マルチメディア通信と分散処理 90－17
コンピュータセキュリティ 2－17
（1998. 9. 18）

Active Replication in Wide-Area Networks

Hiroaki Higaki, Nobumitsu Morishita and Makoto Takizawa
Department of Computers and Systems Engineering
Tokyo Denki University

Network applications are realized by the cooperation of multiple objects by exchanging messages. According to the advanced computer and network technologies, various kinds of network applications have been implemented. In order to implement mission-critical application in the network environment, replication has been introduced. Here, every server object is replicated and placed on multiple computers. Even if some of them fail, the others can continue to execute the application. In an active replication, since all the requests from clients are sent to the replicated server objects in the same order, all the replicas is surely in the same state. In the conventional active replication, the replicas are required to be synchronized. If the replicas are placed on the different kinds of computers with different processing speed, the response time of a request from a client becomes longer and the time-overhead in the system becomes high. In order to solve the problem, the authors have introduced a pseudo-active replication. However, since the speed of the replicas is measured by using the response order observed by a client in the proposed protocol for the pseudo-active replication, it is difficult to apply the pseudo-active replication to a wide-area network where the replicas are distributed and multiple clients are also distributed. Furthermore, the difference of processing speed is detected only if a client sends request messages with short interval. In order to solve this problem, this paper proposes another implementation of the pseudo-active replication. Here, the information of the processing speed in each processor is transmitted by the totally ordered protocol for transmitting a request. In addition, we introduce a novel method that the replicas intentionally computes requests in different order for reducing the response time in the clients.

広域ネットワークのための能動的多重化手法

桧垣 博章　森下 展光　滝沢 誠
{hig, nobu, taki}@takilab.k.dendai.ac.jp
東京電機大学理工学部経営工学科

ネットワークアプリケーションは、複数のオブジェクトがメッセージ交換による協調動作を行なうことによって実現される。コンピュータとネットワークの発達により、さまざまなネットワークアプリケーションが構築されている。ミッションクリティカルなアプリケーションをネットワーク上で実現するために、オブジェクトを複製して複数のコンピュータに配置する多重化手法が提案されている。能動的多重化手法では、クライアントからの要求を複製オブジェクトに対して同一順序で配送することによって、正しい多重化を実現している。従来の能動的多重化手法では、複製オブジェクト間の強い同期が要求されていたため、性能の異なるコンピュータに複製オブジェクトを配置すると時間オーバヘッドが大きくなる。この問題を解消するために、擬似能動的多重化手法が提案されている。しかし、これまでに提案された擬似能動的多重化手法の実現方法では、広域ネットワークに複製オブジェクトを分散配置し、広域に分布する複数のクライアントから要求が発生する場合、および、各クライアントの要求発生頻度が複製オブジェクトからの応答メッセージの到達時間差に比べて大きい場合、遅い複製オブジェクトを適切に特定することができなくなる。そこで、本論文では、クライアントからの要求を配送する全順序プロトコルの実行時に、複製オブジェクトの速度差を決定する方法を提案する。さらに、クライアントからの要求に対する応答時間を短縮するために、複製オブジェクトが故意に異なる順序で要求を処理する機構を導入する。

1 Introduction

According to the advance of computer and network technologies, network applications are widely developed. These applications are realized by the cooperation of multiple *objects*. Here, mission critical applications are also implemented and these applications are required to be executed fault-tolerantly. An *active replication* has been proposed where multiple replicated objects are operational in a network system. In the conventional active replication, all the replicated objects are required to be synchronized. In the network system,each replicated objects 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. Therefore, the synchronization among the replicated objects induces an additional time-overhead. The authors have been proposed a pseudo-active replication [7,13]. Here, not all the replicated objects 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 clients is also reduced.

In the proposed protocol for the pseudo-active replication discussed in [7] and [13], the difference of processing speed in the replicated objects is decided in a client by using the order of receipts of the response messages from the replicated objects. This method works well in a local-area network. However, it is difficult to apply this method to wide-area networks because the difference of the response time is based on not only the processing speed but also the message transmission delay. If the replicated objects are distributed in a wide-area network and multiple clients communicate with them, every client may decide a different object as a faster one. In this paper, we propose a modified protocol to realize the pseudo-active replication in a wide-area network. In the pseudo-active replication, a slower replicated object omits some requests from clients in order to catch up with faster replicated object for reducing the time-overhead in the recovery from the failure of the faster replicated object [7,13]. In this paper, we propose another method to achieve the synchronization among the replicated objects. Here, some requests from multiple replicas are intentionally processed in different order in each replicated object. 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. The overview of another implementation of pseudo-active replication for a heterogeneous wide-area network is discussed in section 3. In section 4, we show a novel protocol for realizing the idea proposed in the previous section.

2 Pseudo Active Replication

2.1 System Model

In a network system $\mathcal{S}$, distributed applications are realized by the cooperation of multiple *objects*. An object o_i is composed of data and operations for manipulating the data. o_i is located on a computer C_i. C_i and C_j are connected to a network and are always assumed to be able to exchange messages. o_i sometimes computes by itself and sometimes communicates with another object o_j. In most of the recent distributed applications, the objects in a network system are classified into *clients* and *servers*. A client object o_i^c request a

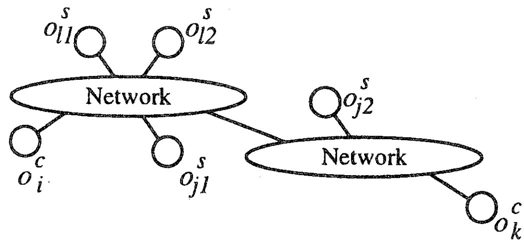

Figure 1: Replication of server objects

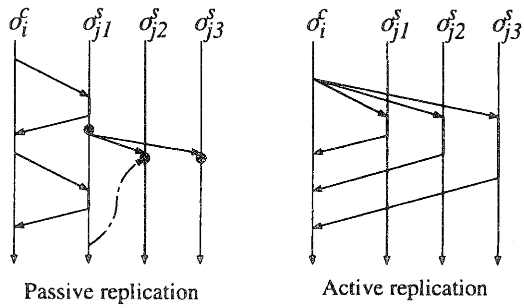

Figure 2: Passive and active replication

server object o_j^s to invoke a specified operation *op* by sending a message. o_j^s manipulates the data and responds to o_i^c also by sending a 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. In order that the application programs are executed fault-tolerantly in $\mathcal{S}$, each server object o_j^s is replicated and located on different computers [Figure 1]. 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 replicating server objects: *passive* and *active* replication [2] [Figure 2]. In the passive replication [3,4], only one of the replicas is operational. The other replicas are *passive*, i.e. these replicas do not invoke any operation. A client object o_i^c sends a request message to only the operational server replica o_{j1}^s. o_{j1}^s invokes the operation requested by o_i^c and sends back a response message to o_i^c. o_{j1}^s sometimes sends the state information to the other replicas o_{jk}^s ($2 \leq k \leq n_j$) and every o_{jk}^s updates the state information. This is called a *checkpoint*. If o_{j1}^s fails, one of the passive replicas say o_{j2}^s becomes operational. Here, o_{j2}^s restarts the execution of the application from the most recent checkpoint. Hence, the recovery procedure takes time because o_{j2}^s has to re-invoke the operations that the failed o_{j1}^s has already invoked before the fail-

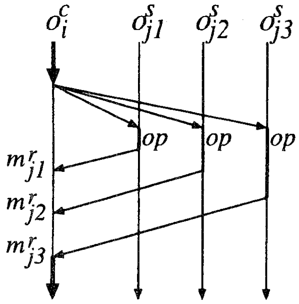

Figure 3: Synchronization overhead in active replication

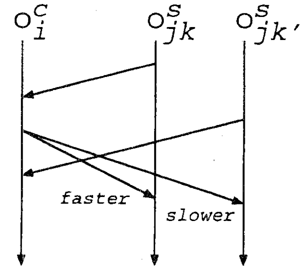

Figure 4: Pseudo-Active replication

ure.

In the active replication [1, 5, 6, 10, 12], all the replicas are operational. A client object o_i^c sends request messages to all the server replicas o_{jk}^s ($1 \leq k \leq n_j$). Every o_{jk}^s invokes the operation requested by o_i^c and sends back a response message to o_i^c. After receiving all the response messages, o_i^c accepts these messages and deliver the result to the application. That is, the server replicas o_{jk}^s are synchronized. Since all the server replicas are operational, even if some replica $o_{jk'}^s$ fails, the other replicas o_{jk}^s ($k \neq k'$) can continue to execute the application. Hence, the recovery procedure in the active replication requires less overhead than that in the passive one.

2.3 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. Hence, the computers in 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. That is, it takes the same time to finish the required operation and the same transmission delay is required for the messages between a client and the replicas. Therefore, a client object 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*. Many different kinds of computers are connected to many different kinds of networks. That is, there are processors with different processing speed, reliability and availability, and networks with different message transmission delay and message loss ratio [14]. Here, it is difficult for a client object o_i^c to receive all the response messages from the replicas o_{jk}^s ($1 \leq k \leq n_j$) of o_j^s simultaneously. In Figure 3, o_i^c delivers the result of an operation *op* to the application after receiving the response message m_{j3}^r from the slowest replica o_{j3}^s, i.e. the application in o_i^c is blocked until receiving m_{j3}^r. Therefore, the synchronization overhead for receiving the response messages is required to be reduced.

The authors have been proposed a *pseudo-active replication* [7, 13] where a client object o_i^c only waits for the first response from the replicas o_{jk}^s under an assumption that only the *stop faults* occur in the replicas, i.e. no failed object sends a message to another one [11]. On receiving the first response message from the replicas, o_i^c delivers the result to the application and restarts to execute the application. Hence, the response time in o_i^c becomes shorter and the synchronization overhead in $\mathcal{S}$ is reduced. However, since o_{jk}^s are placed on processors with different speed and are not synchronized, some replica $o_{jk'}^s$ might finish the computation for all the requests from the client objects and another replica $o_{jk''}^s$ might keep many requests not to be computed because $o_{jk''}^s$ is placed on a slower processor. In this case, if $o_{jk'}^s$ fails, the recovery procedure takes longer time than the conventional active replication because $o_{jk''}^s$ has to compute the requests that $o_{jk'}^s$ has already computed 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 object o_i^c tells the server replicas which replica is faster or slower.
2) If a replica $o_{jk'}^s$ is told to be a slower one, $o_{jk'}^s$ omits some requests from client objects in order to catch up with the faster replicas.

Suppose that a client object o_i^c waits for response messages m_{jk}^r and $m_{jk'}^r$, and sends a request message m_i. In [7] and [13], we define faster/slower replicas based on the *causal relationship* [8] among these messages [Figure 4].

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

In order for the slower replica $o^s_{jk'}$ to catch up with the faster replica o^s_{jk}, $o^s_{jk'}$ omits some requests to compute. Here, suppose that r and r' are requests, $r \circ r'$ is a concatenation of r and r' and $r(s)$ is the state of an object after r is computed in a state s.

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

[Definition: an idempotent request]
A request r is an *idempotent* request iff $r \circ r(s) = r(s)$ for every state s. □
Clearly, even if the slower replica $o^s_{jk'}$ omits identity and idempotent events, $o^s_{jk'}$ can get the same state as the faster replica o^s_{jk}.

[Omission rule]
If the following conditions are satisfied, a request r is omitted in a replica $o^s_{jk'}$:

1) $o^s_{jk'}$ is a slower replica.
2) r is an identity or idempotent request.
3) Some faster replica o^s_{jk} has computed r. □

In [7] and [13], by using *vector clocks* [9] for determining the causal relation ship among the messages, the above conditions 1) and 3) are checked in each replica $o^s_{jk}(1 \le k \le n_j)$. Here, every request message is assumed to be transmitted to all the replicated server objects in the same order, i.e. *totally ordered delivery* is assumed.

3 Pseudo-Active Replication in a Wide-Area Network

First, we remove the omission rule 3) in order to increase the probability that slower servers catch up with faster servers. By removing the confirmation whether a request r has been computed in some other replica before omitting r, the procedure to omit the waiting requests in a slower replica can be invoked more frequently. Therefore, the the slower replicas more easily catch up with the faster ones. Here, some request r may be omitted by all the replicated server objects o^s_{jk} and the client object o^c_i requesting r cannot receive any response message from the replicas. Hence, o^c_i sets a timer on requesting r. If the timer is expired before receiving a response message, o^c_i request r again.

In a wide-area network, processors on which the replicas o^s_{jk} of a server object o^s_j may be connected to different sub-networks, e.g. one is in Japan and another is in Europe, for executing mission-critical applications more fault-tolerantly. In addition, client objects may be distributed in a wide area. In this case, the receipt order of response messages in each client is not a good measurement of the processing speed in the replicated server objects. For example, all the replicated server objects may be informed to be slower [Figure 5]. Consider that a replica o^s_{jk} of o^s_j is placed far from another replica $o^s_{jk'}$, and client objects o^c_i and $o^c_{i'}$

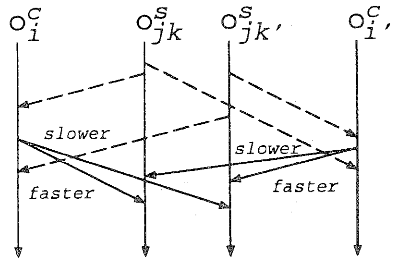

Figure 5: Pseudo-Active replication in a wide-area network

is near o^s_{jk} and $o^s jk'$, respectively. Here, we assume the processing speed of o^s_{jk} and $o^s jk'$ are the same. If o^c_i and $o^c_{i'}$ sends new request messages after receiving a response message of the previous request from near replica before receiving from far one, both o^s_{jk} and $o^s jk'$ are informed to be slower and invoke the procedure to omit the waiting requests.

The difference among response times from each replica o^s_{jk} in a client object o^c_i is caused by both the processing speed of the processors on which o^s_{ik} are placed and the message transmission delay in the channel between o^s_{jk} and o^c_i. In addition, in a wide-area network, the network system S usually consists of many client objects distributed in a wide-area. Hence, the measurement of the processing speed based on the receipt order of the response messages for the previous request in a client object 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.

In each replicated server objects, the requests which can be delivered to the application but not yet delivered are called *waiting* requests and queued until the application can accept them. The length of this queue is a good measure for the processing speed of replicas. In order to find slower replicas by using the queue length, we use the total ordering protocol proposed in [3]. This protocol consists of three phase. In the second phase, a control message cm_2 is transmitted from each replica to the client object. Here, the queue length in a replica is piggied back to cm_2. By receiving cm_2 from all the replicas, the client object can find slower replica. Ideally, the client object receives the queue length at the same time from all the replicas. However, it is impossible in a network system because of the message transmission delay. Hence, we introduce a certain threshold value to find slower replica. Only if the difference of the queue length between some replica o^s_{jk} and the others is larger than this threshold, o^s_{jk} is treated as a slower replica.

The request not being omitted by the omission

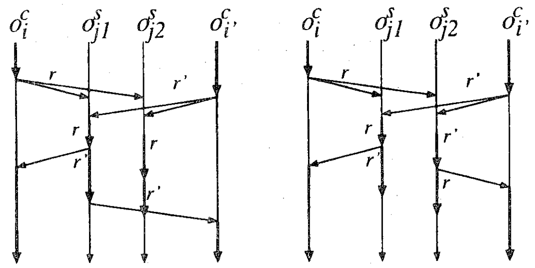

Figure 6: Intentional computing order exchange

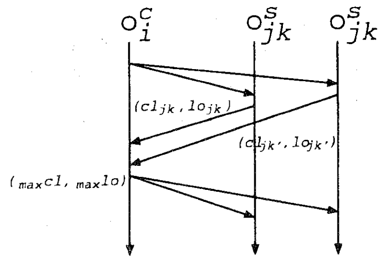

Figure 7: Total ordering protocol for pseudo-active replication

rule in the previous section are computed in the same order in every replicas. However, some pair of requests r and r' can be computed different order.

[Definition: compatible and conflict requests]
Requests r and r' are *compatible* iff $r \circ r'(s) = r' \circ r(s)$ for every state s. Otherwise, these requests are *conflict*. □
If r and r' are compatible, these requests can be computed in different order in each replica.

By computing the requests in different order in each replica, the response time in client objects may be reduced [Figure 6]. If a request r from o^c_i and another request r' from $o^c i'$ are compatible, r and r' are required to be computed first by the replica near o^c_i and $o^c_{i'}$, respectively. That is, the message transmission delay between a client objects and the replicas is reasonable for deciding the computation order of compatible requests. The message transmission delay is not constant but time-variant [14]. Therefore, it is required to be measured each time a request is sent. 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 request is postponed infinitely, the maximum number E_{max} of order exchange is predetermined. If the order of a request r is exchanged E_{max} times, r becomes a conflict request with any other request.

4 Protocol

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

- Logical clock cl_{jk} for totally ordering the requests from client objects.
- Last computed request index loi_{jk} for the measurement of processing speed of server objects.

In the following total ordering protocol, the above variables are piggied back to the control messages in order to exchange the length of the waiting request queue among the replicas [Figure 7]:

[Total ordering protocol]

1) A client object o^c_i sends request messages $req(r)$ with a request r to all the replicated server objects o^s_{jk} $(1 \leq k \leq n_j)$.
2) On receipt of $req(r)$, o^s_{jk} stores r in the buffer with cl_{jk}. o^s_{jk} sends back an ordering message $ord(cl_{jk}, loijk)$ piggying back cl_{jk} and $loijk$. cl_{jk} is incremented by one.
3) After receiving all the ordering messages from o^s_{jk} $(1 \leq k \leq n_j)$, o^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^s_{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 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 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'. □

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 need to evaluate the followings:

- The difference of the length of the waiting request queues in the replicas. If it is smaller than the conventional pseudo-active replication protocol, the system can be quickly recovered from a failure of the faster replica by using our protocol.
- The response time of the request from clients. Here, the efficiency of the intentional order exchange can be evaluated.

We are now implementing a prototype system to evaluate our 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] Borg, A., Baumbach, J. and Glazer, S., "A Message System Supporting Fault Tolerance," Proceeding of the 9th ACM Symposium on OS Principles, pp.27–39 (1983).

[5] Cooper, E.C., "Reliable Distributed Programs," Proceeding of the 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 Transaction on Information and Systems, Vol. E76–D, No. 11, pp. 1348–1357 (1993).

[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] Schneider, F., "Byzantine Generals in Action: Implementing Fail-Stop Processors," ACM Transaction on Computing Systems, Vol. 2, No. 2, pp. 145–154 (1984).

[12] Shima, K., Higaki, H. and Takizawa, M., "Fault-Tolerant Intra-Group Communication," IPSJ Transaction, Vol. 37, No. 5, pp. 883–890 (1996).

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

[14] 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).