マルチメディア通信と分散処理 95－8
（1999. 11. 19）

Response Time Reduction in Pseudo-Active Replication

Naokazu Nemoto, Hiroaki Higaki, Katsuya Tanaka, and Makoto Takizawa
Dept. of Computers and Systems Engineering
Tokyo Denki University
{nemoto, hig, katsu, taki}@takilab.k.dendai.ac.jp

Distributed applications are realized by cooperation of multiple objects based on client-server style communication. Server objects are replicated on multiple computers for achieving fault-tolerance. In the conventional active replication, all the replicated server objects (replicas) receive the same requests in the same order from client objects, invoke the same operations (methods), and send back responses. These replicas might be placed on different kinds of computers with different processing speeds. In addition, these computers might be connected to different networks, that is, replicas might be distributed in WAN (wide area network), e.g the Internet. For applying the active replication to such a heterogeneous environment, this paper proposes a pseudo-active replication where a client object receives only the first response from the replicas. In order to reduce the recovery time due to the difference of processing speeds among the replicas, two techniques are introduced. One is to detect the fastest replica and the other is for the other replicas, i.e. slower replicas, to catch up with the fastest one. Here, requests for identity and idempotent operations (methods) are not invoked in the slower replicas. Furthermore, in order to reduce the response time for requests from client objects, requests for compatible operations (methods) are invoked in different order in the replicas. The order is decided based on the round trip time (RTT) between client objects and replicas for supporting WAN environments. These are realized by piggying back some additional information with the control messages for the totally ordering protocol. That is, no additional message is required in the proposed protocol.

擬似能動的多重化におけるレスポンスタイムの短縮方法

根本 直一　桧垣 博章　田中 勝也　滝沢 誠
{nemoto, hig, katsu, taki}@takilab.k.dendai.ac.jp
東京電機大学 理工学部 情報システム工学科

分散アプリケーションは、複数のオブジェクトの協調によって実現されている。ここでは、クライアントサーバ型の通信が用いられることが多い。フォールトトレランスを実現するために、サーバオブジェクトを複製し、複数の異なるコンピュータに配置する方法が用いられている。多重化の方法のうち、複製されたサーバオブジェクト(レプリカ)がすべて動作する能動的多重化法では、各レプリカがクライアントオブジェクトから送信された要求メッセージを同一順序で処理し、応答を返す。この方法は、すべてのレプリカが同じ性能のコンピュータに配置され、これらのコンピュータが同一のネットワークに接続されているような均質な環境ではうまく機能する。しかし、現在のネットワーク環境は、プロセッサの処理速度が異なるコンピュータが、広域のネットワークに接続される非均質な環境である。このような環境下では、レプリカは性能の異なるプロセッサを持つコンピュータに配置され、これらのコンピュータが異なるネットワークに接続されることが通常である。その結果、クライアントへの応答時間はレプリカごとに異なることになる。能動的多重化では、クライアントオブジェクトがすべてのレプリカから応答メッセージを受け取ってはじめてその応答を受理することができる。これによって、レプリカの同期が実現し、レプリカに障害が発生した場合でも回復のための時間オーバヘッドをほとんど要しないという利点が得られている。ところが、クライアントオブジェクトへの応答時間は、最後に受信される応答メッセージの到着時間によって決定されることから、レスポンスタイムが大きくなるという問題がある。本論文では、クライアントオブジェクトが最初の応答を受けた時点でそれを受理することによってレスポンスタイムを短縮する擬似能動的多重化を提案する。擬似能動的多重化の非同期性によって生じる回復時間の増大については、クライアントオブジェクトからの要求を処理しながら、1) 速いレプリカと遅いレプリカを判別し、2) 遅いレプリカは実行する必要のない要求を処理しないことにより、速いレプリカに追い着く機構を導入することによって解決する。また、要求を処理した結果がその処理順序に依存しないものについては、各レプリカができるだけ異なる順序で処理することにより、レスポンスタイムの短縮を図る。各レプリカは処理順序を決定するためにメッセージをやりとりすることはせず、クライアントオブジェクトとの間の伝達遅延を測定し、より近いクライアントオブジェクトからの要求を先に処理する方法を用いる。本論文の提案するプロトコルは、全順序プロトコルの制御メッセージにいくつかの情報を加えるだけで実現することができ、メッセージ数を増やすことなく実現される。

1 Introduction

According to the advance of computer and network technologies, network applications are widely developed. These applications are realized by multiple *objects*. An object is defined as a collection of data and operations (methods) to manipulate the data. For example, C++ and Java are used for programming objects, and CORBA and Java RMI are getting to be standards of a platform of distributed objects The objects cooperate by exchanging messages with each other. In this paper, the communication in the system is assumed to be in the client-server style. A client object sends a request message to a server object, the server object invokes a requested operation and sends back a response message to the client object. Here, mission critical applications are also implemented and these applications are required to be executed fault-tolerantly.

There are two kinds of techniques. One is *replication* and the other is *checkpoint-recovery*. In the replication technique, a server object is replicated and these replicas are placed on multiple computers. Even if some of the replicas fail, the others continue to process the requests from client objects. On the other hand, in the checkpoint-recovery technique, each object takes a checkpoint by storing the local state information in the stable storage during the execution of an application. If a certain object fails, the recovery is realized by restarting the objects from the checkpoints. This paper discusses a novel method of the replication.

An *active replication* has been proposed where all the replicated objects (replicas) are operational. Here, a client object sends request messages to all the replicas. Since all replicas have to send back the same response to the client and be in the same state after processing the required operation, the same operations are invoked in the same order by all the replicas. Thus, the request messages are transmitted by using a totally ordering protocol. After receiving the response messages from all the replicas, the client object accepts the response and continues to execute an application. That is, replicas are synchronized each time a response is accepted by a client object. An active replication works well in a homogeneous environment where the processing speeds of the replicas are the same and the message transmission delays between a client object and the replicas are also the same. However, there are different kinds of computers in the recent network environment, each replica may be placed on different kinds of computers. That is, computation is realized by different kinds of processors with different processing speed and different reliability. Hence, the response times of the replicas are different and a client object accepts the response when it receives the last response message from the replicas. That is, much synchronization overhead is required. In order to solve this problem, this paper proposes a *pseudo-active replication*. Here, a client object only waits for the first response from the replicas. That is, on receipt of the first response message, the client object accepts it and continues to execute the application. The other response messages from the slow replicas are discarded. Here, less synchronization overhead is required and the response time for a request is reduced.

Since the processing speeds of the replicas are different and the replicas are not synchronized, it takes longer recovery time after the failure of the fastest replica than the active replication. Because in order to catch up with the the failed fastest replica s_i, the operational replicas have to process requests that s_i processed before the failure. In order to reduce the recovery time, the replicas find the fastest replica, and if a replica is not the fastest one, it invokes a procedure to catch up with the fastest one. For detecting the fastest replica, sequence numbers assigned to the most recently processed requests by all the replicas are used. A slower replica removes requests waiting to invoke an *identity* or *idempotent* operation. In addition, if two requests are for *compatible* operations, replicas process these requests in different orders for reducing the response time. In the proposed protocol, round trip times between a client object and the replicas are used as the metric for deciding the processing order. The sequence numbers and the round trip times are piggied back to the control messages of the totally ordering protocol. Hence, no additional message is required.

The rest of this paper is organized as follows: In section 2, we review the conventional replication including the active and passive replications. In section 3, a pseudo-active replication is proposed. Here, the method to detect the fastest replica and the procedure for the slower replicas to catch up with the fastest one is presented. In section 4, the protocol of the pseudo-active replication with high performance is designed.

2 Conventional Replications

2.1 System Model

In most of the recent distributed applications, the objects in a network system are classified into *client* objects and *server* ones. A client object o_i^c requests a server object o_j^s to invoke a specified operation *op* by sending a request message. On receipt of the request message, o_j^s manipulates its data through *op* and responds to o_i^c by sending a response message. This type of communication among the objects is called *client-server* style. In this paper, the communication among the objects is assumed to be the client-server style. In order that the application programs are executed fault-tolerantly in $\mathcal{S}$, each server object o_j^s is replicated and located on multiple computers. Here, *replicas* o_{jk}^s $(1 \leq k \leq n_j)$ of o_j^s are composed of the same data and the same operations.

2.2 Passive and Active Replication

There are two main approaches for the replication techniques: a *passive replication* and an *active replication*. In the passive replication, only one replica o_{j1}^s is operational. A client object only sends a request message to o_{j1}^s. Then, only o_{j1}^s invokes a requested operation, changes its state, and sends back a response message to the client object. Another replicas o_{jk}^s $(2 \leq k \leq n)$ are not operational. The state of the passive replicas are sometimes updated by receiving the newest state information from the operational replica o_{j1}^s. This is called a *checkpoint*. If o_{j1}^s fails, one of the passive replicas, say o_{j2}^s, takes over it. o_{j2}^s starts to execute an application from the most recent checkpoint. Here, the recovery procedure takes time since o_{j2}^s becomes operational at the checkpoint and re-invokes the operations that o_{j1}^s has already finished in order to catch up with the failed o_{j1}^s.

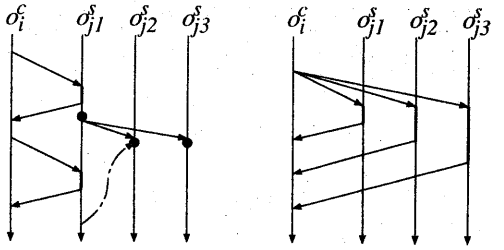

Figure 1: Passive and active replication.

In the *active replication* [1,4,10], all the replicas are operational. A client object o_i^c sends copies of a request message for an operation *op* to all the replicas o_{jk}^s $(1 \leq k \leq n_j)$. In order to keep the states of the replicas same, requested operations are invoked in the same order by all the replicas. It is realized by using a *totally ordering protocol*. Every o_{jk}^s invokes *op* and sends back a response message to o_i^c. After receiving all the response messages, o_i^c accepts the response and delivers it to the application. That is, the replicas o_{jk}^s are synchronized each time a client object accepts a response. Since all the replicas are operational and synchronized, even if a certain replica $o_{jk'}^s$ fails, the other replicas o_{jk}^s $(k \neq k')$ can continue to execute the application without the suspension time. Hence, the recovery procedure in the active replication requires less overhead than that in the passive one.

3 Pseudo-Active Replication

In the conventional active replication, all the replicas o_{jk}^s $(1 \leq k \leq n_j)$ of a server object o_j^s are synchronized each time a client object o_i^c accepts a response. It works well in a homogeneous environment where the computers on which o_{jk}^s are located are assumed to be the same kind ones with the same processing speed and the same reliability and to be connected to the same local area network since o_{jk}^s are easily synchronized. That is, it takes the same time to finish the requested operation and the same transmission delay is required for the messages between a client object and the replicas. Therefore, o_i^c can receive all the response messages from o_{jk}^s at almost the same time. This assumption is reasonable in a local-area network.

However, a wide-area network, e.g. the Internet, is usually *heterogeneous*. Various kinds of computers are connected to various kinds of networks. That is, there are processors with different processing speed, reliability and availability, and there are networks with different transmission delay, bandwidth and message loss ratio. Hence, the response times from the replicas is different. In the active replication, only when a client receives all the responses from the replicas, it accepts the response and delivers it to the application. Therefore, the response time for the application is dominated by the last response message, that is, the client has to wait after it receives the first response message. Especially in the wide-are network, the time overhead for the synchronization becomes a serious problem.

In order to solve this problem, the authors have proposed a *pseudo-active* replication [5–7, 11]. Here, a client object o_i^c only waits for the first response message from the replicas $o_{j1}^s, \ldots, o_{jn_j}^s$. On receipt of the first response message, o_i^c accepts and delivers it to the application and continues to execute the application. o_i^c only discards the other response messages. Thus, the response time for a request in a client object is reduced.

However, since o_{jk}^s are placed on processors with various processing speeds and are not synchronized, it may occur that a certain replica $o_{jk'}^s$ finishes all the operations requested by the client objects and the other replicas still hold some requests where the requested operations are not yet invoked since $o_{jk'}^s$ is placed on a fast processor and the others are placed on slower processors. Now, if $o_{jk'}^s$ fails, the recovery procedure takes time since the other replicas have to finish the operations that $o_{jk'}^s$ has already finished before the failure. If these operations are finished, the operational replicas catch up with the failed fast replica and receives requests that have not yet finished the requested operations. This time overhead for recovery is similar to that in the passive replication. In order to reduce the recover time, the authors introduce the following two mechanism in the proposed protocol:

- A client object tells the replicas which replica is the fastest. This is realized without the communication among the replicas.
- If a replica $o_{jk'}^s$ finds to be slower, it omits some request without invoking the requested operation in order to catch up with the fastest replica.

In the papers [7, 11], the authors proposed the method to find the fastest replica based on the *causal relationship* between a response message for a request and the successive request message. Here, the faster replica is defined as follows:

[Faster and slower replicas (Conventional)]
Suppose a client object o_i^c receives response messages res_k from a replica o_{jk}^s and sends a successive request message *req*. If $res_k \rightarrow req$, o_{jk}^s is a faster replica. Otherwise, o_{jk}^s is a slower one. □

Now, *identity* and *idempotent* operations are defined as follows:

[Identity operation]
Let $op(s)$ denote a state of a replica o_{jk}^s after an operation *op* is invoked in o_{jk}^s at a state *s*. If $op(s) = s$, *op* is an *identity* operation. □

[Idempotent operation]
Let $op \circ op'$ denote a composition of two operations *op* and *op'* and mean that *op'* is invoked after *op* is finished. Hence, $op \circ op'(s)$ denotes a state of replica o_{jk}^s just after *op* and *op'* are invoked in this order in o_{jk}^s at a state *s*. In addition, let $[op(s)]$ denote a response sent by a replica o_{jk}^s when an operation *op* is invoked at a state *s* and is finished. If $op \circ op'(s) = op'(s)$ and $op'(op(s)) = op'(s)$, *op* is an *idempotent* operation. □

In order for a slower replica o_{js}^s to catch up with

a faster one o^s_{jf}, o^s_{js} omits some requests without invoking operations. Even if o^s_{js} omits some requests, a state of o^s_{js} after finishing each operation should be the same as that of o^s_{jf}, and a response of o^s_{js} should also be the same as that of o^s_{jf}. Hence, the following requests can be omitted.

[Omissible request]
If an operation *op* requested by a request *req* is an identity or an idempotent operation, *op* is omissible in a slower replica [7].

[Omission rule]
If the following conditions are satisfied, an operation *op* is omitted in a replica o^s_{jk}.

1. o^s_{jk} is a slower replica.
2. *op* is an omissible, i.e. *op* is an identity or an idempotent operation.
3. Some $o^s_{jk'}$ $(k' \neq k)$ has already finished *op*.

In [7] and [11], by using vector clocks [9], the above rules 1 and 3 are checked in o^s_{jk}. In addition, every request message is assumed to be transmitted to all the replicas in the same order, i.e. *totally ordering protocol* is assumed to work in the lower protocol layer. In the protocols discussed in [7] and [11], the requests are also accepted in the total order and the requested operations are invoked in the same order in an application of every replica. However, a certain pair of operations *op* and *op'* can be invoked in different order in the replicas.

[Compatible and conflict operations]
Let *op* and *op'* are operations required to be invoked at a state *s* in an object *o*. These operations are *compatible* iff $op \circ op'(s) = op' \circ op(s)$, $[op(s)] = [op(op'(s))]$, and $[op'(op(s))] = [op'(s)]$ for every state *s* of *o*. Otherwise, these operations are *conflict*. □

If *op* and *op'* are compatible, these operations can be invoked in different order in each replica. Even if two replicas o^s_{jk} and $o^s_{jk'}$ invokes *op* and *op'* in different order, both of them send back the same responses to the clients and get the same state when the operations are finished. By invoking the compatible operations in different order in each replica, less response times it takes than if the operations are invoked in the same order in all the replicas. For example in Figure 3, compatible operations *op* and *op'* are requested to be invoked in replicas o^s_{jk} and $o^s jk'$ by client objects o^c_i and $o^c_{i'}$, respectively. Here, according to the totally ordering protocol, *op* and *op'* are invoked in this order in both o^s_{jk} and $o^s_{jk'}$. The response times of *op* and *op'* are T and T', respectively. On the other hand in Figure ??, o^s_{jk} invokes *op* before *op'* and $o^s_{jk'}$ invokes *op* after *op'*, i.e. o^s_{jk} and $o^s_{jk'}$ invoke *op* and *op'* in different orders. However, since *op* and *op'* are compatible, the responses sent by o^s_{jk} and $o^s_{jk'}$ are the same and the state o^s_{jk} and $o^s_{jk'}$ are also the same. In addition, the response times of *op* and *op'* are T and $T'' < T'$, respectively. Hence, less response time is required. This intentional change of invocation order of requests is seemed to work well in a wide-area network. If replicas and client objects are widely distributed, the difference of message transmission delays gets large. Hence, if each replica gives higher priority to the operations requested by clients located near the replica, the response times get shorter.

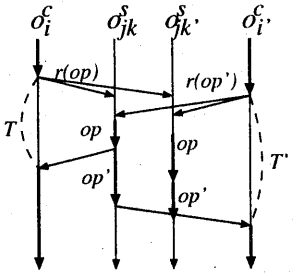

Figure 2: Response time in total order.

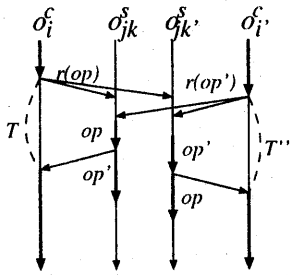

Figure 3: Response time reduction.

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 [12]. Therefore, it is required to be measured each time an operation is requested. In our protocol proposed in the next section, it is measured in the first and the second phase of total ordering protocol. Finally, in order to avoid that the computation of some compatible operation is postponed infinitely, the maximum number E_{max} of order exchange is predetermined. If the order of an operation *op* is exchanged E_{max} times, *op* becomes a conflict operation with any other requests.

4 Protocol

In this section, a protocol for the pseudo-active replication protocol is discussed. This protocol is based on the three phase totally ordering protocol. Two sequence numbers and one time value are piggied back to control messages of the totally ordering protocol. One sequence number is used for the totally ordering delivery of messages and the other sequence number is for finding faster and slower replicas. The time value is a round trip time (RTT) between a client object and a replica. This is used for a metric of intentional change of invocation order of compatible messages.

4.1 Totally ordering protocol

This subsection shows a totally ordering protocol which is a base of the proposed pseudo-active replication protocol. Here, suppose that

each replica o^s_{jk} $(1 \leq k \leq n_j)$ of o^s_j manipulates the following variables.

- The last sequence number $lseq_{jk}$ that is assigned to the most recently received request message from a client object.
- A request buffer $RBUF_{jk}$ which holds requests whose invocation order has not yet decided.
- An application queue APQ_{jk} which holds requests whose invocation order is decided but that has not yet invoked. An application takes a request out of APQ_{jk} and invokes an operation whose identifier is carried by the request.

Each control message m transmitted for the totally ordering protocol [3] carries the following variables.

- A request sequence number seq_m.
- A replica identifier rID_m.

Each replica is assumed to be assigned an identifier which is comparable with each other. An operator for comparing two identifies is '<'.

[Totally ordering protocol]

1. A client object o^c_i sends copies of a reservation message m^r which carries a request $r(op)$ of an operation op to all the replicas o^s_{jk} $(1 \leq k \leq n_j)$.
2. On receipt of m^r, o^s_{jk} increments $lseq_{jk}$ by one and stores $r(op)$ into $RBUF_{jk}$ with a temporal sequence number $lseq_{jk}$. Then, o^s_{jk} sends back a confirmation message m^c where $seq_{m^c} = lseq_{jk}$ and rID_{m^c} is an identifier of o^s_{jk}.
3. After receiving m^c from every replica o^s_{jk} $(1 \leq k \leq n_j)$, o^c_i sends copies of a final message m^f where $seq_{m^f} = \max_k seq_{m^c_k}$ where m^c_k is a confirmation message from o^s_{jk} and $rID_{m^f} = rID_{m^c}$.
4. On receipt of m^f, o^s_{jk} takes $r(op)$ out of $RBUF_{jk}$ and assigns a final sequence number seq_{m^f} to $r(op)$. $r(op)$ is sorted in APQ_{jk} in the order of the final sequence numbers. If there are multiple requests with the same final sequence number, these requests are sorted by using rID.

4.2 Metric of Processing Speed

In the pseudo-active replication, if a replica finds that it is slower, it invokes a procedure to catch up with faster ones. For this purpose, in [7] and [11], causal relationship between a response message for a request and the successive request message is used. It can support replicas which are located on computers with different processing speeds only if these computers are connected to the same network. In this case, since the message transmission delay from the replicas to the client object is almost the same, the causal relationship represents the processing speeds of the computers.

However, if the replicas are located on computers connected to different networks, that is, the replicas are distributed in a wide-area network, the causal relationship represents the difference of sum of the processing speed and the message transmission delay. Hence, it is not a good metric for invocation of the catch-up procedure.

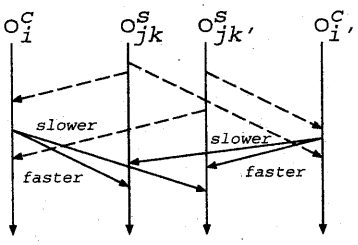

Figure 4: Metric for processing speed in WAN.??

For example, in Figure ??, all the replicas may find it is slower. Consider that a client object o^c_i and a replica o^s_{jk} are in Japan and another client object $o^c_{i'}$ and another replica $o^s_{jk'}$ are in Europe. For simplicity, o^s_{jk} and $o^s_{jk'}$ are located on computers with the same processing speed. Here, o^c_i sends a request after receiving a response for the previous request from o^s_{jk} and before receiving it from $o^s_{jk'}$. On the other hand, $o^c_{i'}$ sends a request after receiving a response for the previous request from $o^s_{jk'}$ and before receiving it from o^s_{jk}. Thus, both o^s_{jk} and $o^s_{jk'}$ find that it is slower on receipt of the request messages from $o^c_{i'}$ and o^c_i, respectively.

In order to solve this problem, a metric of processing speed of each replica should be used for invocation of the catch-up procedure. Here, the proposed protocol uses the sequence number $fseq_{jk}$ of the request that has been most recently finished in a replica o^s_{jk}. At the second step of the totally ordering protocol, o^s_{jk} sends back a confirmation message m^c to a client object o^c_i. Here, $fseq_{jk}$ is piggied back to m^c. Then, on receipt of confirmation messages from all the replicas, o^c_i computes $mfseq = \max_k fseq_{jk}$ and sends a final message m^f with $mfseq$. On receipt of the m^f, o^s_{jk} compares $fseq_{jk}$ with $mfseq$. If $fseq_{jk} = mfseq$, o^s_{jk} is a faster replica. Otherwise o^s_{jk} is a slower one. In order to get a tradeoff between the efficiency and overhead of the catch-up procedure, the following rule is applied:

[Invocation of catch-up procedure]
If $mfseq - fseq_{jk} > L$ where L is predetermined threshold, o^s_{jk} invokes a catch-up procedure. □

In the catch-up procure, o^s_{jk} searches identity and idempotent requests in APQ_{jk} from the head to the tail and takes them out of APQ_{jk}.

4.3 Metric of transmission delay

As discussed in Section 3, a pair of compatible requests can be invoked in any order in each replica. In order to reduce the response times of requests, each replica gives a higher priority to requests issued by client objects located nearer to the replica. Hence, the message transmission delay between client objects and a replica can be used for a metric for deciding the processing order of the compatible requests. In the proposed protocol, a round trip time (RTT) between a client object o^c_i and replicas o^s_{jk} are measured in o^c_i. That

is, o_i^c measures the time T since it sends a reservation message to a replica o_{jk}^s till it receives a confirmation message from o_{jk}^s. T is an RTT between o_i^c and o_{jk}^s. T is piggied back to a final message to o_{jk}^s. If there is a pair of compatible requests $r(op)$ and $r(op')$ in an application queue APQ_{jk} of o_{jk}^s, o_{jk}^s compares the RTTs piggied back to these requests. The request with a shorter RTT is ordered before the other in APQ_{jk}.

Finally, the pseudo-active replication protocol is as follows.

[Pseudo-Active Replication Protocol]

1. A client object o_i^c sends copies of a reservation message m^r which carries a request $r(op)$ of an operation op to all the replicas o_{jk}^s $(1 \leq k \leq n_j)$. o_i^c starts a timer to measure RTTs T_{ijk}.
2. On receipt of m^r, o_{jk}^s increments $lseq_{jk}$ by one and stores $r(op)$ into $RBUF_{jk}$ with a temporal sequence number $lseq_{jk}$. Then, o_{jk}^s sends back a confirmation message m^c where $seq_{m^c} = lseq_{jk}$ and rID_{m^c} is an identifier of o_{jk}^s. In addition m^c carries $fseq_{jk}$.
3. On receipt of m^c from a replica o_{jk}^s, o_i^c stops a timer for T_{ijk}. After receiving m^c from every replica o_{jk}^s $(1 \leq k \leq n_j)$, o_i^c sends copies of a final message m^f which carries $seq_{m^f} = \max_k seq_{m_k^c}$ and $mfseq_{m^f} = \max_k fseq_{m_k^c}$ where m_k^c is a confirmation message from o_{jk}^s and $rID_{m^f} = rID_{m^c}$. T_{ijk} is also carried by m^f to o_{jk}^s.
4. On receipt of m^f, o_{jk}^s takes $r(op)$ out of $RBUF_{jk}$ and assigns a final sequence number seq_{m^f} to $r(op)$. $r(op)$ is sorted in APQ_{jk} in the order of the final sequence numbers. If there are multiple requests with the same final sequence number, these requests are sorted by using rID. If $mfseq - fseq_{jk} > L$ where L is predetermined threshold, o_{jk}^s invokes a catch-up procedure. Identity and idempotent requests are removed form APQ_{jk}. If a pair of successive requests in APQ_{jk} are compatible, these requests are ordered in their RTT between o_{jk}^s and the client objects.

5 Concluding Remarks

In order to realize a high performance active replication in a wide area network, this paper has proposed a pseudo-active replication. Here, a client only waits for the first response from a set of replicas. In order to reduce the recovery time from a failure of the fastest replica, a procedure for omitting identity and idempotent requests is invoked. For the invocation, each replica finds it is slower by using the sequence number assigned to the most recently finished in the replica. In addition, for reducing the response time, some compatible requests are processed in different order in a set of replicas. The order is decided based on an RTT between client objects and a replica. These additional information is carried by control messages of the totally ordering protocol. Hence, no additional control message is transmitted in the proposed protocol.

References

[1] Ahamad, M., Dasgupta, P., LeBlanc R. and Wilkes, C., "Fault Tolerant Computing in Object Based Distributed Operating Systems," Proceeding of the 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," Proceeding of the 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 Transaction on Computer Systems, Vol. 5, No. 1, pp. 47–76 (1987).

[4] Higaki, H. and Soneoka, T., "Group-to-Group Communications for Fault-Tolerance in Distributed Systems," IEICE Transaction on Information and Systems, Vol. E76–D, No. 11, pp. 1348–1357 (1993).

[5] Higaki, H., Morishita, N. and Takizawa, M., "Active Replication in Wide-Area Networks," IPSJ Technical Report, vol. 98, No. 84, pp. 93–98 (1998).

[6] Higaki, H., Morishita, N. and Takizawa, M., "Protocol for Pseudo-Active Replication in Wide-Area Network" IPSJ Technical Report, vol. 99, No. 4 , pp. 85–90 (1999).

[7] 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).

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

[9] Mattern, F., "Virtual Time and Global States of Distributed Systems," Parallel and Distributed Algorithms, North–Holland, pp. 215–226 (1989).

[10] Powell, D., Chereque, M. and Drackley, D., "Fault-Tolerance in Delta–4," ACM Operating System Review, Vol. 25, No. 2, pp. 122–125 (1991).

[11] Shima, K., Higaki, H. and Takizawa, M., "Pseudo-Active Replication in Heterogeneous Clusters," IPSJ Transaction, Vol. 39, No. 2, pp. 379–387 (1998).

[12] Tachikawa, T., Higaki, H., Takizawa, M., Liu, M., Gerla, M. and Deen, M., "Flexible Wide-area Group Communication Protocols – International Experiments," Proceedings of the 27th International Conference on Parallel Processing, pp. 570–577 (1998).