMap(A)$ is composed of *nodes* interconnected with output-input relations for each transactional agent A. A node G_i is composed of nodes. A node shows a computer or a group. A *group* is a collection of nodes. A node showing a computer is *primitive*. Thus, $HMap(A)$ is a group of nodes. Let us consider a group $G_i = \{D_{i1}, ..., D_{il_i}\}$ ($l_i \geq 1$) of an object o_{il_i}. Each D_{ij} is primitive of G_i. Suppose each D_{ij} has a replica o_{ij}. A transactional agent A is specified with the following commitment condition $CC(A, G_i)$ for a group G_i:

1. *Atomic (A) commitment condition*: All the nodes should be successfully visited in a group G_i.
2. *Majority (M) commitment condition*: More than half of the nodes should be successfully visited.
3. *At-least-one (AO) commitment condition*: At least one of the nodes should be successfully visited.
4. (n, r) *commitment condition*: More than $(r/n)l_i$ nodes out of l_i nodes should be successfully visited.

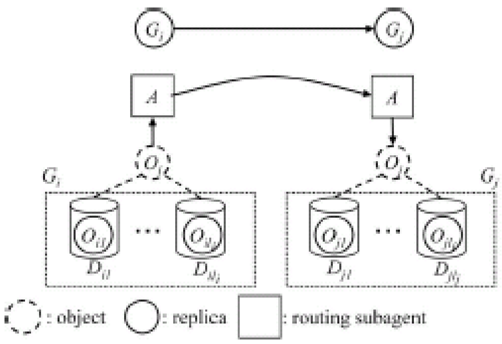

Figure 3. Input-output relation among groups.

If all the objects in G_i have to be successfully manipulated in a transactional agent A, the atomic commitment is taken for G_i, i.e. $CC(A, G_i) = A$. For example, if A just derives values from the object o_i, A can visit one computer in G_i. Here, $CC(A, G_i) = AO$. In another example, an object o_i is decomposed into subagents, i.e. subobjects $o_{i1}, ..., o_{il_i}$. Each subobject o_{ij} is a part of o_i. Here, a group G_i is composed of nodes $\{G_{i1}, ..., G_{il_i}\}$ where each G_{ij} is related with a subobject o_{ij} . Then, each o_{ij} is replicated in replicas $o_{ij1}, ..., o_{ijl_{ij}}$, where each replica o_{ijk} is in D_{ijk}. Here, G_i is a collection of primitire nodes $\{G_{ij1}, ..., G_{ijl_{ij}}\}$ where each G_{ijk} shows a computer D_{ijk} with a replica o_{ijk}. If a transactional agent A navigates data in the object o_i, $CC(A, G_i) = A$ and $CC(A, G_{ij}) = AO$.

Let G_k be $\{D_{k1}, ..., D_{kl_h}\}$ for object o_k. If a transactional agent A manipulates the object o_j after o_i, G_i *precedes* G_j ($G_i \rightarrow G_j$) in A as shown on Figure 4. Thus, A decides on in which order A would visit node in each G_i. We discuss how to make a schedule $Sch(A)$ from $HMap(A)$ composed of nodes $G_1, ..., G_l$. A node G_{ij} is *initial* in G_i if there is no node G_{ik} such that $G_{ik} \rightarrow G_{ij}$. Furthermore, each group G_i is furthermore composed of nodes $G_{i1}, ..., G_{il_i}$ ($l_i \geq 1$). Here, ``$<b_1, …, b_m> + a$” = $<b_1, …, b_m, a>$. A schedule is obtained from G_i by the following procedure.

[Schedule(*A*, *Gi*)]

1. $G_i := \varphi$. return (Sch);
2. I_i := subset of initial nodes which have no incoming edge in G_i; $R_i := \varphi$; $Sch := \varphi$;
3. Otherwise, take one initial node G_{ij} in I_{ij};
   $G_i := G_i - \{G_{ij}\}$; $R_j := R_j \cup \{G_{ij}\}$;
   If G_{ij} is an aggregate node, $Sch := Sch$ + **Schdule**(A, G_{ij}), else $Sch := Sch + G_{ij}$;
4. If $CC(A, G_i) = A$, go to 1;
   if $CC(A, G_{ij}) = AO$, return (Sch) if $|R_i| \neq o$,
   if $CC(A, G_{ij}) = M$, return (Sch) if $|R_i| > l_i / 2$,
   if $CC(A, G_{ij}) = (n, r)$, return (Sch) if $|R_i| \geq (r / n)l_i$;
   go to 1.

In Figure 5, a group G is composed of three nodes G_1, G_2, and G_3 where $G_1 \rightarrow G_3$ and $G_2 \rightarrow G_3$. Suppose the atomic commitment is taken for G in a transactional agent A, $CC(A, G) = A$(*atomic*). Here, the initial set I is $\{G_1, G_2\}$. One of the initial nodes G_1 and G_2 is taken, say G_1. Then, G_2 is taken. Here, G_1 *precedes* G_2 ($G_1 \Rightarrow G_2$). Lastly, G_3 is taken. $G_1 \Rightarrow G_2 \Rightarrow G_3$. G_1 includes computers D_1, D_2, and D_3. Here, suppose $CC(A, G_1) = AO$. Suppose $D_{11} \rightarrow D_{12}$ in G_1. Here, $I_1 = \{D_{11}, D_{13}\}$. Suppose D_{11} is taken. Since $D_{11} \rightarrow D_{12}$, D_{12} is next taken. Here, $D_{11} \Rightarrow D_{12}$. Since $CC(A, G_1) = AO$, D_{13} is not taken. Then, D_{12} and D_{13} are taken. Next, D_2 in G_2 is taken. Finally, the nodes in G_3 are taken. Suppose that $CC(A, G_3) = A$. For example, D_{31} is first taken because D_{31} is less loaded than D_{32}. $D_{31} \Rightarrow D_{32}$. Finally, a schedule $D_{11} \Rightarrow D_{12} \Rightarrow D_2 \Rightarrow D_{31} \Rightarrow D_{32}$ is obtained by **Schedule**(A, G).

4: A FAULT-TOLERANT MODEL

4.1: TYPES OF FAULTS

A routing (R) subagent $RA(A)$ of a transactional agent A is initiated on a base computer D_1 (= $Base(A)$). Then,

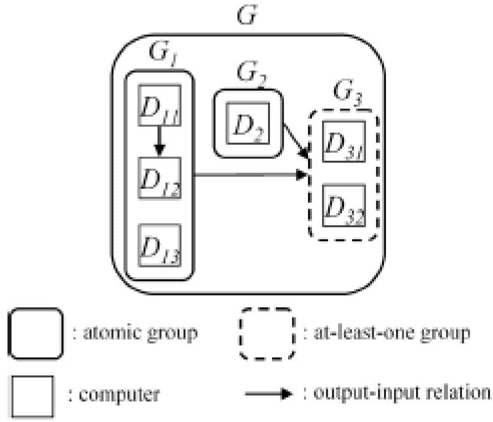

Figure 4. Hierarchical navigation map.

$RA(A)$ moves to D_2 from D_1 and a manipulation (M) subagent $MA(A, D_2)$ locally manipulates objects in D_2. Thus, $RA(A)$ visits computers $D_1, D_2, ..., D_i$. The sequence $D_1, D_2, ..., D_i$ is the *history* of A. Then, $RA(A)$ moves to D_{i+1} from D_i. Here, D_{i-1} and D_{i+1} are the *direct predecessor* and *direct successor* of D_i, respectively. D_{i-h} and D_{i+h} ($h > 0$) are the *predecessor* and *successor* of D_i, respectively. We assume networks are reliable but a computer may stop by fault. Suppose $RA(A)$ is on a current computer D_i. There are the following types of faulty computers in a history $D_1, ..., D_i$:

1. A *destination* computer D_{i+1} 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.

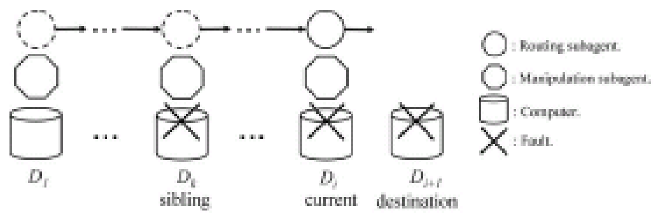

Figure 5. Faults of computers.

4.2: FAULT OF DESTINATION COMPUTER TYPES OF FAULT

First, suppose a routing (R) subagent $RA(A)$ finds a destination computer D_j in $HMap(A)$ on the current D_i. Here, suppose the destination D_j is faulty and $RA(A)$ first tries to find another operational destination from D_i as follows:

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

Independent and replica computers of D_j are *candidates.* If a candidate D_k is found, $RA(A)$ moves to D_k. If another destination from the current D_i is not found, $RA(A)$ backs to the direct predecessor D_h. Here, $MA(A, D_i)$ is aborted. On backing to the predecessor D_h, $RA(A)$ brings information on the faulty computer. If

$RA(A)$ on D_h finds another candidate D_k ($k \neq i$), $RA(A)$ moves to D_k. Otherwise, $RA(A)$ furthermore backs to the direct predecessor.

[Faulty destination computer]

1. $RA(A)$ takes one of the following actions:
   a. Finds another candidate computer D_k.
   b. Waits on the current computer D_i until the faulty destination D_{i+1} is recovered.
2. $RA(A)$ takes the action *a*:
   If a candidate D_k is found, $RA(A)$ moves to D_k. If not found, go to 4.
3. $RA(A)$ takes the action *b*:
   If the destination D_{i+1} is recovered, $RA(A)$ moves to D_{i+1}. If the timer expires, go to 4.
4. If the current computer D_i is D_1, $RA(A)$ aborts. Otherwise, $RA(A)$ backs to the direct predecessor D_{i-1}. Then, $RA(A)$ takes the action *a*.

Each sibling *M* subagent $MA(A, D_i)$ exchanges control messages with the direct predecessor $MA(A, D_{i-1})$ and the direct successor $MA(A, D_{i+1})$ as shown in Figure 6. Here, a pair of $MA(A, D_{i-1})$ and $MA(A, D_{i+1})$ find D_i to be faulty by the time-out mechanism.

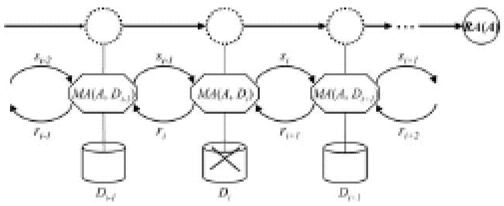

Figure 6. Communication among manipulation subagents.

Suppose an *R* 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 D_i for D_{i+1}, $RA(A)$ gives information of D_{i+1} to $MA(A, D_i)$. Here, a variable $Pred_i$ denotes a sequence of the predecessors $<D_1, \ldots, D_{i-1}>$ and $Succ_i$ shows a sequence of its successors which $MA(A, D_i)$ knows. Variables $DSucc_i$ and $DPred_i$ show the direct successor and predecessor of $MA(A, D_i)$, respectively. $Dsucc_i = D_{i-1}$ and $Dpred_i = D_{i+1}$. a_1 and a_n are *top* and *last* elements of a list $<a_1, \ldots, a_n>$. An element D_h in $Succ_i = <D_{i+1}, D_{i+2}, \ldots, D_h>$ is the *last* successor of D_i. Sibling *M* subagents $MA(A, D_1)$, ..., $MA(A, D_i)$ communicate with each other as follows [Figure 6]:

[Faulty sibling computer]

1. $MA(A, D_i)$ is created on D_i on arrival of $RA(A)$ with the log *Log* from the direct predecessor D_{i-1}. Here, $Pred_i := Log$, $DPred_i := D_{i-1}$, and $Succ_i := DSucc_i := \varphi$. $MA(A, D_i)$ sends a *State* message s_j to the direct predecessor $MA(A, D_{i-1})$ where $s_j.succ_i = Succ_i$.
2. On receipt of a *State* s_{i+1} from $MA(A, D_{i+1})$, $MA(A, D_i)$ manipulates a variable $Succ_i$ as $Succ_i := s_{i+1}.succ + D_{i+1}$ and sends as follows:
   a. a *State* message s_i to $MA(A, D_{i-1})$ where $s_i.succ := Succ_i$.
   b. *State-response* message r_i to $MA(A, D_{i+1})$ where $r_i.pred := Pred_i$.
3. On receipt of 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 *State-responce* r_i to $MA(A,D_{i+1})$ where $r_i.pred := Pred_i$.
4. When $RA(A)$ leaves D_i for the direct successor D_{i+1}, $Log := Log + D_i$ and $RA(A)$ carries *Log*. Here, $Dsucc_i := D_{i+1}$.

If $MA(A, D_i)$ does not receive any message from the direct predecessor $MA(A, D_{i-1})$ and direct successor $MA(A, D_{i+1})$ for some time units, $MA(A, D_i)$ sends *State-responce* r_i and *State* s_i to $MA(A, D_{i-1})$ and $MA(A, D_{i+1})$, respectively. If $MA(A, D_i)$ does not receive any message after sending some number of *State-response* and *State* messages, $MA(A, D_i)$ perceives the neighbors $MA(A, D_{i-1})$ and $MA(A, D_{i+1})$ to be faulty, respectively. 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 configuration on what computers are the successors on receipt of *State-response* from the direct successor $MA(A, D_{i+1})$. In addition, $MA(A, D_i)$ forwards *State* from the direct predecessor to the direct successor.

4.4: FAULT OF CURRENT COMPUTER

A routing (*R*) subagent $RA(A)$ is faulty only if a current computer D_i is faulty. Suppose that $RA(A)$ comes from a computer D_{i-1} to D_i. Suppose the direct predecessor $MA(A, D_{i-1})$ detects D_i to be faulty, i.e. $RA(A)$ is faulty on D_i. Here, the direct predecessor $MA(A, D_{i-1})$ recreates a new incarnation of $RA(A)$. The new incarnation tries to take another destination D_k in $HMap(A)$. Here, $MA(A, D_{i-1})$ sends *State* s_{i-1} to $MA(A, D_{i-2})$, where $s_{i-1}.succ = <D_{i-1}>$. If another destination D_k is found, $RA(A)$ moves to D_k. If not found, $RA(A)$ further backs to $MA(A, D_{i-2})$ and $MA(A, D_{i-1})$ aborts.

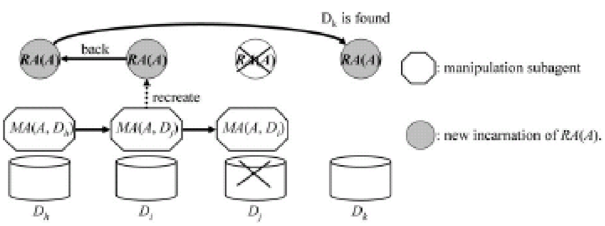

Figure 7. Reincarnation of routing subagent.

[Faulty current computer]

1. The direct predecessor $MA(A, D_{i-1})$ detects the current computer D_i to be faulty by timeout.
2. $MA(A, D_{i-1})$ recreates a new incarnation $RA(A)$.
3. $RA(A)$ on D_{i-1} takes the action of the faulty destination.

4.5: FAULT OF MANIPULATION SUBAGENT

A sibling *M* subagent $MA(A, D_i)$ which $RA(A)$ has visited may be faulty. The direct predecessor $MA(A, D_{i-1})$ and the direct successor $MA(A, D_{i+1})$ detect $MA(A, D_i)$ to be faulty by the time-out mechanism as discussed before. If the commitment condition 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 of the faulty $MA(A, D_i)$, i.e. $MA(A, D_1)$, ..., $MA(A, D_{i-1})$ while $MA(A, D_{i-1})$ knows that $MA(A, D_i)$ is its direct successor but may not know every successor $MA(A, D_h)$ ($h > i$) of $MA(A, D_i)$. Hence, $MA(A, D_{i+1})$ sends *State* s_{i+1} to $MA(A, D_{i-1})$. Here, $MA(A, D_{i+1})$ and $MA(A, D_{i-1})$ are now neighbors as shown in Figure 8.

Secondly, the direct 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 as discussed before. Here, $MA(A, D_{i-1})$ sends *State* s_{i-1} where $s_{i-1}.succ = \varphi$ to $MA(A, D_{i-2})$ to inform the fault of D_i. Thus, every predecessor $MA(A, D_h)$ ($h < i-1$) of $MA(A, D_{i-1})$ eventually knows that D_i is faulty and the successors D_{i+1}, D_{i+2}, ... of the faulty $MA(A, D_i)$ are not sibling ones. The direct successor $MA(A, D_{i+1})$ also detects $MA(A, D_i)$ to be faulty. We consider the following three ways to deliver *Abort* to the old incarnation.

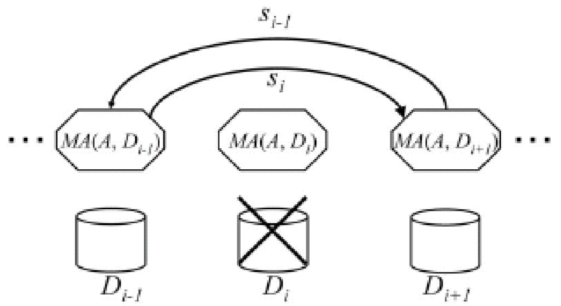

Figure 8. Detection of faulty computer.

[Linear chain (LC) way]

1. $MA(A, D_{i+1})$ sends an *Abort* message to the direct successor $MA(A, D_{i+1})$ and then aborts.
2. On receipt of *Abort* from $MA(A, D_{i-1})$, $MA(A, D_i)$ forwards *Abort* to $MA(A, D_{i+1})$ if D_i is not current. $MA(A, D_i)$ forwards *Abort* to $RA(A)$ if D_i is current.
3. Then, $MA(A, D_i)$ aborts.

[Modified chain (MC) way]

1. On receipt of *Abort* from the direct successor $MA(A, D_{i+1})$, $MA(A, D_i)$ 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 the direct successor $MA(A,D_{i+1})$ but also the last successor $MA(A,D_h)$. $MA(A, D_i)$ aborts.
3. On receipt of *Abort* from a predecessor $MA(A, D_j)$ ($j < i - 1$), $MA(A, D_i)$ forwards *Abort* to $MA(A, D_{i+1})$ and $MA(A, D_{i-1})$. $MA(A, D_i)$ aborts. If D_i is current, the old incarnation of $RA(A)$ aborts.

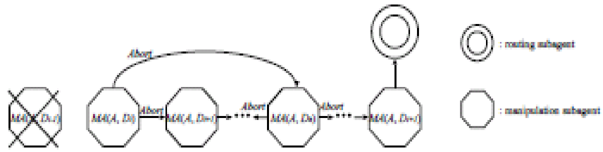

Figure 9. MC way.

[Broadcast (BC) way]

1. $MA(A, D_i)$ detects the direct predecessor $MA(A, D_{i-1})$ to be faulty.
2. $MA(A, D_i)$ sends *Abort* with the successors list $succ_i$ to every successor $MA(A, D_h)$ ($h > i$) in the log $succ_i$. Then, $MA(A, D_i)$ aborts.
3. On receipt of an *Abort* message from a predecessor $MA(A, D_h)$, $MA(A, D_i)$ forwards *Abort* to $RA(A)$ if D_i is current. Otherwise $MA(A, D_i)$ forwards *Abort* to every successor $MA(A, D_h)$ ($h > i$) in the log $succ_i$ - $succ_h$. Then $MA(A, D_i)$ aborts.

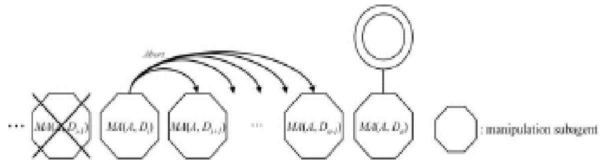

Figure 10. BC way.

5: Evaluation

In the transactional agent (TA) model, a routing (*R*) subagent is moving in a network even after some sibling manipulation (*M*) subagent is faulty. An *Abort* message has to be delivered to the old incarnation of the *R* subagent, which is still moving. We evaluate the TA model in terms of how long it takes to deliver an *Abort* message to the old incarnation if some sibling computer is faulty. We assume that it takes 16 [msec] to perform a transactional agent on each computer, i.e. an *R* subagent moves to the current computer from another computer, classes of an *M* subagent are loaded to the current computer from the base computer, and the *M* subagent is performed by manipulating objects in the current computer. We assume it takes 3 [msec] to deliver a message from a computer to another. In this evaluation, each subagent is realized as thread. The movement of an *R* subagent is simulated as creating a thread.

Initially, there are a sequence of 100 *nodes* M_0, ..., M_{99} where each M_i shows $MA(A, D_i)$. M_{99} is current and M_0 is a base computer. Every 16 [msec], one *M* subagent M_i is created (i = 100, 101, ...). We assume one node M_i out of 100 nodes M_0, ..., M_{99} is faulty. In the LC way, *Abort* is sent from a node to only neighboring nodes. *Abort* is forwarded to the old incarnation. In the MC way, *Abort* is sent to not only neighboring nodes but also the last node M_h. Then, the node M_h forwards *Abort* to its neighboring node and last node if the node is not current. In the BC way, *Abort* is sent to every successor which M_i knows. The last node is sent to every successor.

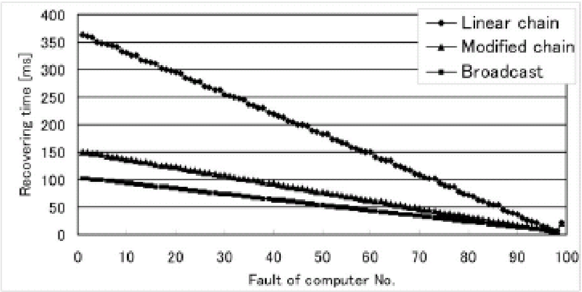

Figure 11. Delivery time of Abort message.

Figure 11 shows the delivering time of the *Abort* message if M_i is faulty (i = 0, ..., 99). For example, if M_{10} is faulty, it takes 330 [msec], 138 [msec], and 94 [msec] to deliver *Abort* in the LC, MC, and BC ways, respectively. Figure 12 shows how many computers the old incarnation of the *R* subagent visits after some sibling computer is faulty until *Abort* is delivered to the old incarnation. In the LC and BC ways, only one node is created until the old incarnation receives *Abort*.

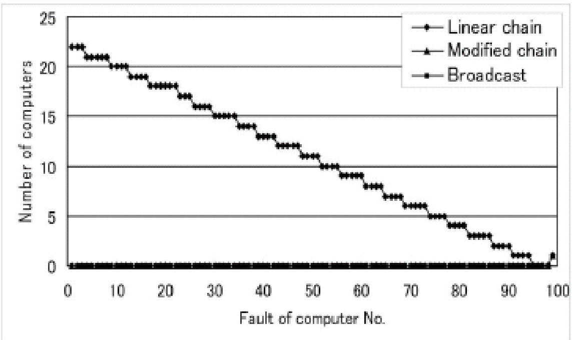

Figure 12. Number of computers.

In the LC way, the old incarnation of the *R* subagent visits 13 computers after M_{40} is detected to be faulty. Figure 13 shows how many *Abort* messages are transmitted among nodes until the old incarnation is aborted. For example, 20 messages are transmitted in the BC way, 22 in the MC way, and 28 in the LC way if the 80th node M_{80} is faulty. Following the figures, the BC way implies the best performance.

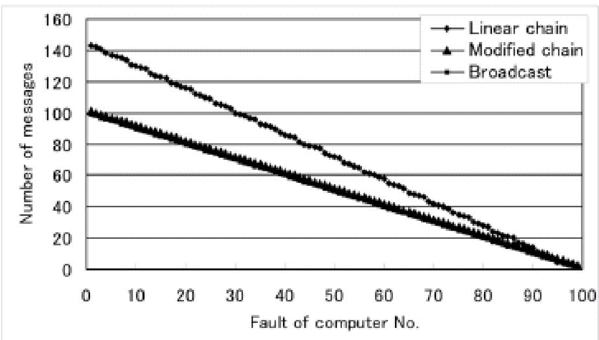

Figure 13. Number of messages.

6: Concluding Remarks

We discussed how to realize a fault-tolerant application to manipulate distributed objects with a mobile agent in presence of computer faults. A transactional agent (TA) is a mobile agent which manipulates objects with some commitment condition. There are types of computer faults, *home*, *destination*, *sibling*, and *current* computers. We discussed how to make the 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 TA model, a transactional agent autonomously finds another destination computer even if computers are faulty. Thus, application programs can be reliably realized in the TA model. We evaluated how long it takes to recover from faulty computers.

REFERENCES

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

[2] T. Enokido, K. Hori, M. Takizawa, and M. Raynal. Quorum-Based Multi-Invocation Model for Replicated Objects. *In Concurrent Engineering: Research and Application Journal (CERA)*, 12:185–194, 2004.

[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] T. Komiya, T. Enokido, and M. Takizawa. Mobile agent model for transaction processing on distributed objects. In *Information Sciences*, volume 154, pages 23–38, 2003.

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

[9] 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.

[10] 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.

[11] D. Skeen. Nonblocking Commitment Protocols. In Proc. of ACM SIGMOD, pages 133–147, 1982.

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

[13] 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.

[14] 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.