Transactions on Distributed Mobile Objects *

Takeaki Yoshida and Makoto Takizawa †
Tokyo Denki University ‡
e-mail {take, taki}@takilab.k.dendai.ac.jp

According to the advances of communication technologies, kinds of mobile stations like personal handy systems and intelligent robots are available. Objects are distributed in not only fixed stations but also mobile ones. Transactions manipulate multiple, possibly replicated objects in mobile and fixed stations. While the objects are moving from one location to others in the system, the quality of service (QoS) supported by the objects are changed. That is, the performance aspects like the bandwidth and latency are changed while the stations are moving in the system. The connection is tentatively closed by the mobile wireless station in order to reduce the power consumption while the operations issued by the mobile station are being computed, i.e. disconnected operations. In this paper, we discuss how to manage the transactions manipulating multiple, possibly mobile and replicated objects.

1 Introduction

According to the advances of communication and computer technologies, kinds of mobile wireless stations like *personal handy systems* are available. Communication among robots in industrial factories and automobile cars are also kinds of mobile communications. The distributed systems are composed of mobile and fixed stations interconnected by the communication networks. The fixed stations are connected to the communication network at the fixed location, i.e. service access point (SAP). The mobile stations in a *cell* communicate with the *mobile support station* (*MSS*) in the cell by using the wireless communication. The mobile support station maintains the *connection* between the mobile station and another station. If the mobile station moves to another cell, it can continue to communicate with the station through the mobile support station in the cell. Tanaka [17] and Teraoka [18] discuss how to support the connection with the mobile stations.

The transactions computed in the *mobile* stations issue the operations to the server stations. The mobile stations like the personal handy systems are not equipped with enough capacity of battery to have long-time communication. In order to reduce the power consumption of the mobile stations, the connections among the mobile stations and the other stations are disconnected while the operations issued by the mobile stations are being computed, i.e. *disconnected* operations [12]. One technique to reduce the power consumption of the mobile station is to *cache* data in the other station like servers to the mobile station. Without communicating with the other station, users can manipulate the data cached into the mobile station. Barbara [3] and Huang [9] present how to cache the data in the fixed stations to the mobile stations and how to maintain the mutual consistency among the caches and the fixed stations. JING [11] discusses the locking scheme based on the optimistic two-phase locking [4] on the replicas and a way to reduce the communication overhead to release the locks.

In this paper, the distributed system is assumed to be composed of objects distributed in multiple stations. Each object is composed of data and operations for manipulating the data. Users or objects send operations to the objects. On receipt of the operations, the objects start to compute the operations. On completion of the operations, the objects send back the responses. The operations may issue operations to other objects. The computation of each operation on an object is viewed to be *atomic*, i.e. the operation is completely computed or nothing. The atomic unit of computation is a *transaction* [2, 7]. The computation of an operation in the transaction is also atomic. Thus, the transaction is *nested* [16, 19].

The objects may be replicated into multiple replicas which are allocated into multiple stations in order to increase the reliability, availability, and performance. They are *replicated* objects. In this paper, we assume that the object is fully replicated, i.e. the replicas have the same data and operations as the object.

Objects are stored in the mobile or fixed stations. According to the movement of the mobile stations, the objects in the mobile stations are viewed to move from one location to different location. There are kinds of objects, i.e. *mobile* and *fixed* ones. *Mobile* objects are objects which can move from one location to others in the system. The objects moved from one station to another are also mobile stations. Fixed objects are in the fixed stations. Each object is considered to support the *quality of service* (QoS) like *response time*. On the other hand, the response time of the object may get longer due to the increased latency. Thus, according to the movement of the object o,

*分散移動型オブジェクト上のトランザクション
†吉田 丈成 滝沢 誠
‡東京電機大学

the QoS of o is *changed*. The movement of o is modeled as *change* of the QoS, e.g. bandwidth, latency, response time, disconnection, supported by o in this paper. Problem is how to support users with the service required by the users under situations where the objects are moving in the system. In this paper, we would like to discuss how to manage transactions which manipulate mobile and replicated objects distributed in multiple stations.

In section 2, we present the system model. In section 3, we discuss how to compute disconnect operations in the mobile objects. In section 4, we present how to compute operation on mobile objects. In section 5, we discuss how to maintain the mutual consistency among the replicas.

2 System Model

The distributed system is composed of multiple stations interconected by communication networks [Figure 1]. There are two kinds of stations, i.e. *fixed* and *mobile* ones. The fixed stations are connected at the fixed service access point (SAP) of the network. The mobile stations in a cell communicate with the mobile support station by using the wireless channel. If the mobile station moves to another cell, it communicates with the mobile support station in the cell. By the current network technologies [17, 18], the connection among the stations can be maintained even if the locations of the stations are changed in the network.

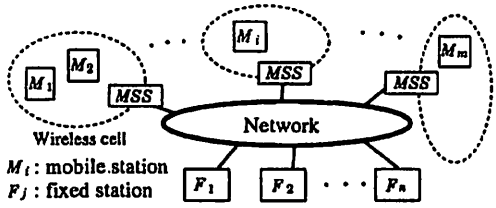

Figure 1: System model

A unit of resource in the system is referred to as *object*, which is composed of data and operations for manipulating the data. Each object o can be manipulated only by the operations supported by o.

There are two kinds of objects, i.e. *class* and *instance*. The class includes the scheme of the data and the operations for manipulating the data. The instance is composed of the data instance of the scheme and the operations inherited from the class.

The objects are distributed into stations in the system. Some objects may be replicated into multiple *replicas* which are in different stations. The object is referred to as *replicated* if there are multiple replicas on the system. Here, suppose that an object o is replicated into multiple replicas $o^1, \ldots, o^m$ $(m \geq 2)$ where each o^i is in a station s_i $(i = 1, \ldots, m)$. If the replicas have the same data and operations as o, o is referred to as *fully replicated* to $o^1, \ldots, o^m$. If not, they are *partially replicated*. If o is the class, s_i has all operations supported by o if fully replicated. s_i has some operations of o if partially replicated. If o is the instance, each s_i has the data instance and the operations. If o is partially replicated, s_i has a part of the data of o. If an object o is in a mobile station, the location of o is changed, i.e. o is moved to a different location according to the movement of the station.

We would like to think abort how the movement of the object o is viewed. For example, the response time to manipulate o may be increased due to the increased latency to o while being able to communicate with o. Thus, the movement of o is modeled to be the change of the quality of service (QoS) supported by o.

[Definition] An object o is *mobile* iff the *QoS* supported by o is time-variant. □

The computation of an operation *op* in an object o may invoke operations in other objects. The computation of *op* is considered to be *atomic*. That is, all the operations invoked by *op* complete successfully or none of them. If some operation invoked by *op* fails, all the operations invoked by *op* have to be aborted. The computation of each operation invoked by *op* is also atomic. Hence, the computation of the operation is considered to be a *nested* transaction [16, 19].

3 Operations on Disconnected Objects

We would like to discuss how to compute operations on mobile and replicated objects.

3.1 Disconnected operations

Suppose that there are three objects o_i, o_{ij}, and o_{ijk} with the data d_i, d_{ij}, and d_{ijk}, respectively. Suppose that an operation op_i in o_i invokes an operation op_{ij} in o_{ij} and op_{ij} further invokes op_{ijk} in o_{ijk} as shown in Figure 2. op_{ij} manipulates d_{ij} in o_{ij} and op_{ijk} manipulates d_{ijk} in o_{ijk}. Since the mobile station like the personal

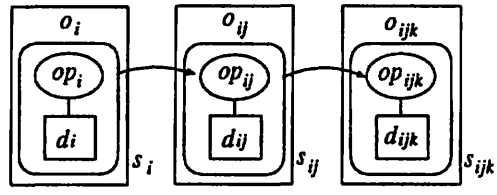

Figure 2: Invocation

handy system is not equipped with such powerful battery that it can have long-time communication, the mobile station often has to close the connection with other stations to reduce the power consumption. The mobile station also may not be disconnected due to the jamming and noise. Thus, the operations may be disconnected while the operations are being computed. The situation is referred to as *disconnected* operation [11].

A *disconnected* object is one which has no connection with the other objects. Operations in the disconnected objects which are invoked by other objects are disconnected one. Object which are not disconnected are *connected*.

There are ways to continue the distributed computation on mobile and fixed objects in the presense of the disconnected objects:

(1) migration of objects, and
(2) replication of objects.

In the migration way, the operations and data of the disconnected object are transferred to another station. On behalf of the disconnected operations, the operations migrated are continued to be computed. The caching of the data is a kind of migration where only data in the object is copied to other station. In the replication way, the object is replicated into multiple replicas. If an object o_{ij} used by an object o_i is disconnected, o_i manipulates a replica of o_{ij} on behalf of o_{ij}.

3.2 Migration of objects

First, we would like to discuss how to migrate objects from one station to others. Here, suppose that o_{ij} is to be disconnected due to the close of the connections in Figure 2. There are two ways for migrating the object:

(1) to migrate the disconnected object o_{ij} in s_{ij} to another station, and
(2) to migrate the connected object o_{ijk} in s_{ijk} to the disconnected station s_{ij}.

One way is to migrate the disconnected object o_{ij} to another station. For example, op_{ij} and d_{ij} of o_{ij} are migrated to the station s_{ijk} as shown in Figure 3. If s_{ijk} has the class of o_{ij}, only d_{ij} can be migrated to s_{ijk} since $sijk$ has the operations op_{ij} After migrating o_{ij} to s_{ijk}, op_i still invokes op_{ij} of o_{ij} in s_{ijk}. If o_{ij} in s_{ij} is reconnected, o_{ij} waits until op_{ij} in s_{ijk} completes. Then, d_{ij} in s_{ijk} is sent to s_{ij}. On receipt of d_{ij}, d_{ij} is restored to the data in o_{ij}. In stead of migrating o_{ij} to s_{ijk}, o_{ij} may be migrated to s_i or other station.

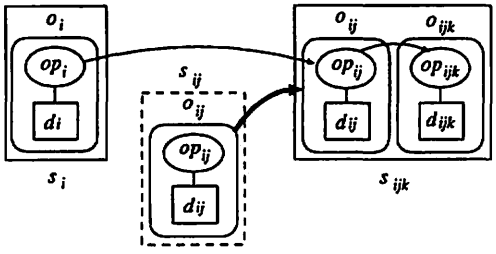

Figure 3: Migration of o_{ij} to s_{ijk}

Another way is to move objects whose operations are invoked to the disconnected station s_{ij}. For example, suppose that o_{ijk} is migrated to s_{ij} as shown in Figure 4. o_{ijk} is migrated to s_{ij} from s_{ijk}. Since o_{ijk} is still connected, o_{ijk} is manipulated by other objects while d_{ijk} is being manipulated in s_{ij}. In the caching method, only d_{ij} is sent to s_{ij} assuming that s_{ij} has the operations for d_{ij}. It is problem how to maintain the mutual consistency of d_{ijk} among s_{ij} and s_{ijk}. The problem is discussed already by many researches [3, 9].

As stated now, if o_{ij} is to be disconnected, there are two ways, i.e. op_{ij} and d_{ij} of o_{ij} in s_{ij} are migrated to another station or op_{ijk} and d_{ijk} invoked by op_{ij} are migrated from s_{ijk}. It depends on which object o_{ij} or o_{ijk} coordinates the distributed computation. For two objects o_{ij} and o_{ijk}, if o_{ij} coordinates the computation on o_{ij} and o_{ijk}, o_{ij} is referred to *superior* to o_{ijk}. An object which is not superior is migrated to a superior object. For example, if o_{ij} is in the mobile handy station and a user interactively manipulates o_{ijk} through o_{ij}, o_{ij} is superior to o_{ijk}, i.e. o_{ijk} is migrated into s_{ij}. If neither o_{ij} nor o_{ijk} are superior, o_{ij} and o_{ijk} are referred to as *equivalent*.

Suppose that o_{ij} and o_{ijk} are equivalent. The following migration strategy is adopted to reduce the communication overhead:

[Migration]

(1) If either o_{ij} or o_{ijk} are updated, the object whose state is not changed is moved to the other.
(2) If a volume of operation and data sent to s_{ijk} is smaller than o_{ij}, o_{ijk} is migrated to s_{ij}. Otherwise, o_{ij} is migrated to s_{ijk}.□

Suppose that an object o_{ij} is changed by the operation op_{ij}. If the object o_{ij} is migrated to another station s_{ijk}, o_{ij} in s_{ij} has to be synchronized with one migrated in s_{ijk} when o_{ij} in s_{ij} is reconnected.

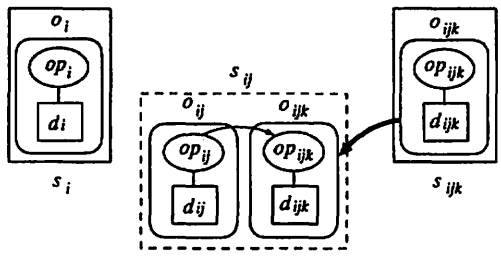

Figure 4: Migration of o_{ijk} to s_{ij}

3.3 Replication of objects

We would like to discuss a case that o_{ij} is replicated into multiple replicas. If one replica o_{ij}^h being manipulated is disconnected, another replica o_{ij}^k is used on behalf of o_{ij}^h. Suppose that o_{ij} is replicated into two replicas o_{ij}^1 and o_{ij}^2 as shown in Figure 5. In this paper, we assume that the objects are fully replicated, i.e. o_{ij}^1 and o_{ij}^2 are the same as o_{ij}. If o_{ij}^1 is to be disconnected, op_i can invoke op_{ij} in the replica o_{ij}^2 and op_{ij} in o_{ij}^2 can invoke op_{ijk} as shown in Figure 5. Here, the current state of o_{ij}^1 has to be sent to o_{ij}^2. On receipt of the states of d_{ij} and op_{ij}, the states are restored to d_{ij} and op_{ij} in o_{ij}^2 and then o_{ij}^2 starts to compute

op_{ij} for the current state received from o_{ij}^1.

Another way is to abort op_i. Due to the abortion of op_i, op_{ij} and op_{ijk} are aborted. Then, op_i is restarted and op_i invokes op_{ij} in o_{ij}^2.

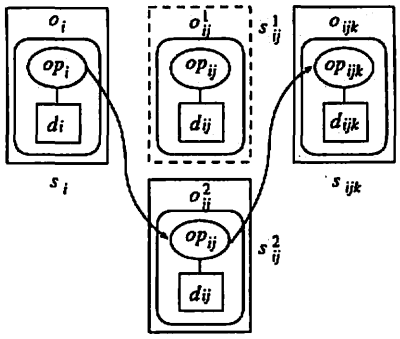

Figure 5: Replication of o_{ij}

4 Operations on Mobile Objects

We would like to discuss how to compute operations on mobile objects.

4.1 Under-qualified operations

According to the movement of the object, the *quality of service* (QoS) of the object is changed. For example, the bandwidth between o_{ij} and o_{ijk} in Figure 2 is changed to be lower if o_{ijk} is moved to a station s'_{ijk} which is connected with the lower bandwidth network[Figure 6]. Suppose that o_{ij} is replicated to two replicas o_{ij}^1 and o_{ij}^2 as shown in Figure 4. If the QoS of o_{ij}^1 for o_i is degraded, o_i can use another replica o_{ij}^2 in stead of o_{ij}^1 if o_{ij}^2 supports the better QoS than o_{ij}^1.

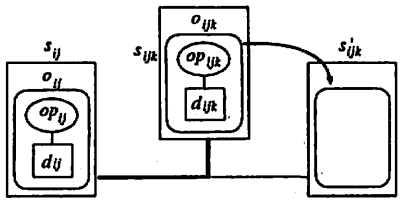

Figure 6: Movement of object

[Definition] Suppose that op_i invokes op_{ij}. op_{ij} is referred to *under-qualified* for op_i if the QoS of op_{ij} is degraded under one which op_i expects to take from o_{ij}. □

Suppose that an operation an object o_{ij} is replicated to $m(\geq 2)$ replicas $o_{ij}^1, \ldots, o_{ij}^m$, where each replica o_{ij}^k is stored in a station $s_{ij}^h (h = 1, \ldots, m)$. o_i has to find a replica o_{ij}^k whose QoS is the best among $o_{ij}^1, \ldots, o_{ij}^m$. o_i selects the best replica o_{ij}^k among the replicas as follows.

[Selection of the best replica]

(1) o_i sends *Rq-QoS* messages to all the replicas $o_{ij}^1, \ldots, o_{ij}^m$.

(2) On receipt of the *Rq-QoS* message from o_i, each replica o_{ij}^k sends back the *Rp-QoS* message with the QoS of o_{ij}^k to $o_i (k = 1, \ldots, m)$.

(3) If o_i receives the *Rp-QoS* messages from the replicas, o_i selects one replica o_{ij}^k with the best QoS among them. □

Then, op_i invokes the operation op_{ij} in o_{ij}^k. This method implies greater communication overhead to broadcast *Rq-QoS* messages to all the replicas. Hence, we adopt the following heuristics to select the replica.

[Selection of the best replica]

(1) If there is a replica o_{ij}^k in the same cell as o_i, the replica o_{ij}^k is selected. If there are multiple replicas in the cell, the replica o_{ij}^k which supports o_i with the best QoS among them is selected.

(2) If there is no replica in the cell, o_i broadcasts the *Rq-QoS* message by the selection algorithm. □

Another way is that there is one coordinator of the replicas, say o_{ij}^1. o_{ij}^1 monitors the change of QoS of each replica o_{ij}^k. o_i first asks o_{ij}^1 to find the best replica for o_i. Then, o_{ij}^1 selects the best one, say o_{ij}^k.

While op_{ij} is computed in o_{ij}^k, the QoS of o_{ij}^k may be changed according to the movement of o_{ij}^k. If op_{ij} could not support the QoS required, i.e. op_{ij} is under-qualified, o_i can select another replica of o_{ij} which supports the better QoS than o_{ij}^k.

[Resolution of the replicas]

(1) If the QoS of o_{ij}^k is being degraded for some time units, o_i finds the best replica o_{ij}^h which is better than o_{ij}^k by the selection procedure.

(2) If o_{ij}^h is detected, o_i requires o_{ij}^k to send the states of d_{ij} and op_{ij} to o_{ij}^h. On receipt of them from o_{ij}^k, o_{ij}^h restore them to the state. o_i invokes op_{ij} in o_{ij}^h. □

4.2 Faulty replicas

One problem on considering the disconnected operations is how to differentiate the disconnected objects and the faulty objects. Suppose that o_{ij} is faulty in Figure 2. If o_{ij} stops by failure, the connection with o_{ijk} is closed. o_{ijk} cannot know whether o_{ij} is faulty or not because the connection is closed and there is no way to communicate with o_{ij}. Here, we make the following assumptions:

[Assumptions]

(1) the network is synchronous, i.e. the propa-

gation delay is bounded, and
(2) the computations in the objects are synchronous [6]. □

The synchronomy assumptions mean that the faulty objects can be detected by the timeout mechanism.

We adopt the following strategy to detect the faulty objects:

[Detection of faulty objects]

(1) The disconnected object o_{ij} sends periodically *Alive* massage to o_i and o_{ijk}.
(2) After the disconnection, if o_{ijk} or o_i does not receive any message from o_{ij} for some predetermined time units, o_{ijk} considers that o_{ij} is faulty. □

The operational objects have to send *Alive* messages to inform other objects of their being operational. The *Alive* message is sent by using the connections less communication.

4.3 Computation of QoS

Operations in objects are nested. The QoS of an operation op_i of an object o_i depends on not only the computation of actions in o but also QoSs of operations invoked by op_i. Let $QoS(op_i)$ denote the QoS of op_i. Suppose that op_i invokes operations $op_{i1}, \ldots, op_{im}$ of objects $o_{i1}, \ldots, o_{im}$, respectively. $QoS(op_i)$ is computed as follows:

$$QoS(op_i) = f_i(QoS(op_{i1}), \ldots, QoS(op_{im}), qos(op_i)).$$

Here, $qos(op_i)$ denotes the QoS supported by the actions of op_i on o_i. f_i is a function which gives the QoS of op_i from the QoSs of $op_{i1}, \ldots, op_{im}$. There are kinds of QoS. The computation time of op_i is obtained by adding the computation times of $op_{i1}, \ldots, op_{im}$ and op_i, i.e. f_i is "+" op_i is computed sequentially. If $op_{i1}, \ldots, op_{im}$ in op_i are computed in parallel, the QoS is obtained by taking the minimum one of $op_{i1}, \ldots, op_{im}$.

In order to compute the QoS of op_i, o_i asks o_{ij} to send $QoS(op_{ij})$ periodically or each time op_i is invoked.

5 Operation-Type Based Optimistic Concurrency Control

We would like to discuss how to maintain the mutual consistency among the replicas.

5.1 Lock modes

Objects may be replicated. Here, for an object o_i, let $r(o_i)$ be a collection of replicas of o_i, i.e. $r(o_i) = \{o_i^1, \ldots, o_i^{l_i}\} (l_i \geq 2)$, where each o_i^j is a replica of o_i $(j = 1, \ldots, l_i)$. Each replica o_i^j is stored in a station s_{ij} $(j = 1, \ldots, l_i)$. We would like to discuss how to maintain the mutual consistency among the replicas.

Before an operation op_i is applied to o_i, o_i is locked. If o_i is locked, op_i is computed in o_i. If not, op_i waits. Two operations op_i and op_j are referred to as *compatible* iff the states obtained by computing op_i and op_j in any order are the same. In order to increase the concurrency, kinds of lock modes are introduced, e.g. *read* and *write* modes. The objects support more kinds of operations than *read* and *write* of the file objects. An operation op_i of o_i is assigned a lock mode $m(op_i)$. The compatibility relation among the lock modes is defined as follows [13].

[Definition] For every pair of lock modes m_1 and m_2 supported by an object o_i, m_1 is *compatible with* m_2 iff an operation of m_1 is compatible with operations of m_2. □

If m_1 is not compatible with m_2, m_1 *conflicts* with m_2. That is, op_i of m_1 has to wait until the operations of m_2 complete in o_i. For example, a *Bank* object supports operations *Deposit* and *Withdrawal*. The modes of *Deposit* and *Withdrawal* are compatible.

Objects support various kinds of abstract operations like *Deposit* and *Withdrawal* while the database systems support only *read* and *write* operations, i.e. *read* and *write* modes. Hence, various kinds of lock modes are supported by the objects. The precedence relation among the lock modes is defined in [13]. Here, let M_0 be a set of lock modes supported by an object o. For each mode m in M_0, let $c(m)(\subseteq M_0)$ be a set of modes which m is compatible with.

[Definition] For every pair of modes m_1 and m_2 of an object, $m_1 \prec m_2$ (m_1 is *not more restricted* than m_2) iff $c(m_1) \supseteq c(m_2)$. □

Here, $m_1 \preceq m_2$ means that m_2 is stronger than m_2. If neither $m_1 \prec m_2$ nor $m_2 \prec m_1$, $m_1 \parallel m_2$. Here, $m_1 \preceq m_2$ or $m_1 \parallel m_2$. Here, $read \prec write$ because $c(read) = \{read\} \supseteq c(write) = \phi$.

5.2 optimistic locking

The typical scheme to maintain the mutual consistency among multiple replicas is the *read-one* and *write-all* principle. That is, the read operation is issued to one replica while the write operation is issued to all the replicas. If one replica is locked in a read mode, the read operation can be computed in the replica. On the other hand, if all the replicas could be locked in the *write* mode, the write operation is computed in all the replicas. In order to reduce the communication overhead, the optimistic approach [14] is adopted. Carey [4] discusses the optimistic two-phase locking (O2PL) protocol, Jing [11] extends the O2PL as to reduce the communication overhead by avoiding the release of the locks. In the O2PL, one replica is locked by *read* but the replicas are not locked by *write*. When the transaction commit the replicas to be updated are locked by *write*. More abstract types of operations are considered in the objects than the read and write operations. The read-one and write-all principle can be extended by taking into account the various kind of lock modes.

The second point on the operations is concerned with whether the operations change the state of the object or not. For example, *Deposit* and *Withdrawal* change the state of *Bank* while they are compatible. If an operation *op* does not change the state of o, *op* can be computed in only one replica of o. Otherwise, *op* has to be computed in all the replicas to keep the mutual consistency among the replicas.

The third point is concerned with whether the operations invoke another operation or not. Suppose that an operation op_i in o_i invokes op_{ij} in o_{ij} and o_i is replicated to replicas o_i^1 and o_i^2. If op_i is computed in o_i^1 and o_i^2, op_{ij} is invoked twice, i.e. by op_i^1 in o_i^1 and op_i^2 in o_i^2. It implies the inconsistency. Hence, if an operation in an object invokes another operation, the operation can be computed in only one replica and the state obtained by computing the operation in the replica has to be transferred to the other replicas.

[Optimistic locking] For every operation op of a mode m_1 supported by an object o,

(1) if $m_1 \prec m_2$ for every mode m_2 of o, one replica o^k in $r(o)$ is locked, and op is computed in o^k if op does not change the state of o, otherwise op is computed in all the replicas,

(2) otherwise, all the replicas in $r(o)$ are locked, and op is computed in all the replicas.□

Problem is the communication overhead since all the replicas have to be locked by the operations whose modes are not minimal.

5.3 Mode-based locking

We adopt the optimistic approach to reduce the communication overhead. We make the following assumption. [assumption] The less restricted operations are, the more often they are used.□

Each operation op locks some number of replicas $r(o)$ rather than locking all the replicas. The more restricted the operation mode is, the more replicas are locked. For each operation op_i in o_i, a number $q(op_i)$ is given as follows.

- $q(op_i) < q(op_j)$ if $m(op_i) \prec m(op_j)$.
- $1 \leq q(op_i) \leq l_i$.
- for every op_j, if $m(op_i) \preceq m(op_j)$, $q(op_i) = 1$.

op_i locks $q(op_i)$ replicas of o_i. For example, suppose that there are five replicas of an object o_i and o_i has three operations op_{i1}, op_{i2}, and op_{i3}. Suppose that $m(op_{i1}) \prec m(op_{i2}) \prec m(op_{i3})$. $q(op_{i1}) = 1$. $q(op_{i2})$ and $q(op_{i3})$ are, for example, given as 2 and 3, respectively. Before computing op_{i2}, two replicas in five ones are locked.

An operation op_i locks an object o_i by the following scheme.

[Locking scheme]

(1) Before computing op_i, $q(op_i)$ replicas in $r(o_i)$ are locked in a mode $m(op_i)$. Here, let $s(op_i)$ be a subset of replicas in $r(o_i)$ which are locked here.

(2) If all replicas in $s(op_i)$ are locked, op_i is computed.
   (a) If op_i invokes operations in other objects, op_i is computed in one replica in $s(op_i)$.
   (b) Otherwise, op_i is computed in all the replicas.

(3) If some replica in $s(op_i)$ is not locked, op_i aborts.□

Since a more strict operation op locks more replicas, the operation op is aborted if other operations conflicting with op are manipulating the replicas. If a replica is in the same cell as an object invoking op_i, the replica is locked at step (1). The replicas whose QoS is better are selected to be locked as discussed in prereding subsection.

We would like to discuss how an operation op invoking op_i commits. The commitment of op_i on multiple replicas is controlled by the two-phase commitment [2]. One replica o_i^k in $s(op_i)$ plays a role of the coordinator and the other replicas are the participants.

[Commitment]

(1) o_i^k sends a *Prepare* message to all the replicas. The participant replica o_i^j which is not locked by op_i, i.e. replicas in $r(o_i) - s(op_i)$, is locked in the mode $m(op_i)$ on receipt of the *Prepare* message. If locked, the replica o_i^j sends back *Yes* message to o_i^k.

(2) If some replica o_i^j in $r(o_i) - s(op_i)$ is not locked, o_i^j sends *No* to o_i^k.

(3) If o_i^k receives *Yes* from all the participant replicas, o_i^k sends *Commit* to all the participants. If o_i^k receives *No* from some participant, o_i^k sends *Abort* to the participants sending *Yes*.

(4) If the participant replica o_i^j receives *Abort*, o_i^j abort op_i if o_i^j had computed op_i.

(5) If the participant replica o_i^j receives *Commit*, all the replicas in $r(o_i) - s(op_i)$ are locked.
   (a) Unless op_i invokes operations in other objects, op_i is computed in all replicas in $r(o_i)$ if op_i changes the state, otherwise op_i commits.
   (b) Otherwise, the state of the replica whose op_i is computed is sent to all the replicas. □

If op_i invokes an operations op_{ij} in another object and op_i is computed in o_i, op_{ij} is computed more than once. In order to avoid the iterated computation, op_i is computed in only one replica, say o_i. In stead of computing op_i in the other replica, the state of o_i is sent to all the other replicas of o_i. If op_i commits, all the locks on the replicas are released.

6 Concluding Remarks

In this paper, we have discussed how to support nested transactions manipulating replicated and mobile objects in the distributed system. We have modeled the mobile objects to be ones whose QoS is changed according to the movement of the objects. We have discussed the optimistic two-phase locking to maintain the mutual consistency among the replicas. Here, the read-one and write-all principal is extended so that the objects can support more kinds of abstract operations than *read* and *write*.

References

[1] Badrinath, B. R., Acharya, A., and Imielinski, T., "Structuring Distributed Algorithms

for Mobile Hosts," *Proc. of the 14th ICDCS*, 1994, pp. 21–28.

[2] Bernstein, P. A., Hadzilacos, V., Goodman, N., "Concurrency Control and Recovery in Database Systems," *Addison-Wesley*, 1987.

[3] Barbara, D. and Imielinski, T., "Sleepers and Workaholics: Caching Strategies in Mobile Environments," *Proc. of the ACM SIGMOD*, 1994, pp. 1–12.

[4] Carey, J. M. and Livny, M., "Conflict Detectin Tradeoffs for Replicated Data," *ACM TODS*, Vol. 16, No. 4, 1991, pp.703–746.

[5] Clifton, C., Garcia-Molina, H., and Bloom, D., "HyperFile: A Data and Query Model for Documents," *VLDB Journal*, Vol. 4, No. 1, 1955, pp. 45–86.

[6] Fischer, J. M., Nancy, A. L., and Michael, S. P., "Impossibility of Distributed Consensus with One Faulty Process," *Journal of ACM*, Vol.32, No.2, 1985, pp.374–382.

[7] Gray, J., "The Transaction Concept : Virtues and Limitations," *Proc. of VLDB*, 1981, pp. 144-154.

[8] Gruber, R., Kaashoek, F., Liskov, B., and Shrira, L., "Disconnected Operation in the Thor Object-Oriented Database System," *Proc. of IEEE Workshop on Mobile Computing Systems and Applications*, 1994, pp. 51–56.

[9] Huang, Y., Sistla, P., and Wolfson, O., "Data Replication for Mobile Computers," *Proc. of the ACM SIGMOD*, 1994, pp. 13–24.

[10] Imielinski, T., Viswanathan, S., and Badrinath, B. R., "Energy Efficient Indexing on Air," *Proc. of the ACM SIGMOD*, 1994, pp. 25–36.

[11] Jing, J., Bukhres, O., and Elmagarmid, A., "Distributed Lock Management for Mobile Transactions," *Proc of the 15th ICDCS*, 1995, pp. 118–125.

[12] Kistler, J. J. and Satyanaranyanan, M., "Disconnected Operation in the Coda File System," *ACM Trans. on Database Systems*, Vol. 10, No. 1, 1992, pp. 2–25.

[13] Korth, H. F., "Locking Primitives in a Database System," *JACM*, Vol. 30, No. 1, 1983, pp. 55-79.

[14] Kung, H. T. and Robinson, J. T., "On Optimistic Methods for Concurrency Control," *ACM Trans. on Database Systems*, Vol. 6, No. 2, 1981, pp. 213–226.

[15] Lu, Q. and Satyanaranyanan, M., "Isolation-Only Transactions for Mobile Computing," *ACM Operating Systems Review*, Vol. 28, No. 2, 1994, pp. 81–87.

[16] Moss, J. E., "Nested Transactions : An Approach to Reliable Distributed Computing," *The MIT Press Series in Information Systems*, 1985.

[17] Tanaka, R. and Tsukamoto, M., "A CLNP-based Protocol for Mobile End Systems within an Area," *Proc. IEEE International Conf. on Network Protocols (ICNP)*, 1993, pp. 64–71.

[18] Teraoka, F., Yokote, Y., and Tokoro, M., "A Network Architecture Providing Host Migration Transparency," *Proc. of ACM SIGCOMM*, 1991, pp. 209–220.

[19] Weikum, G. and Schek, H.-J., "Concepts and Applications of Multilevel Transaction and Open Nested Transactions," *Database Transaction Models for Advanced Applications*, 1992, pp. 516–553.

[20] Yasuzawa, S. and Takizawa, M., "Uncompensatable Deadlock in Distributed Object-Oriented Systems," *Proc. of the International Conf. on Parallel and Distributed Systems (ICPADS)*, 1992, pp. 150–157.

[21] Yeo, L. H. and Zaslavsky, A., "Submission of Transactions from Mobile Workstations in Cooperative Multidatabase Processing Environment," *Proc of the 14th ICDCS*, 1994, pp. 372–379.