システムソフトウェアと オペレーティング・システム 77－12
マルチメディア通信と分散処理 87－12
（1998. 2. 26）

Pseudo-Active Replication of Objects in Heterogeneous Processors

Tsunetake Ishida, Hiroaki Higaki and Makoto Takizawa

Tokyo Denki University
E-mail {tsune, hig, taki}@takilab.k.dendai.ac.jp

An approach to making distributed systems fault-tolerant is to replicate objects in the systems. The replicas have to be realized in various types of computers since it is difficult to get all the same types of computers. In the active replication, the replicas are computed and communicated in the same synchronous way. The response time depends on the computation time of the slowest replica and the longest delay time. In this paper, we discuss a pseudo-active replication where events may not occur simultaneously, not in the same order, and may not occur in the replicas. The succeeding requests can be issued to the replicas if some responses, not necessarily all response, are received from the replicas without waiting for the completion of the slower replicas.

不均質プロセッサにおける疑似能動的多重化

石田常竹　桧垣 博章　滝沢 誠

東京電機大学理工学部経営工学科
E-mail {tsune, hig, taki}@takilab.k.dendai.ac.jp

分散型アプリケーションは、複数のオブジェクトが互いにメッセージを交換し合い協調動作を行なうことで実現されている。航空管制システムのようなサービスの停止、中止が許されないミッションクリティカルなシステムでは、障害が発生しても、サービスを継続する能力(フォールトトレラント)が必要である。システムをフォールトトレラントなものにする手法の1つにプロセスの複製(レプリカ)を用いる手法がある。このレプリカが同機種のコンピュータ上で実行されることは稀である。さらに、能動的多重化では、全てのレプリカ上の演算と通信が同一の順序で処理される必要がある。これにより、プロセスの処理速度は最も遅いレプリカと最大遅延時間に依存する。本論文では、全レプリカで、同一イベントが、同一順序で起きるとは限らない疑似能動的多重化を提案する。本手法により、プロセスの処理は、処理速度の遅くメッセージの伝達遅延の大きいレプリカに依存せずに継続できる。

1 Introduction

In distributed applications, multiple autonomous application objects cooperate to achieve some objectives by exchanging messages. The object-based system like CORBA [13] is now getting a standard framework of the distribed systems. In the mission-critical disributed applications, objects are required to be fault-tolerant. The objects may suffer from faults, i.e. *stop* and *Byzantine* faults [9, 14]. An approach to making the system fault-tolerant is to replicate the objects in the system. In this paper, a collection of *replicas* of an object is named a *cluster*. In the *active* replication [15] adopted by Isis [3], every replica does the same computation and communication. In the *passive* one [4], only one primary replica does the computation and communication. The primary replica takes periodically a checkpoint and sends the state information at the chackpoint to all the backup replicas. The replicas can continuously support the service in the presence of faults in the active replication while it takes time to recover from the fault of the primary replica in the passive one. That is, the passive replication is less available than the active one. In order to reduce the communication overhead in the active replication, the *hybrid* replication [19] is proposed. It is similar to the active one except that only primary replica sends the response back and sends periodically the checkpoints to the other replicas as the passive one. The active replication approach implies more redundant processing and communication than the passive one. But the computation can be continued as long as at least one replica is operational.

If the replicas in a cluster are allocated in various types of computers and communication delays between replicas are different, the cluster is *heterogeneous*. The computers have different processing speeds and levels of reliability. The computation of an object complete only if the computations of all the replicas complete in the active replication. In this paper, we discuss a novel *pseudo-active* replication in order to reduce the response time and the total processing time and to support the same level of reliability and availability as the active one in the heterogeneous clusters. Here, the computation of the object can complete if the the faster replicas complete the computation without waiting for the slower replicas. The slower replicas have to catch up with the faster ones. We discuss a *distributed* way where each replica detects the slower replicas by using the vector clock [10] carried by messages. In addition, we discuss how the slower replicas catch up with the faster ones by omitting events in the slower ones. The pseudo-active replication can support shorter response time and total computation time of the replicas than the active replication even if slower

and farther computers are included. In addition, the pseudo-active replication supports the same level of reliability as the active one, i.e. the process is operational as long as at least one replica is operational.

In section 2, we present the system model. In section3, the active replication is overviewed. In section 4, we discuss the pseudo-active replication. In section 5, we evaluate the pseudo-active replication in terms of total computation time compared with the active one.

2 System Model

2.1 Clusters

A distributed system is composed of multiple computers interconnected by the communication network. A distributed application is realized by the cooperation of a group G of multiple objects $o_1, \ldots, o_n$. Each object is stored in a computer. On receipt of a request m from o_j, o_i computes m and sends back the response to o_j.

A collection $\{o_{i1}, \ldots, o_{il_i}\}$ $(l_i \geq 1)$ of replicas of o_i is a *cluster* c_i [Figure 1]. c_i is a *one-replica* cluster iff c_i includes only one replica of c_i. c_i is *homogeneous* iff all the replicas in c_i are in the same type of computers and the communication delay between every pair of replicas is the same. Each computer is characterized in terms of processing speed, delay time, and reliability level. c_i is *heterogeneous* if some replicas are in different types of computers. For example, a replica o_{ij} is computed in a faster computer like UltraSparc station and another o_{ik} in a slower computer like Sparc5. Even if all the computers are the same type, c_i is heterogeneous if the delay times among the replicas are different. For example, two computer A and B are in the same local area network but another C is connected with the Internet. It takes a longer time for A to send a message to C than B.

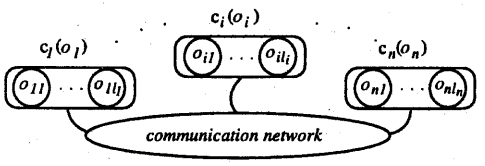

Figure 1: Clusters.

The network is assume to support the reliable data transmission between every pair of replicas in the group. That is, each replica can deliver messages to every other replica without loss and duplication in the sending order.

2.2 Causal precedence

It is significant to discuss in which order events occur in distributed systems. Let $s_{ij}(m)$ and $r_{ij}(m)$ denote the sending and receipt events of a message m in a replica o_{ij}, respectively.

[Causal precedence] [8] An event e_1 *causally precedes* e_2 $(e_1 \rightarrow e_2)$ iff one of the following conditions holds:

(1) e_1 occurs before e_2 in some object.

(2) $e_1 = s_{ij}(m)$ and $e_2 = r_{kl}(m)$.

(3) $e_1 \rightarrow e_3 \rightarrow e_2$ for some event e_3. □

[Definition] A message m_1 *causally precedes* m_2 $(m_1 \rightarrow m_2)$ iff $s_{ij}(m_1) \rightarrow s_{kl}(m_2)$. □

Here, m_1 and m_2 are *concurrent* $(m_1 \parallel m_2)$ iff neither $m_1 \rightarrow m_2$ nor $m_2 \rightarrow m_1$. The replicas have to deliver m_1 before m_2 if $m_1 \rightarrow m_2$. Many group communication protocols [3, 5, 7, 11, 12, 17, 18] are proposed to support a group of multiple processes with the causally ordered delivery of messages.

In order to causally deliver messages, each o_{ij} manipulates the vector clock [3,10] $V = \langle V_{kh} \mid k = 1, \cdots, n, h = 1, \cdots, l_i \rangle$. Each element V_{kh} shows the local clock of o_{kh} which o_{ij} knows. Initially, $V_{kh} = 0$. Each time o_{ij} sends a message m, V_{ij} is incremented by one. m carries the local clock $m.V$, i.e. $m.V_{kh} = V_{kh}$ for every k and h. On receipt of a message m, o_{ij} manipulates V as $V_{kh} := \max(V_{kh}, m.V_{kh})$ for every k and h. The vector clock satisfies the following property [10].

[Property] For every pair of messages m_1 and m_2, $m_1 \rightarrow m_2$ iff $m_1.V < m_2.V$. □

By using the vector clock, the messages received are causally ordered.

3 Active Replication

Replicas $o_{i1}, \ldots, o_{il_i}$ of an object o_i are often obliged to be distributed to various types of existing computers, i.e. the cluster c_i is *heterogeneous*. In the active replication, every replica is required to do the same computation and communication at the same time [15]. Here, suppose that an object o_s sends a request to o_{ij} and o_{ik}. If o_{ij} is faster and nearer to o_s than o_{ik}, o_s has to wait for the response from o_{ik} even after receiving the response from o_{ij}. Thus, every replica has to wait for the slower and farther replicas in c_i. Hence, we propose a *pseudo-active* replication by relaxing the constraints of the *active* replication in order to decrease the response time in the heterogeneous cluster.

Each replica o_{ij} can be modeled as a deterministic finite state machine [Figure 2]. Let s^0_{ij} denote the initial state of o_{ij}. Here, $s^0_{i1} = \cdots = s^0_{il_i}$. s^h_{ij} is transit-ed to s^{h+1}_{ij} if e^{h+1}_{ij} occurs. Here, the state of o_{ij} is represented by s^0_{ij} and a sequence of the events $e^1_{ij} \circ \cdots \circ e^{t_{ij}}_{ij}$. Here, let $e^h_{ij}(s^{h-1}_{ij})$ denote a state s^h_{ij} and $e^{h-1}_{ij} \circ e^h_{ij}(s^{h-2}_{ij}) = e^h_{ij}(s^{h-1}_{ij}) = s^h_{ij}$. There occur local events and communication events, i.e. *sending* and *receipt* events. e^h_{ij} denotes a instance of an event e^h in o_{ij}.

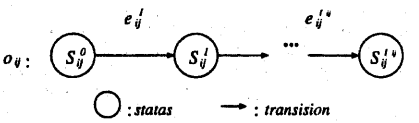

Figure 2: State transition.

o_i is *actively replicated* in a cluster c_i if the following conditions hold.

[Active replication (AR) conditions]

AR1: For every pair of operational replicas o_{ij} and o_{ik}, $s^h_{ij} = s^h_{ik}$ and $e^h_{ij} = e^h_{ik}$ for every h.

AR2: For every pair of operational replicas o_{ij} and o_{ik}, e_{ij}^h causally precedes e_{ik}^{h+1} ($e_{ij}^h \rightarrow e_{ik}^{h+1}$) iff $e_{ik}^h \rightarrow e_{ij}^{h+1}$ for every h.

AR3: Every operational replica o_{ij} does not lose any event. □

AR1 means that every replica does the same computation and communication in c_i, i.e. the same events occur in the same order.

AR2 means that every event occurs simultaneously in every replica. Every event e_{ij}^{h+1} occurs in o_{ij} after e_{ik}^h occurs in every o_{ik}. o_{ij} and o_{ik} are *synchronized*. Suppose that an application object o_i sends a message m. In the active replication, each replica o_{ij} sends m. A message m^{ij} sent by a replica o_{ij} is referred to as an *instance* of m^{ij}.

[Proposition] o_{ij} and o_{ik} are *synchronized* if $m_s^{ij} \rightarrow m_t^{ik}$ iff $m_s^{ik} \rightarrow m_t^{ij}$ for every pair of messages m_s and m_t sent/received by o_{ij} and o_{ik}. □

In Figure 3, o_{ij} and o_{ik} receive a message m_1. After receiving m_1, o_{ij} and o_{ik} send m_2^{ij} and m_2^{ik}, respectively. Since $m_1^{hl} \rightarrow m_2^{ij}$ and $m_1^{hl} \rightarrow m_2^{ik}$, o_{ij} and o_{ik} are synchronized.

AR3 means that every replica o_{ij} does the same computation as the object o_i. If every replica misses some event, AR3 does not hold although AR1 and AR2 hold. o_{ij} *includes* o_{ik} iff every event in o_{ik} occurs in o_{ij}.

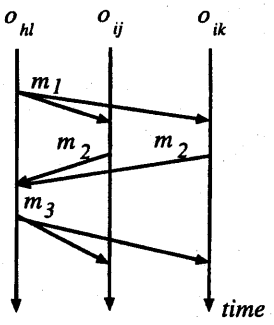

Figure 3: Synchronized replicas.

4 Pseudo-Active Replication

4.1 Following relation

Replicas in the faster and nearer computers support shorter response time than slower and farther ones. In the active replication, the computation speed of the cluster c_i depends on the slowest and farther replica in c_i because AR2 has to be satisfied. The response time can be reduced if the computation of c_i completes before the computation of every replica completes.

First, we consider a case that AR1 holds but AR2 does not hold. That is, every event occurs in the same order but may not occur simultaneously.

[Definition] o_{ik} *follows* o_{ij} ($o_{ij} \Rightarrow o_{ik}$) iff $e_{ij} = e_{ik}$, $e'_{ij} = e'_{ik}$, $e_{ij} \rightarrow e'_{ij}$, $e_{ik} \rightarrow e'_{ik}$, and $e_{ij} \rightarrow e'_{ik}$, but $e_{ik} \not\rightarrow e'_{ij}$ for some events e_{ij} and e'_{ij} occurring in o_{ij} and e_{ik} and e'_{ik} in o_{ik}. □

o_{ij} and o_{ik} are synchronized iff neither $o_{ij} \Rightarrow o_{ik}$ nor $o_{ik} \Rightarrow o_{ij}$. If $o_{ij} \Rightarrow o_{ik}$ and $o_{ik} \Rightarrow o_{ij}$, o_{ij} and o_{ik} are *thrashed*. o_{ij} and o_{ik} may be thrashed if one of o_{ij} and o_{ik} sometimes gets slower due to the overload.

[Definition] o_{ik} *fully follows* o_{ij} ($p_{ij} \approx\!\!\Rightarrow o_{ik}$) iff $e'_{ij} \rightarrow e'_{ik}$ if $e_{ij} \rightarrow e_{ik}$ for every events e_{ij} and e'_{ij} in o_{ij}, and e_{ik} and e'_{ik} in o_{ik} such that $e_{ij} = e_{ik}$, $e'_{ij} = e'_{ik}$, $e_{ij} \rightarrow e'_{ij}$, and $e_{ik} \rightarrow e'_{ik}$. □

For example, if the computer of o_{ij} is always faster than o_{ik}, o_{ik} fully follows o_{ij}.

A cluster c_i is *regular* iff every pair of replicas o_{ij} and o_{ik} are synchronized or $o_{ij} \approx\!\!\Rightarrow o_{ik}$ for every i and k. In an irregular cluster c_i, both $o_{ij} \Rightarrow o_{ik}$ and $o_{ik} \Rightarrow o_{ij}$ hold in some o_{ij} and o_{ik}.

Suppose that o_{ij} and o_{ik} send message instances m_2^{ij} and m_2^{ik}, respectively, after receiving m_1 before m_3 as shown in Figure 4. In the active replication, o_{hl} is required to send m_3 after receiving m_2 from every replica. However, o_{hl} sends m_3 after receiving m_2^{ij} without waiting for m_2^{ik}. On receipt of m_3, o_{ij} and o_{ik} know that $m_2^{ij} \rightarrow m_3$ but $m_2^{ik} \not\rightarrow m_3$, i.e. that o_{hl} sends m_3 before receiving m_2^{ik} from o_{ik}. Hence, o_{ij} and o_{ik} can decide which one follows the other according to the following theorem:

[Theorem 1] o_{ik} *follows* o_{ij} ($o_{ij} \Rightarrow o_{ik}$) if $m_s^{ij} \rightarrow m_t$ but $m_s^{ik} \not\rightarrow m_t$ for some messages m_t received and m_s sent by o_{ij} and o_{ik}. □

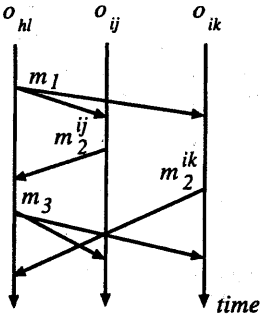

Figure 4: o_{ik} follows o_{ij}.

In Figure 4, o_{ik} follows o_{ij} on receipt of m_3 since $m_2^{ij} \rightarrow m_3$ but $m_3 \parallel m_2^{ik}$. o_{ij} and o_{ik} are synchronized before receipt of m_3. For example, suppose m_1 and m_3 are requests and m_2 is the response of m_1. Here, suppose it takes a longer time to compute the request m_1 in o_{ik} than o_{ij}. That is, o_{ik} is slower than o_{ij} or the communication delay to o_{ik} is larger than o_{ij}. Without waiting for all the responses of m_1, o_{hl} can send another request m_3 to o_{ij} and o_{ik} on receipt of m_2^{ij}. By this method, the response time can be reduced even if the system includes slower computers.

4.2 Decision rule

Next, we consider how each replica o_{ij} can decide in the distributed way if o_{ij} follows or suc-

ceeds other replicas in the cluster c_i. Suppose that o_{ij} receives a message m carrying the vector clock $m.V$ [9]. From Theorem 1, o_{ik} knows that o_{ik} follows o_{ih} if $m.V_{ik} < m.V_{ih}$. Hence, o_{ij} decides how the replicas are followed on receipt of m according to the following rule.

[**Following (F) rule**] For every pair of replicas o_{ih} and o_{ik} in c_i,

(1) o_{ik} follows o_{ih} if $m.V_{ik} < m.V_{ih}$,

(2) o_{ik} and o_{ih} are synchronized if $m.V_{ik} = m.V_{ih}$. □

o_{ik} *follows* o_{ih} *in* o_{ij} ($o_{ih} \Rightarrow_{ij} o_{ik}$) if o_{ij} decides that $o_{ih} \Rightarrow o_{ik}$. This means that o_{ij} knows that o_{ik} follows o_{ih}. Here, o_{ij} considers that o_{ih} is the *fastest* and *slowest* in c_i if $m.V_{ih}$ is the maximum and the minimum in $m.V$, respectively.

Suppose o_{hl} in c_h sends m_1 and m_3 to o_{ij} and o_{ik} in c_i, and o_{tu} in c_t sends m_4 and m_6 as shown in Figure 5. o_{hl} sends m_3 on receipt of m_2^{ij} from o_{ij} before receipt of m_2^{ik} from o_{ik}. Hence, on receipt of m_3, $o_{ij} \Rightarrow_{ij} o_{ik}$ and $o_{ij} \Rightarrow_{ik} o_{ik}$. o_{tu} sends m_6 on receipt of m_5^{ik} from o_{ik} before receipt of m_5^{ij}. Hence, on receipt of m_6, o_{ij} and o_{ik} know that $o_{ik} \Rightarrow o_{ij}$. On receipt of m_3 and m_5, o_{ij} and o_{ik} are thrashed, i.e. $o_{ij} \Rightarrow o_{ik}$ and $o_{ik} \Rightarrow o_{ij}$.

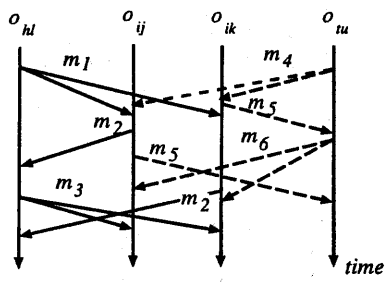

Figure 5: Thrashing.

It is straightforward for the following theorem to hold since every replica is assumed to receive all messages in the same order.

[**Theorem 2**] On receipt of a message m, $o_{ij} \Rightarrow_{ih} o_{ik}$ iff $o_{ij} \Rightarrow_{il} o_{ik}$ for every pair of operational replicas o_{ih} and o_{il}. □

If o_{ih} knows that o_{ij} follows o_{ik} by the F rule on receipt of m, it is sure that $o_{ij} \Rightarrow_{ih} p_{ik}$ in another o_{ih} receiving m.

[**Definition**] A message m is *delayed* on o_{ij} if $m.V_{ij} < m.V_{ik}$ for some o_{ik}. □

If o_{ij} receives a message delayed on o_{ij}, o_{ij} knows that o_{ij} follows some replica.

4.3 Equivalent sequences of events

The slower replica o_{ik} has to catch up with the faster full replica o_{ij} in c_i. If every event stored in the receipt queue is required to occur in o_{ik}, o_{ik} cannot catch up with o_{ij} since o_{ij} is faster than o_{ik}. Hence, we try to make o_{ik} catch up with o_{ij} by omitting events not necessarily to occur.

First, we try to relax the AR1 condition.

[**Definition**] An event e is an *identity* event in o_{ij} iff $e(s_{ij}) = s_{ij}$ for every state s_{ij} of o_{ij}. An event sequence S is *idempotent* in o_{ij} iff $S \circ S(s_{ij}) = S(s_{ij})$ for every state s_{ij} of o_{ij}. □

For example, a SQL *select* statement is an identity event.

[**Definition**] An event sequence S_1 is *absorbed* by S_2 iff $S_1 \circ S_2(s_{ij}) = S_2(s_{ij})$ for every state s_{ij} of o_{ij}. □

Suppose a write event w_1 of a value v_1 occurs before w_2 of v_2 in o_{ij}. Since v_1 is overwritten by v_2, w_2 absorbs w_1.

[**Definition**] An event sequence S_1 is *equivalent* with S_2 in o_{ij} ($S_1 \equiv S_2$) iff $S_1(s_{ij}) = S_2(s_{ij})$ for every state s_{ij} of o_{ij}. □

In Figure 6, o_{ij} includes o_{ik}. If u and w are identities, $c = b$ and $e = d$, $t \circ v \equiv t \circ u \circ v \circ w$. Here, o_{ih} can catch up with o_{ij} by omitting two identity events u and w.

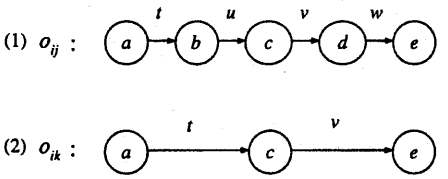

Figure 6: Indentity Events.

[**Omission rules**] Let S_1 and S_2 be event sequences and e be events.

(1) $S_1 \circ e \circ S_2 \equiv S_1 \circ S_2$ if e is an identity.

(2) $e \circ S \circ e \equiv S_1 \circ e$ if e is idempotent and S_1 includes no event conflicting with e. □

That is, the slower replicas can omit identity and idempotent events occurring in the faster replicas.

4.4 Catching up

We discuss how the slower replicas catch up with the faster ones by using the omission rule. 7, suppose o_{ik} follows o_{ij}. In the computation of a request m_1, o_{ik} receives requests m_3 and m_5 from o_{hl}. Since o_{ik} is sure that o_{ij} completes m_3 and o_{hl} receives the response m_4 of m_3 from o_{ij}, o_{ik} does not need to compute m_3. However, o_{ik} is not sure that o_{ij} completes m_5.

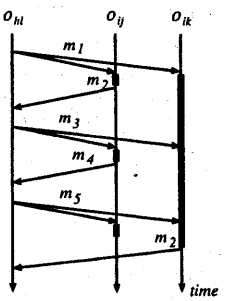

Figure 7: Obsolete message.

[**Definition**] A message m is *obsolete* in o_{ij} iff

(1) m is received but is not delivered to o_{ij},

(2) m is delayed on o_{ij}, and

(3) there is some message m' received in o_{ij} such that $m \rightarrow m'$, m' is delayed on o_{ij}, and $m.V_{ij} < m'.V_{ik}$ for some o_{ik}. □

Thus, o_{ij} can omit an *obsolete* request m since m is surely finished already in the faster replica o_{ik}.

On receipt of a message m, o_{ij} stores m in the receipt queue RQ_{ij}. The messages in RQ_{ij} are causally sequenced. o_{ij} takes a top message m from RQ_{ij} and computes m, In Figure 7, m_3 and m_5 are stored in RQ_{ij} and $m_3 \rightarrow m_5$ during the computation of m_1. After sending m_2, o_{ik} takes m_3 from RQ_{ij}.

o_{ij} executes the following procedure to check if a top message in RQ_{ij} is obsolete.

[Receipt] A message m arrives at o_{ij}.

(1) m is stored in RQ_{ij} in the causal order by using the vector clock.

(2) If m is delayed on o_{ij}, m is marked *delayed*.

(3) If m is delayed, RQ_{ij} is searched for every *delayed* message m' causally preceding m ($m' \rightarrow m$) in RQ_{ij}.

(3-1) if m' is an identity request, m' is marked *omissible*.

(3-2) if m' is idempotent and there is the same kind of request m'' as m' which precedes m and succeeds m' in RQ_{ij}, m' is marked *omissible* if there is no message between m and m' in RQ_{ij} which conflicts with m'. □

o_{ij} takes the top message m from RQ_{ij}. If m is marked *omissible*, o_{ij} removes m and sends back the dummy response of m with no result.

In Figure 7, suppose m_2 and m_4 are the idempotent requests. m_3 arrives at o_{ik} during the computation of the request m_1. Here, m_3 is delayed on o_{ik} but not obsolete. m_3 is enqueued into RQ_{ij}. Then, the request m_5 arrives at o_{ik}. m_5 is delayed on o_{ik}. The messages in RQ_{ij} are checked by the receipt procedure if they are obsolete. m_3 in RQ_{ij} is obsolete since $m_3 \rightarrow m_5$ and m_5 is delayed. Hence, m_3 is marked *omissible*. After the completion of m_1, i.e. sending m_5, o_{ik} takes m_3 from RQ_{ij}. Since m_3 is marked omissible, o_{ik} omits m_3 and sends back a dummy response of m_3 to o_{hl}. Then, o_{ik} starts to compute m_5 since m_5 is not obsolete while being delayed.

5 Evaluation

The pseudo-active replication supports the same level of reliability as the active one, i.e. the cluster can service as long as at least one replica is operational. Let c_i be a cluster of of an object o_i. Let d_i be another cluster of o_i. Here, suppose that an object o_h receives multiple instances m^{i1}, ..., m^{il_i} of a message m from $o_{i1} \ldots o_{il_i}$. o_h takes only one of them. The sequence of messages from c_i is an *output sequence* of c_i to o_h.

[Definition] A cluster c_i is *equivalent* to d_i ($c_i \equiv d_i$) iff every output sequences of c_i and d_i are the same for every input sequence of messages. □

[Theorem 3] In the pseudo-active replication, every cluster c_i is equivalent to some one-replica cluster of o_i. □

We evaluate the pseudo-active replication cluster $c_i = \{o_{i1}, \ldots, o_{il_i}\}$ in terms of response time and computation time compared with the active one. An object o_h sends requests to the replicas in c_i and receives the responses from the replicas. Let δ_i be the propagation delay time between o_h and the replicas in c_i. There are f_i types of requests which o_{ij} can take. π_i^i, π_i^d, and π_i^o denote probabilities that one request type is an identity, idempotent, and other one in c_i, respectively. Here, $\pi_i^i + \pi_i^d + \pi_i^o = 1$. We assume that every pair of operations conflict if both of them are not identities. We assume that o_{i1} is the fastest and o_{il_i} is the slowest in c_i. Suppose that o_h issues w requests to c_i. After sending a request r, o_h sends a succeeding request to c_i on receipt of the response of r. In this paper, we assume that it takes τ_{ij}^i, τ_{ij}^d, and τ_{ij}^o time units to compute each identity, idempotent, and other event, respectively, in o_{ij}. Let τ_{ij} be an average computation time of each event in o_{ij}, i.e. $\tau_{ij} = \pi_i^i \cdot \tau_{ij}^i + \pi_i^d \cdot \tau_{ij}^d + \pi_i^o \cdot \tau_{ij}^o$. We assume that $\tau_{ij}^i/\tau_{i1}^i = \tau_{ij}^d/\tau_{i1}^d = \tau_{ij}^o/\tau_{i1}^o = \tau_{ij}/\tau_{i1}$ (≤ 1). It is expected to take $w \cdot \tau_{ij}$ time units to compute w requests in o_{ij}. The expected processing time is $R_P = w \cdot (\tau_{i1} + 2 \cdot \delta_i)$ in the pseudo-active replication while $R_A = w \cdot (\tau_{il_i} + 2 \cdot \delta_i)$ in the active one. It is clear that $R_P \leq R_A$ since $\tau_{i1} \leq \tau_{il_i}$.

If no event is omitted in every o_{ij}, w requests are computed. Hence, the total computation time of c_i in the active replication is $\Sigma_{j=1,\ldots,l_i} w \cdot \tau_{ij}$. In the pseudo-active one, some *obsolete* events are omitted. Let us consider the total computation

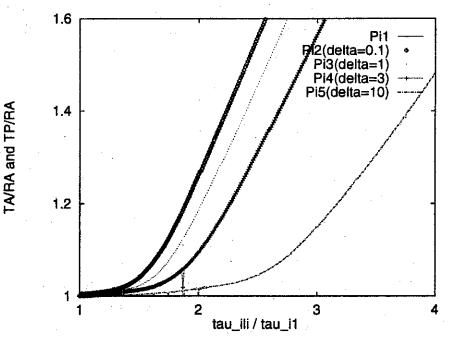

Figure 8: Ratio of total computation time.

times of the fastest replica o_{i1} and slowest o_{il_i}. The total time of o_{ij} is defined to be a duration from time when o_{ij} receives the first request until when o_{ij} sends the response of the last request. Let T_A be the total time of o_{il_i} to compute the requests in the active replication, i.e. no requests are omitted. Let T_P be the total computation time of o_{il_i} in the pseudo-active replication. Here, suppose $\pi_i^i = 0.5$, $\pi_i^d = 0.3$, and $\pi_i^o = 0.2$. o_h sends w ($= 100$) requests by selecting randomly operations in f_i ($= 10$) ones. Let δ be a ratio of the propagation delay δ_i among o_h and the replicas in c_i to the processing time τ_{i1}^i of the identity

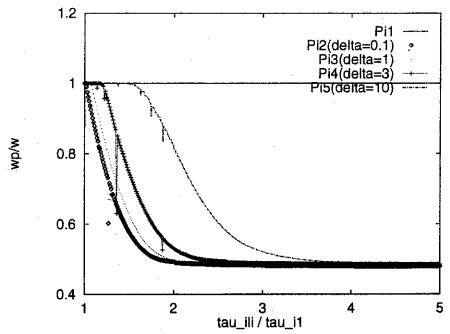

Figure 9: Ratio of omissible events.

request in o_{i1}, δ_i/τ^i_{i1}. By the simulation, the total processing times T_P and T_A are obtained. The number w_P of operations computed in o_{il_i} in the pseudo-active replication is also obtained. Figure 8 shows T_A/R_A and T_P/R_A with $\delta = 0.1, 1, 3, 10$ for τ_{i1}/τ_{il_i}. The dotted lines show T_P/R_A. Figure 9 shows w_P/w with δ for τ_{i1}/τ_{il_i}.

Figures 8 and 9 show the pseudo-active replication can reduce the total processing time and the number of operations computed in the slower replica. Furthermore, the longer the distance among o_h and the replicas is, the more efficient the pseudo-active replication is. In Figure 9, let us consider w_P/w for $\delta = 3$. If it takes 10 msec. to compute the identity request like *read* in o_{i1}, the propagation delay is 30 msec. If o_{il_i} is five times slower than o_{i1}, the number of requests computed in o_{il_i} is the same, i.e. 50%. This means that every identity request is omitted in o_{il_i} since every request issued to o_{il_i} is queued in RQ_{ij}.

6 Concluding Remarks

This paper has discussed the pseudo-active replication in the heterogeneous cluster. In the heterogeneous cluster, the computation of the cluster can complete if the fastest replica completes while the slower replicas are still being computed. We have presented the vector clock based method by which each replica can decide how the replica follows others. The slower replicas can catch up with the fastest replicas by omitting the identity and idempotent events and changing the commutative events. We have shown that the pseudo-active replication implies shorter response time and total computation time than the active one while supporting the same level of reliability as the active one.

References

[1] Agarwal, D. A., Moser, L. E., Melliar-Smith, P. M., and Budhia, R. K., "A Reliable Ordered Delivery Protocol for Interconnected Local-Area Networks," *Proc. of IEEE ICNP*, 1995, pp.365–374.

[2] Amir, Y., Moser, L. E., Melliar-Smith, P. M. Agarwal, D. A., and Ciarfella, P., "The Totem Single-Ring Ordering and Membership Protocol," ITACM *TOCS*, Vol.13, No.4, 1995, pp.311–342.

[3] Birman, P. Kenneth, and Renesse, V. Robbert, "Reliable Distributed Computing with the Isis Toolkit," *IEEE CS Press*, 1994.

[4] Budhiraja, N., Marzullo, K., Schneider, B. F., and Toueg, S., "The Primary-Backup Approach," *Distributed Computing Systems, ACM Press*, 1994, pp.199-221.

[5] Ezhilchelvan, P. D., Macedo, R. A., and Shrivastava, S. K., "Newtop: A Fault-Tolerant Group Communication Protocol," *Proc. of IEEE ICDCS-15*, 1995, pp.296–307.

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

[7] Florin, G. and Toinard, C., "A New Way to Design Causally and Totally Ordered Multicast Protocols," *ACM Operating Systems Review*, Vol.26, No.4, 1992, pp.77–83.

[8] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM*, Vol.21, No.7, 1978, pp.558–565.

[9] Lamport, L., Shostak, R., and Pease, M., "The Byzantine Generals Problem," *ACM Trans. Programming Languages and Systems*, Vol.4, No.3, 1982, pp.382–401.

[10] Mattern, F., "Virtual Time and Global States of Distributed Systems," *Parallel and Distributed Algorithms* (Cosnard, M. and Quinton, P. eds.), *North-Holland*, 1989, pp.215–226.

[11] Melliar-Smith, P. M., Moser, L. E., and Agrawala, V., "Broadcast Protocols for Distributed Systems," *IEEE Trans. on Parallel and Distributed Systems*, Vol.1, No.1, 1990, pp.17–25.

[12] Nakamura, A. and Takizawa, M., "Causally Ordering Broadcast Protocol," *Proc. of IEEE ICDCS-14*, 1994, pp.48–55.

[13] *Object Management Group Inc.*, "The Common Object Request Broker Architecture and Specification," Rev. 2.0, 1995.

[14] Schneider, B. F., "Byzantine Generals in Action: Implementing Fail-Stop Processors," *ACM Tran. on Computing Systems*, Vol.2, No.2, 1993, pp.145-154.

[15] Schneider, B. F., "Replication Management using the State-Machine Approach," *Distributed Computing Systems, ACM Press*, 1993, pp.169-197.

[16] Shima, K., Higaki, H., and Takizawa, M., "Fault-Tolerant Causal Delivery in Group Communication," *Proc. of IEEE ICPADS'96*, 1996, pp.302-309.

[17] Tachikawa, T. and Takizawa, M., "Selective Total Ordering Broadcast Protocol," *Proc. of IEEE ICNP-94*, 1994, pp.212-219.

[18] Tachikawa, T. and Takizawa, M., "Distributed Protocol for Selective Intra-group Communication," *Proc. of IEEE ICNP-95*, 1995, pp.234-241.

[19] Thomas, L. C. and Mukesh, S., "The Delta-4 Distributed Fault-Tolerant Architecture," *Distributed Computing Systems, IEEE CS Press*, 1990, pp.223-247.