Vol. 41 No. 2 Transactions of Information Processing Society of Japan Feb. 2000

Regular Paper

Pseudo-active Replication in Wide-area Network

Hiroaki Higaki,† Katsuya Tanaka† and Makoto Takizawa†

In order to realize mission-critical applications in distributed systems, the systems are required to be fault-tolerant. In this paper, we discuss how to replicate a server in order to obtain fault-tolerant services. In the active replication, all the requests from a client are performed by all the server replicas in the same order. The replicas are rather placed on heterogeneous computers and the distributed applications are now being realized in wide-area networks, e.g., the Internet, where each communication channel has different message transmission delay and bandwidth. Hence, the response times of the replicas observed by the clients are not the same. We newly propose a pseudo-active replication where the client does not wait for any response after receiving one response and slower replicas do not perform every request performed by the faster replicas.

1. Introduction

Client-server applications are widely developed in a world-wide manner by using the Internet. Here, in order to implement mission critical applications, the systems are required to be fault-tolerant. One way to realize fault-tolerant applications is an *active replication*[1),5),6),11),13)]. In the active replication, all the replicated objects are required to be synchronized to perform all the requests issued by the clients in the same order. Each replica may be placed on a different kind of computer with different processing speed and different level of reliability.

Therefore, the synchronization among the replicas induces an additional time-overhead. The response time for the application in a *client* depends on the speed of the slowest replica. The authors propose a *pseudo-active* replication[8),14)]. Here, after receiving a response from the fastest replica, the client does not wait for all the other responses. Here, not all the replicas are required to be synchronized. By using the pseudo-active replication, the synchronization overhead is reduced and the response time for the application in the client is also reduced.

In the proposed protocol for the pseudo-active replication discussed in Refs. 8) and 14), the difference of processing speed among the replicas is measured by a client according to the receipt order of response messages from the replicas. This method works well in a local-area network. However, in the wide-area networks, the response time is affected by not only the processing speed but also the message transmission delay. If the replicas are distributed in a wide-area network and multiple clients communicate with them, a pair of clients may take different replicas as faster ones. In this paper, we propose a novel protocol to realize the pseudo-active replication in a wide-area network. In the pseudo-active replication, a slower replica omits some requests waiting to be performed in the queue in order to catch up with the fastest one[8),14)]. In the proposed protocol, the processing order of requests from multiple clients are intentionally changed in each replica. By using this method, the response time for the requests from clients is reduced and the total processing time in the replicated objects may be also reduced.

In Section 2, we review the pseudo-active replication and discuss the implementation of pseudo-active replication for a heterogeneous wide-area network. In Section 3, we show a protocol for realizing our idea and some properties satisfied by our protocol. In Section 4, we show the evaluation results of our protocol.

2. Pseudo-active Replication in Wide-area Networks

2.1 System Model

In the network system $\mathcal{S}$, *objects* are distributed in *servers*. An object o_i is composed of data and operations for manipulating the data. Applications in *clients* send *request messages* to servers to manipulate objects and the servers send the *response message* back to the clients. On receipt of the request, a server invokes an operation specified by the request message to

† Department of Computers and Systems Engineering, Tokyo Denki University

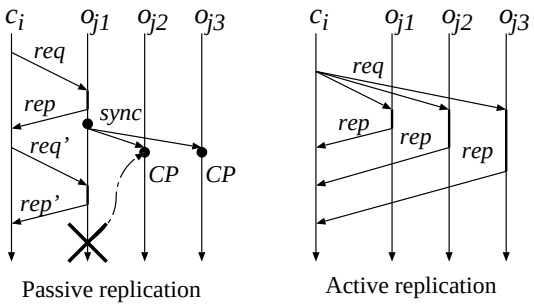

Fig. 1 Passive and active replication.

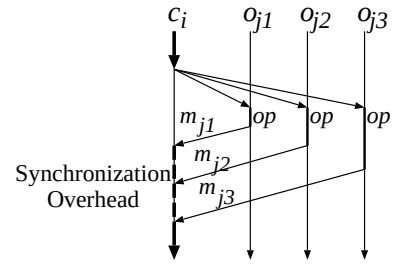

Fig. 2 Synchronization overhead in active replication.

manipulate an object. In order that the application programs are executed fault-tolerantly in $\mathcal{S}$, each object o_j^s is replicated and the *replicas* of the object are located on different computers. Here, let o_{jk} $(k = 1, \ldots, n_j)$ denotes a replica of an object o_j. Each replica is composed of the same data and the same operations.

2.2 Passive and Active Replication

There are two main approaches to replicating objects in servers: *passive* and *active* replications (**Fig. 1**). In the passive replication[3),4)], only one of the replicas is operational. That is, it receives request messages from clients, performs them, and sends back response messages to clients. The operational replica is the *primary* one. The other replicas are *passive*, i.e., these replicas do not invoke any operation. A client c_i sends a request message only to the primary replica o_{j1}. o_{j1} invokes the operation requested by c_i and sends back a response message to c_i. o_{j1} sometimes sends the state information to the other replicas o_{jk} $(2 \leq k \leq n_j)$ and every o_{jk} updates the state information. This is called a *checkpoint*. If o_{j1} fails, one of the passive replicas, say o_{j2}, takes over o_{j1} and becomes the primary replica. Here, o_{j2} restarts the execution of the application from the checkpoint taken most recently. Hence, the recovery procedure takes time because o_{j2} has to reinvoke the operations that the failed primary replica o_{j1} has already finished before the failure.

On the other hand, in the active replication[1),2),5),6),11),13)], all the replicas are operational. A client c_i sends request messages to all the replicas o_{jk} $(1 \leq k \leq n_j)$ of an object o_j. Every replica o_{jk} invokes the operation requested by c_i and sends back a response message to c_i. After receiving all the response messages, c_i delivers the response message to the application. Since every operational replica synchronously performs the same requests, the applications can be performed as long as there exists an operational replica and the recovery procedure requires less time-overhead than that in the passive replication.

2.3 Pseudo-active Replication

In the conventional active replication, all the replicas $o_{j1}, \ldots, o_{jn_j}$ of an object o_j are synchronized to perform the same request messages in the same order. Here, the computers on which the replicas are located are assumed to be the same kind, i.e., computers with the same processing speed and the same reliability, and connected with the same local-area network. That is, it takes almost the same time to perform a requested operation and to transmit a request message and a response message between a client c_i and the replicas o_{jk}. Therefore, c_i can receive every response message from the replicas o_{jk} at almost the same time. This assumption is reasonable only in a local-area network.

However, a wide-area network like the Internet is intrinsically *heterogeneous*. Different kinds of computers are connected with different kinds of networks. That is, there are different types of computers with respect to processing speed, reliability and availability, and different types of networks with respect to message transmission delay and message loss ratio[15)]. Here, it is difficult for a client to receive all the response messages from the replicas simultaneously. In **Fig. 2**, a client c_i delivers the result of an operation *op* to the application after receiving the response message m_{j3} from the slowest replica o_{j3}, i.e., the application in c_i is blocked until receiving m_{j3}.

The authors have proposed a *pseudo-active replication*[8),14)] where a client c_i only waits for the first response message from some replica o_{jk} under an assumption that only the *stop-faults* occur in the replicas, i.e., no failed replica

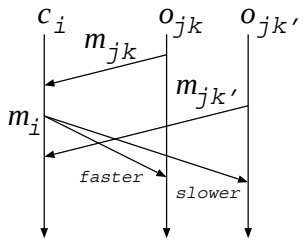

Fig. 3 Pseudo-active replication.

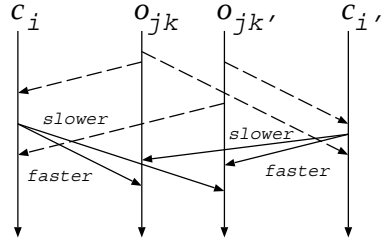

Fig. 4 Pseudo-active replication in a wide-area network.

sends a message to another one[12]. On receiving the first response message from the replicas, the client c_i delivers the result to the application before receiving all the response messages. Hence, the response time in c_i can be shorter and the synchronization overhead in $\mathcal{S}$ is reduced. However, since the replicas are placed on heterogeneous computers and are not synchronized, some replica $o_{jk'}$ might have performed all the request messages from clients and some requests are still left to wait to be performed in the queue in another replica $o_{jk''}$ because $o_{jk''}$ is placed on a slower computer. In this case, if $o_{jk'}$ fails, the recovery procedure takes longer time than the conventional active replication because $o_{jk''}$ has to perform the request messages which $o_{jk'}$ has already performed before the failure occurs as shown in the passive replication.

In order to solve this problem, we introduce the following two methods in the pseudo-active replication:

1. Each client c_i notifies each replica o_{jk} of which replica is faster or slower.
2. If a replica $o_{jk'}$ is notified to be slower, $o_{jk'}$ omits some request messages from clients in order to catch up with the faster replicas.

Suppose that a client c_i waits for response messages m_{jk} and $m_{jk'}$, and sends a request message m_i. In Refs. 8) and 14), we define faster/slower replicas based on the *causal relationship*[9] among these messages (**Fig. 3**).

[Definition: faster/slower replicas]
If $m_{jk} \rightarrow m_i$ and $m_{jk'} \not\rightarrow m_i$, where $m \rightarrow m'$ denotes a message m causally precedes another message m', o_{jk} *is followed by* $o_{jk'}$. Here, o_{jk} and $o_{jk'}$ are defined to be a faster and a slower replicas, respectively. □

2.4 Pseudo-active Replication in a Wide-area Network

In a wide-area network, server computers on which the replicas of an object o_j are placed may be connected with different sub-networks, e.g., one is in Japan and another is in Europe, for executing mission-critical applications more fault-tolerantly. In addition, clients are also distributed in a world-wide area. Clients cannot estimate the processing speed of a replica by measuring the response time in such a wide-area network environment. For example, all the replicas may be informed to be slower as shown in **Fig. 4**.

In Fig. 4, suppose a pair of a client c_i and a replica o_{jk} are nearer and another pair of $c_{i'}$ and $o_{jk'}$ are also nearer. Also suppose that the replicas o_{jk} and $o_{jk'}$ are farther. This means that it takes longer to send a message to c_i from o_{jk} than $o_{jk'}$. Here, suppose that o_{jk} and $o_{jk'}$ send the response messages m^i_{jk} and $m^i_{jk'}$ to both of the clients c_i and $c_{i'}$. c_i receives the response message m^i_{jk} and then sends a next request message to o_{jk} and $o_{jk'}$. On the other hand, $c_{i'}$ receives the response message $m^{i'}_{jk'}$ before $m^{i'}_{jk}$ because $o_{jk'}$ is nearer to c_i than o_{jk}. Then, $c_{i'}$ sends a next request message to o_{jk} and $o_{jk'}$ before receiving $m^{i'}_{jk'}$. Here, c_i considers that o_{jk} is faster than $o_{jk'}$ but $c_{i'}$ considers that $o_{jk'}$ is faster than o_{jk}. Hence, both o_{jk} and $o_{jk'}$ are informed to be slower and invoke the procedure to omit the waiting requests.

Let $R(c_i, o_{jk})$ denotes a response time from a replica o_{jk} to a client c_i. The difference $|R(c_i, o_{jk}) - R(c_i, o_{jk'})|$ is influenced by the difference between the processing speeds of o_{jk} and $o_{jk'}$ and the difference between the message transmission delays to c_i from o_{jk} and $o_{jk'}$. In addition, the network system usually consists of multiple clients distributed in a wide-area network. Hence, the processing speed observed based on the receipt order of the response messages for the previous request in a client is relative and does not show the difference of processing speed in the replicas. Therefore, it is not suitable for a pseudo-active replication in a

wide-area network.

The requests which can be delivered but are not yet delivered to the application are called *waiting* request messages. The waiting request messages are queued in an application queue (APQ). In the paper of Ref. 7), the length of APQ is used as a metric of the processing speed of each replica and is piggied back with a response message transmitted from the replica to a client. However, if multiple clients near to the faster replica send request messages burstly, the APQ of the faster replica might be longer than that of the slower replica temporally. In order to solve this problem, a sequence number SEQ is assigned to the request message performed most recently. SEQ is used to be a metric of the speed of the computer on which a replica is placed. Here, a request message is carried by using a total ordering protocol proposed in Ref. 3). SEQ is piggied back with the message. According to the following protocol, slower replicas are found. Suppose that a client c_i requests a server to perform a request $\mathcal{R}$ on a replica o_{jk} of an object o_j.

1. A client c_i sends a reservation message $Res(\mathcal{R})$ to every replica o_{jk} of o_j.
2. On receipt of $Res(\mathcal{R})$, o_{jk} sends back a confirmation message $Conf(\mathcal{R})$ to c_i with a sequence number $SEQ_{jk}(\mathcal{R})$.
3. After receiving all the $Conf(\mathcal{R})$ messages, c_i sends a request message $Req(\mathcal{R})$ to every o_{jk} with the maximum sequence number $MSEQ(\mathcal{R}) = \max_k SEQ_{jk}(\mathcal{R})$ assigned to the received $Conf(\mathcal{R})$ messages. $Req(\mathcal{R})$ message carries $\mathcal{R}$.
4. On receipt of $Req(\mathcal{R})$, each o_{jk} enqueues $\mathcal{R}$ to APQ_{jk}. In APQ_{jk}, $\mathcal{R}$ is sorted by the maximum sequence number $MSEQ(\mathcal{R})$.

Here, the sequence number $SEQ_{jk}(\mathcal{R})$ assigned to the request most recently performed in a replica o_{jk} is piggied back with the $Conf(\mathcal{R})$ message. By receiving $Conf(\mathcal{R})$ from all the replicas, the client c_i can find which replica is slower. Ideally, the client o_i^c receives $Conf(\mathcal{R})$ from all the replicas simultaneously. However, it is impossible in a wide-area network due to the difference of message transmission delays. Hence, we introduce a certain threshold value to find slower replicas. Only if the difference between the sequence numbers of some replica o_{jk} and another is larger than this threshold, o_{jk} is considered to be a slower replica.

In order for a slower replica $o_{jk'}$ to catch up with the faster one o_{jk}, $o_{jk'}$ omits some waiting request messages in the queue and does not perform them. Here, let $op \circ op'$ be a *concatenation* of a requested operations op and op'. Let $op(s)$ denote a state of an object after op is performed in a state s.

[Definition: an identity request]
An operation op is an *identity operation* iff $op(s) = s$ for every state s. □

[Definition: an idempotent request]
An operation op is an *idempotent operation* iff $op \circ op(s) = op(s)$ for every state s. □

Clearly, even if the slower replica $o_{jk'}$ omits identity and idempotent operations, $o_{jk'}$ can get the same state as the faster replica o_{jk}.

[Omission rule]
If all the following conditions are satisfied, an operation op in the waiting request stored in $APQ_{jk'}$ is omitted by a replica $o_{jk'}$:

1. $o_{jk'}$ is a slower replica.
2. op is an identity or idempotent operation.
3. Some faster replica o_{jk} has performed op.

□

In the papers of Refs. 8) and 14), by using *vector clocks*[10] for determining the causal relationship among the messages, the omission rules 1 and 3 are checked in each replica o_{jk} $(1 \leq k \leq n_j)$. Here, every request message is assumed to be transmitted to all the replicas in the same order, i.e., *totally ordered delivery* is assumed.

Requests not being omitted by the omission rule are performed in the same order in every replicas. However, some pair of operations op and op' can be performed in different orders.

[Definition: compatible and conflict operations]
Operations op and op' are *compatible* iff $op \circ op'(s) = op' \circ op(s)$ for every state s. Otherwise, these operations are *conflict*. □

If op and op' are compatible, these operations can be performed in different order in each replica.

By computing the operations in different order in each replica, the response time in client objects may be reduced (**Fig. 5**). If an operation op requested by c_i and another operation op' requested by $c_{i'}$ are compatible, op and op' are required to be computed first by the replica near c_i and $c_{i'}$, respectively. That is, the message transmission delay between a client objects and the replicas is reasonable for deciding the computation order of compatible operations. The message transmission delay is not constant but time-variant[15]. Therefore, it is required to be measured each time an oper-

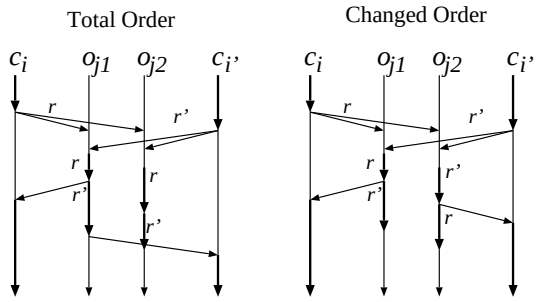

Fig. 5 Intentional computing order exchange.

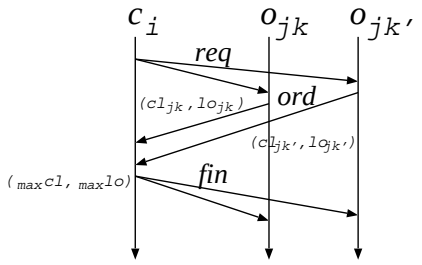

Fig. 6 Total ordering protocol for pseudo-active replication.

ation is requested. In our protocol proposed in the following section, the delay is measured in the first and the second phases of total ordering protocol. Finally, in order to avoid that the computation of some compatible operations is postponed infinitely, the order for some request to be performed is maximally exchanged with another one in a predetermined $E_{\max}$ times. That is, if an operation *op* is exchanged in $E_{\max}$ times, *op* becomes a conflict operation with every other request.

3. Protocol

In this section, we propose another protocol for implementing the pseudo-active replication by using the total ordering protocol[3]. Each replica o_{jk} $(1 \leq k \leq n_j)$ of a server o_j manipulates the following variables:

- Logical clock cl_{jk} for totally ordering the requests from client objects.
- Index loi_{jk} of a lastly computed request for the metric of processing speed of a replica o_{jk}.

In the following total ordering protocol, the above variables are piggied back with the control messages in order to exchange the length of the waiting request queue among the replicas (**Fig. 6**):

[**Total ordering protocol**]

1. A client object c_i sends request messages $req(r)$ with a request r to all the replicas o_{jk} $(1 \leq k \leq n_j)$.
2. On receipt of $req(r)$, the replica o_{jk} stores the request r in the buffer with cl_{jk}. o_{jk} sends back an ordering message $ord(cl_{jk}, loi_{jk})$ piggying back cl_{jk} and loi_{jk}. cl_{jk} is incremented by one.
3. After receiving all the ordering messages from the replicas, c_i sends final messages $fin(\max cl, \max loi, ord)$ where $\max cl = \max_k cl_{jk}$, $\max loi = \max_k cl_{jk}$ and ord is the receipt order of the ordering message from o_{jk}.
4. On receipt of $fin(\max cl, \max loi, ord)$, r is restored from the buffer and enqueued to APQ ordered by $oi(r) = \max cl$. □

APQ is an FIFO request queue and the application dequeues requests from APQ. If the application finishes the computation of a request r with $oi(r)$, loi_{jk} is updated to $oi(r)$. Hence, loi_{jk} is always incremented. $\max loi$ piggied back to the final message means that the fastest server object has finished to compute a request with $\max loi$. Hence, the procedure for omitting requests is invoked as follows:

[**Omitting operations**]

- If $\max loi - loi_{jk} > threshold$, identity and idempotent operations in APQ is removed. □

Finally, if a request r and another request r' in APQ are compatible, r is enqueued into APQ according to the following procedure:

[**Intentional order exchange procedure**]

1. If r and r' are compatible and $ord(r) < ord(r')$, r is enqueued before r'.
2. If r and r' are compatible and $ord(r) = ord(r')$, r is enqueued before r' with probability 1/2.
3. Otherwise, r is enqueued after r'. □

4. Evaluation

4.1 Evaluation Metrics

We evaluate our protocol described in by comparing with the protocol in Ref. 2). Here, we measure the following:

- Number of request messages in APQ.
- Differences among numbers of request messages in APQ.
- Processing time for our protocol.

We assume that there are two replicas o^s_{j1} and o^s_{j2} of a server o^s_j. Request messages from clients are assumed to be performed according to our protocol and queued into application message queues APQ_{j1} and APQ_{j2} by o^s_{j1} and

Table 1 Machine architectures.

Host	Machine	*CPU*	Memory
H_1	Sun Enterprise 450	300 MHz × 2	512 MB
H_2	Sun Ultra5	167 MHz × 2	256 MB

o^s_{j2}, respectively. The request messages are dequeued from the APQ sequentially. The fewer messages are waiting in APQ_{js}, the less memory the protocol uses. In our evaluation, the numbers of messages in APQ of the faster replicas in both protocols are equal. Hence, we measure the number Aq_j of messages in APQ of the slower replica for the evaluation.

As discussed in Section 2, if a faster replica o^s_{j1} fails, one of the slower replicas o^s_{j2} takes over after catching up with o^s_{j1}. Thus, recovery time is determined by the difference qd_j of the numbers of request messages queued in APQ_{j1} and APQ_{j2}, i.e., $qd_j = |APQ_{j1} - APQ_{j2}|$. The smaller qd_j is, the shorter the recovery time is required in case of a failure.

Finally, we measure the processing time for the protocol, i.e., the protocol overhead. This protocol overhead includes the time required for totally ordering the request messages, recognition of processing speeds of the computers on which the replicas are located, searching APQ for detecting omissible request messages, and removing omissible messages from APQ. In a faster replica, there is no difference between the processing times of our protocol and conventional one. Here, we measure the protocol processing time CU_j in a slower replica for our evaluation. The shorter CU_j is, the less CPU usage is required for protocol processing.

As stated above, we evaluate the following three items.

- Recovery time
- Memory usage
- CPU usage

4.2 Assumptions

We evaluate the performance of the proposed protocol under the following assumptions:

- There are two multiprocessor computers H_1 and H_2 interconnected by 100 Mbps Ethernet (**Table 1**).
- There are two replicas o^s_{j1} and o^s_{j2} of a server object o^s_{jk} and 20 client objects which request two kinds of operations op_1 and op_2. These objects are located as shown in **Table 2**.

In order to simulate various kinds of wide-area network environments, we use the following four parameters as shown in **Fig. 7**: T_c [sec], $T_{w_{jk}}$ [sec], $T_{d_{jk}}$ [sec], P_1 [%]. A parameter T_c denotes the interval between receipt of a response message and transmission of the next request message in a client object. Here, all requests are invoked synchronously, i.e., if a client object o^c_i invokes a request req^i_l, the next request req^i_{l+1} is never invoked before receiving a response of req^i_l. In our evaluation, each client object o^c_i issues 4 requests. Thus, each replica o^s_{jk} computes 80 requests. $T_{w_{jk}}$ denotes the processing time for one requested operation in a replica o^s_{jk}. $T_{w_{j1}}$ and $T_{w_{j2}}$ are assumed to be 1.03 [sec] and 14.78 [sec], respectively. $T_{d_{jk}}$ denotes the message transmission delay in the network. In our evaluation, we assume five clients are located near o^s_{j1} and the other five client objects are located near o^s_{j2}. The message transmission delay for near and far replicas are assumed to be 0.06 [sec] and 0.30 [sec], respectively. Each client object requests operations op_1 and op_2 with probabilities P_1 and P_2 ($P_1 + P_2 = 100$ [%]). Finally, a conflicting relation among the operations is defined as follows:

- op_1 and op_1 are conflict.
- op_2 and op_2 are compatible.
- op_1 and op_2 are conflict.

For example, op_1 and op_2 are *write* and *read* operations for an integer object. In the following subsection, we will show our evaluation result. Here, only T_c and P_1 are changed as shown in **Table 3**.

Table 2 Objects allocation.

Host	server	client
H_1	o^s_{j1}	10 clients
H_2	o^s_{j1}	10 clients

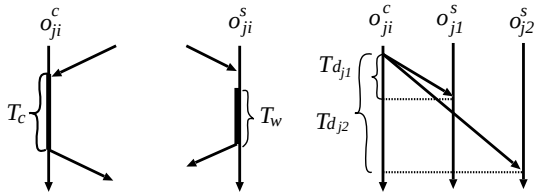

Fig. 7 parameters.

Table 3 Parameter values of T_c and P_1.

T_c [sec]	0.00, 2.00, 4.00, 6.00, 8.00, 10.00
P_1 [%]	0, 25, 50, 75, 100

4.3 Evaluation Result

This subsection describes the evaluation results under the environment discussed in Section 4.1. First, we measure the change of qd_j.

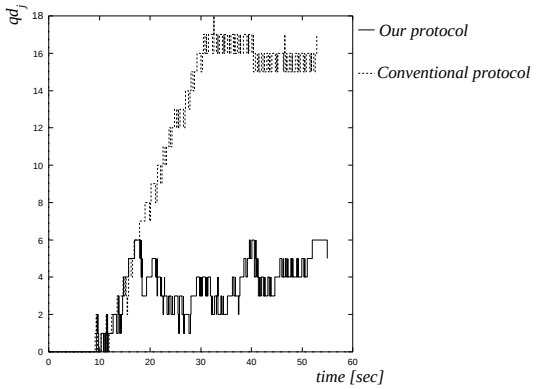

Fig. 8 qd_j (T_c =8.00 and P_1=25).

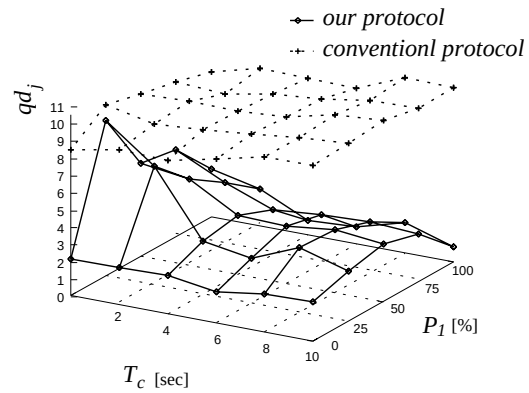

Fig. 9 Average recovery overhead (dq_j).

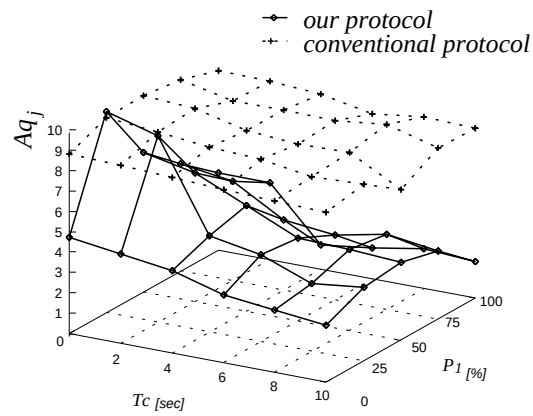

Fig. 10 The average overhead (Aq_j).

In **Fig. 8**, a solid line and a dotted one represent differences of numbers of messages in APQs, i.e., qd_j, in our protocol and a conventional one, where $T_c = 8.00$ [sec] and $P_1 = 25$ [%]. Here, qd_j in our protocol is always smaller than in a conventional one. This means that the proposed protocol implies shorter recovery time in case of a failure of a faster replica. In addition, the first time when a catch up procedure is invoked is 18 [sec] in the protocol and 30 [sec] in conventional one, even though the same threshold value is used in these protocols. This is because the protocol can detect slower replicas by using every request message and a conventional one can only detect them by using causally related messages. That is, there are more chances to detect them in the protocol.

Next, we measure averages of qd_j where T_c and P_1 are changed as shown in Table 3. In **Fig. 9**, a solid line and a dotted one represent average differences of numbers of messages in APQs, i.e., qd_j, in our protocol and a conventional one. In every evaluation environment, qd_j in our protocol is smaller than in a conventional one. That is, less recovery time is required in our protocol. Moreover, consider the case where $P_1 = 0$ [%] (P_2=100 [%]). Here, all the requested operations are op_2. That is, all the queued requests in APQ of the slower replica are omitted in both protocols. Therefore, the result shows there are more chances to detect slower replicas in our protocol.

Figure 10 shows the measurement result of the number Aq_j of waiting messages in APQ_{jk} in a slower replica o^s_{jk}. Aq_{jk} in our protocol is much smaller than a conventional protocol except for the two parameter values T_c = 0.00 [sec], P_1 = 25 [%] and T_c = 2.00 [sec], P_1 = 25 [%]. The waiting request messages queued in APQ_{jk} are stored in the memory. Hence, the amount of memory used for thses messages are proportional to the number of messages in APQ_{jk}, i.e., Aq_{jk}. Since Aq_{jk} in the proposed protocol is much less than that in a conventional one, much less memory is reduced by using the proposed protocol.

Finally, we measure the protocol processing time CU_j in a slower replica of a server object o^s_j. CU_j includes the time for detecting a slower replica, searching and removing omissible messages from an APQ. **Table 4** shows the result of the measurement for 80 requests comparing with the conventional protocol. Here, our protocol is realized with almost the same processing time i.e., the same CPU usage, as the conventional one. In our protocol, there are more chances for detecting slower replicas. Thus, the procedure for searching omissible messages from the APQ is invoked more frequently. However, as shown before, the APQ in our protocol is shorter. Therefore, the additional CPU usage is sufficiently reduced.

Table 4 CPU usage (CU_j [sec]).

	avg	*max*	*min*
ours	2.97	3.38	2.70
conventional	2.90	3.28	2.60

5. Concluding Remarks

In order to apply the pseudo-active replication in a wide-area and large-scale network systems, we proposed another protocol designed by modifying the total ordering protocol. In order to make clear the efficiency of our protocol, we have implement our protocol in a prototype system simulating a wide-area network and shown that our protocol realizes the pseudo-active replication with less overhead. In future, we will evaluate our protocol in the Internet environment.

References

1) Ahamad, M., Dasgupta, P., LeBlanc R. and Wilkes, C.: Fault Tolerant Computing in Object Based Distributed Operating Systems, *Proc. 6th IEEE Symposium on Reliable Distributed Systems*, pp.115–125 (1987).
2) Barrett, P.A., Hilborne, A.M., Bond, P.G. and Seaton, D.T.: The Delta-4 Extra Performance Architecture, *Proc. 20th International Symposium on Fault-Tolerant Computing Systems*, pp.481–488 (1990).
3) Birman, K.P. and Joseph, T.A.: Reliable Communication in the Presence of Failures, *ACM Trans. Computer Systems*, Vol.5, No.1, pp.47–76 (1987).
4) Borg, A., Baumbach, J. and Glazer, S.: A Message System Supporting Fault Tolerance, *Proc. 9th ACM Symposium on OS Principles*, pp.27–39 (1983).
5) Cooper, E.C.: Reliable Distributed Programs, *Proc. 10th ACM Symposium on OS Principles*, pp.63–78 (1985).
6) Higaki, H. and Soneoka, T.: Group-to-Group Communications for Fault-Tolerance in Distributed Systems, *IEICE Trans. Information and Systems*, Vol.E76-D, No.11, pp.1348–1357 (1993).
7) Higaki, H., Morishita, N. and Takizawa, M.: Active Replication in Wide-Area Networks, IPSJ Technical Report, Vol.98, No.84, pp.93–98 (1998).
8) Ishida, T., Higaki, H. and Takizawa, M.: Pseudo-Active Replication of Objects in Heterogeneous Processors, IPSJ Technical Report, Vol.98, No.15, pp.67–72 (1998).
9) Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System, *Comm. ACM*, Vol.21, No.7, pp.558–565 (1978).
10) Mattern, F.: Virtual Time and Global States of Distributed Systems, *Parallel and Distributed Algorithms*, pp.215–226, North-Holland (1989).
11) Powell, D., Chereque, M. and Drackley, D.: Fault-Tolerance in Delta-4, *ACM Operating System Review*, Vol.25, No.2, pp.122–125 (1991).
12) Schneider, F.: Byzantine Generals in Action: Implementing Fail-Stop Processors, *ACM Trans. Computing Systems*, Vol.2, No.2, pp.145–154 (1984).
13) Shima, K., Higaki, H. and Takizawa, M.: Fault-Tolerant Intra-Group Communication, *IPSJ Trans.*, Vol.37, No.5, pp.883–890 (1996).
14) Shima, K., Higaki, H. and Takizawa, M.: Pseudo-Active Replication in Heterogeneous Clusters, *IPSJ Trans.*, Vol.39, No.2, pp.379–387 (1998).
15) Tachikawa, T., Higaki, H., Takizawa, M., Liu, M., Gerla, M. and Deen, M.: Flexible Wide-area Group Communication Protocols – International Experiments, *Proc. 27th International Conference on Parallel Processing*, pp.570–577 (1998).

(Received May 11, 1999)
(Accepted October 7, 1999)

Hiroaki Higaki was born in 1967. He received his B.E. degree from the Department of Mathematical Engineering and Information Physics, the University of Tokyo in 1990. From 1990 to 1996, he was in NTT (Nippon Telegraph and Telephone Corporation) Software Laboratories. Since 1996, he is in the Department of Computers and Systems Engineering, Tokyo Denki University. He received his D.E. degree from the Department of Computers and Systems Engineering, Tokyo Denki University in 1997. His research interests include distributed systems, distributed algorithms, distributed operating systems, fault-tolerant systems and computer network protocols. He received IPSJ Convention Award and IPSJ SIG Research Award in 1995 and 1997, respectively. He is a member of ACM, IEEE CS, IPSJ and IEICE.

Katsuya Tanaka was born in 1971. He received his B.E. and M.E. degrees in Computers and Systems Engineering from Tokyo Denki University, Japan in 1995 and 1997, respectively. From 1997 to 1999, he worked for NTT Data Corporation. Currently, he is an assistant in the Department of Computers and Systems Engineering, Tokyo Denki University. His research interests include distributed systems, transaction management, recovery protocols, and computer network protocols.

Makoto Takizawa was born in 1950. He received his B.E. and M.E. degrees in Applied Physics from Tohoku University, Japan, in 1973 and 1975, respectively. He received his D.E. degree in Computer Science from Tohoku University in 1983. From 1975 to 1986, he worked for Japan Information Processing Developing Center (JIPDEC) supported by the MITI. He is currently a Professor of the Department of Computers and Systems Engineering, Tokyo Denki University since 1986. From 1989 to 1990, he was a visiting professor of the GMD-IPSI, Germany. He is also a regular visiting professor of Keele University, England since 1990. He was a vice-chair of IEEE ICDCS, 1994, and is serving as a program cochair of IEEE ICDCS, 1998 and serves on the program committees of many international conferences. His research interests include communication protocols, group communication, distributed database systems, transaction management, and groupware. He is a member of IEEE, ACM, IPSJ, and IEICE.