マルチメディア通信と分散処理 91－15
（1999. 1. 22）

Protocol for Pseudo-Active Replication in Wide-Area Network

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

According to the advances of computer and network technologies, various kinds of network applications have been implemented. In order to realize mission-critical applications in the network, replication has been introduced. Every server object is replicated and placed on multiple computers. If some of them fail, the others continue to execute the application. In an active replication, since all the requests from a client are sent to all 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 observed by the application in the client depends on the slowest replica. To solve this problem, we 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 this method 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 within a short interval. Hence, 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.

広域ネットワークにおける擬似能動的多重化プロトコル

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

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、システムの動作は最も遅い複製の速度に支配されることになる。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。筆者らは、この問題を解決するために、処理待ちの要求メッセージを格納するキューの長さを用いて、複製オブジェクトの速度差を検出する方法を提案しているが、バースト的に要求が発生する場合に、どの複製オブジェクトにも処理されない要求が発生する確率が高くなるという問題があることが明らかになった。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、どの要求までアプリケーションが処理を終えたか、という情報を交換することによって、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、この方法を実現するためのシステム構成と実現プロトコルについて述べる。

1 Introduction

According to the advances of computer and network technologies, network applications are widely developed. These applications are realized by 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 response time for the application in a *client object* is depends on the speed of the slowest replicated *server object*. The authors have been proposed a *pseudo-active* replication [8, 14]. 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 client object is also reduced.

In the proposed protocol for the pseudo-active replication discussed in [8] and [14], 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 [8,14]. 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 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.

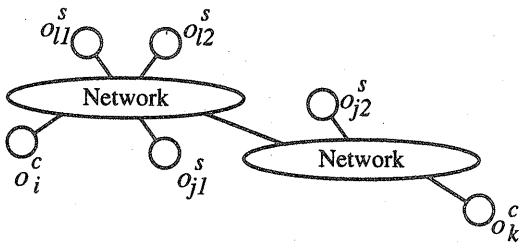

Figure 1: Replication of server objects

2 Pseudo-Active Replication in Wide-Area Networks

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^c_i request a server object o^s_j to invoke a specified operation *op* by sending a *request message*. o^s_j manipulates the data and sends back a *response message* to o^c_i. This type of communication among the objects is called *client-server* type. In this paper, all the communication among the objects is assumed to be client-server. In order that the application programs are executed fault-tolerantly in $\mathcal{S}$, each server object o^s_j is replicated and located on different computers [Figure 1]. Here, *replicas* o^s_{jk} $(1 \leq k \leq n_j)$ of o^s_j 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^c_i sends a request message to only the operational server replica o^s_{j1}. o^s_{j1} invokes the operation requested by o^c_i and sends back a response message to o^c_i. o^s_{j1} sometimes sends the state information to the other replicas o^s_{jk} $(2 \leq k \leq n_j)$ and every o^s_{jk} updates the state information. This is called a *checkpoint*. If o^s_{j1} fails, one of the passive replicas say o^s_{j2} becomes operational. Here, o^s_{j2} restarts the execution of the application from the most recent check-

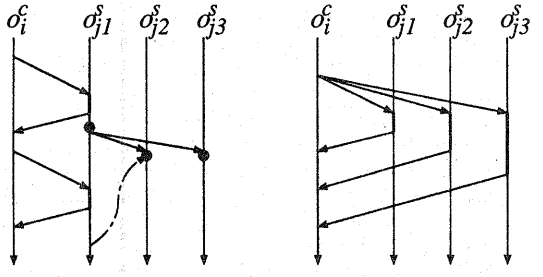

Figure 2: Passive and active replication

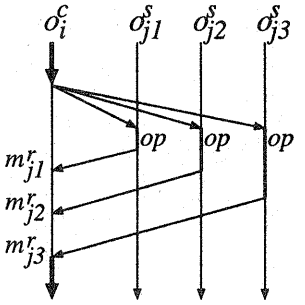

Figure 3: Synchronization overhead in active replication

point. Hence, the recovery procedure takes time because o_{j2}^s has to re-invoke the operations that the failed o_{j1}^s has already finished before the failure.

In the active replication [1, 5, 6, 11, 13], 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 delivers the result to the application. Since all the server replicas o_{jk}^s are operational and synchronized, 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 time-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. Here, 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. That is, it takes almost 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 [15]. 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* [8, 14] 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 [12]. 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 of 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 [8] and [14], we define faster/slower replicas based on the *causal relationship* [9] 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 $m \rightarrow m'$

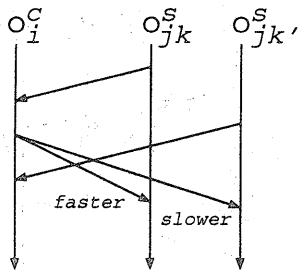

Figure 4: Pseudo-Active replication

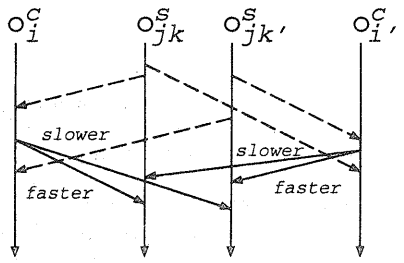

Figure 5: Pseudo-Active replication in a wide-area network

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 a faster and a slower replicas, respectively. □

2.4 Pseudo-Active Replication in a Wide-Area Network

In a wide-area network, processors on which the replicas o_{jk}^s of a server object o_j^s 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 as follows[Figure 5]. Consider that a replica o_{jk}^s of o_j^s is placed far from another replica $o_{jk'}^s$, and client objects o_i^c and $o_{i'}^c$ is near o_{jk}^s and $o_{jk'}^s$, respectively. Here, we assume the processing speed of o_{jk}^s and $o_{jk'}^s$ are the same. If o_i^c and $o_{i'}^c$ sends new request messages after receiving a response message of the previous request from near replica before receiving from far one, both o_{jk}^s 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_{jk}^s in a client object o_i^c is caused by the differences of both the processing speed of the processors on which o_{ik}^s are placed and the message transmission delay in the communication channel between o_{jk}^s and o_i^c. In addition, in a wide-area network, the network system $\mathcal{S}$ 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 in an application queue (APQ) until the application can accept them. In [7], the length of APQ is used as a mesure for the processing speed of replicas and is piggied back with the messages transmitted from replicated server objects to a client. However, if many clients near the faster replica sends request messages burstly, the APQ of the faster replica might be longer than that of the slower replica. In order to solve this problem, we apply sequence numbered assigned to the most recently processed request message SEQ as the mesure of the speed of the processors. In order to find slower replicas by using SEQ, we use the total ordering protocol proposed in [3]. This protocol consists of the following phases:

1) A client object o_i^c sends a reservation message *res* to every replicated server objects o_{jk}^s.
2) On receipt of *res*, o_{jk}^s sends back a confirmation message *conf* to o_i^c with a sequence number.
3) After receiving all the *conf* messages, o_i^c sends a final message *fin* to every o_{jk}^s with the maximum sequence number assigned to the received *conf* messages. The *fin* message carries the request message of the application.
4) On receipt of *fin*, o_{jk}^s enqueues the request message to APQ. In APQ, request messages are sorted by the assigned sequence numbers.

Here, the sequence number assigned to the most recently processed in a replica is piggied back to the *conf* message. By receiving *conf* from all the replicas, the client object can find slower replica. Ideally, the client object receives the sequence number 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 sequence numbers between some replica o_{jk}^s and the others is larger than this threshold, o_{jk}^s is treated as a slower replica.

In order for the slower replica $o_{jk'}^s$ to catch up with the faster replica o_{jk}^s, $o_{jk'}^s$ omits some re-

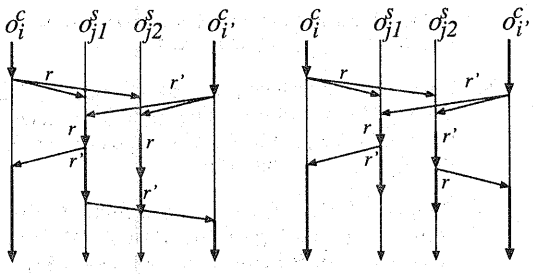

Figure 6: Intentional computing order exchange

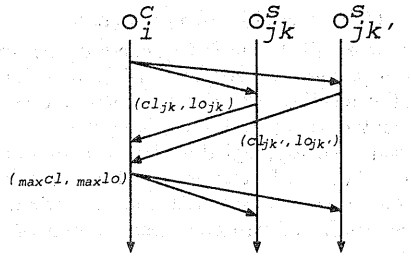

Figure 7: Total ordering protocol for pseudo-active replication

quests to compute. Here, suppose that op and op' are required operations, $op \circ op'$ is a *concatenation* of op and op' and $op(s)$ is a state of an object after op is computed 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 $o_{jk'}^s$ omits identity and idempotent operations, $o_{jk'}^s$ can get the same state as the faster replica o_{jk}^s.

[Omission rule]
If the following conditions are satisfied, an operation op is omitted in a replica $o_{jk'}^s$:

1) $o_{jk'}^s$ is a slower replica.
2) op is an identity or idempotent operation.
3) Some faster replica o_{jk}^s has computed r. □

In [8] and [14], by using *vector clocks* [10] for determining the causal relation ship among the messages, the above conditions 1) and 3) are checked in each replica $o_{jk}^s (1 \leq k \leq n_j)$. 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.

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

[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 computed 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 [Figure 6]. If an operation op requested by o_i^c and another operation op' requested by $o_{i'}^c$ are compatible, op and op' are required to be computed first by the replica near o_i^c and $o_{i'}^c$, 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 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.

3 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 $o_{jk}^s (1 \leq k \leq n_j)$ 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_i^c sends request messages $req(r)$ with a request r to all the replicated server objects o_{jk}^s $(1 \leq k \leq n_j)$.
2) On receipt of $req(r)$, o_{jk}^s stores r in the buffer with cl_{jk}. o_{jk}^s 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 o_{jk}^s $(1 \leq k \leq n_j)$, o_i^c 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}^s.
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'. □

4 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] 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," Communications of the ACM, Vol. 21, No. 7, pp. 558–565 (1978).

[10] Mattern, F., "Virtual Time and Global States of Distributed Systems," Parallel and Distributed Algorithms, North–Holland, pp. 215–226 (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 Transaction on Computing Systems, Vol. 2, No. 2, pp. 145–154 (1984).

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

[14] Shima, K., Higaki, H. and Takizawa, M., "Pseudo-Active Replication in Heterogeneous Clusters," IPSJ Transaction, 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," Proceedings of the 27th International Conference on Parallel Processing, pp. 570–577 (1998).