Short Paper

A Fault-Tolerant Protocol for Generating Sequence Numbers for Total Ordering Group Communication in Distributed Systems*

CHIH-MING HSIAO[1,2] AND GE-MING CHIU[2]
[1]*Department of Computer Science and Information Engineering*
St. John's University
Taipei, 251 Taiwan
[2]*Department of Computer Science and Information Engineering*
National Taiwan University of Science and Technology
Taipei, 106 Taiwan

In this paper, we propose a fault-tolerant scheme for generating global sequence numbers for total ordering in group communication. Our method is based on the notion of quorums, which have been used mostly to solve the mutual exclusion problem. In our scheme, each process in the group may initiate the generation of sequence numbers independently for messages emitted by itself. For the sake of enhancing fault tolerance, the information about the sequence numbers is maintained by all the members of a quorum at all times. We show that the sequence numbers generated by our scheme are unique, consecutive natural numbers. Our mechanism only incurs a moderate amount of communication traffic. The message complexity is on the order of the size of a quorum. Failure handling and crash recovery are also addressed in the paper.

Keywords: coterie, fault-tolerance, quorum, sequence number, total order

1. INTRODUCTION

In distributed computing systems, processes are often structured into groups to support various applications [3, 18], such as computer-supported-cooperative work (CSCW), replicated services, news groups, etc. A message sent to a destination group is received by all of its members. The order in which messages are *delivered* to a destination process has been an important issue in research on group communication [4, 11]. This issue is mainly due to the nondeterminism and asynchronism of network communication in distributed computing systems. Two of the most important delivery ordering requirements are *total order* and *causal order*. Causal ordering is derived from the *happened-before* relation defined by Lamport [17]. Messages are delivered according to their causal relation to satisfy the causal ordering requirement. Total ordering delivery involves the re-

Received April 21, 2004; revised August 23 & November 23, 2004; accepted December 27, 2004.
Communicated by Chu-Sing Yang.

* A preliminary version of this paper has been presented in *Proceedings of the ISCA 15th International Conference on Parallel and Distributed Computer Systems*, 2002, pp. 231-236

quirement that messages be delivered in the same relative order at each process. In this paper, we address the design of a fault-tolerant scheme for generating sequence numbers to meet total ordering requirement in group communication.

Totally-ordered delivery of messages in an asynchronous distributed system has been extensively studied in the literature [11]. Among the existing total ordering algorithms, *sequence-number-based* mechanisms have proven to be simple, efficient, and practical approaches. In these approaches, a unique sequence number is assigned to each multicast message, and the multicast messages are processed in the order of the associated sequence numbers. Sequence-number-based algorithms can be further classified into two categories: centralized and distributed algorithms. Among centralized algorithms, there are fixed-sequencer and moving-sequencer *token-based* alternatives. The Amoeba family protocols [13, 14] belong to the typical fixed sequencer method. In these protocols, one sequencer site is responsible for ordering the multicast messages on behalf of all the other processes. All sequence numbers are generated by the same sequencer site. Clearly, the sequencer site may become a bottleneck in terms of performance as all requests are addressed to the single site. Furthermore, such a fixed sequencer method has poor fault tolerance as the sequencer may become a single point of failure. To eliminate the performance bottleneck, protocols based on multiple sequencers have been developed [9, 15]. However, the lack of fault tolerance remains an unsolved problem.

On the other hand, the token-based schemes [1, 7, 12] implement total ordering and *atomic* (*safe*) message delivery by circulating a token around a logical ring formed of all of the participating processes. The token carries the most recently used sequence number. Only the token holder may generate subsequent sequence numbers for a multicast message. Essentially, the token holder can be considered as the sequencer site. A long delay may be observed by a process that has a message to be multicast as it has to wait for the arrival of the token. The length of this delay is, of course, proportional to the number of processes in the system. In addition, circulation of the token incurs extra communication overhead. This type of scheme also suffers from the problem of poor fault tolerance. In fact, detecting the loss of a token can be a more cumbersome task than dealing with the failure of a fixed sequencer site in this method. For distributed schemes [5, 6], the sequence number of a message is collectively determined by all of the participating processes in a distributed manner. The basic idea behind the technique is based on a two-phase protocol. A multicast message received by a process is first placed in the hold-back queue. Any receiving process then proposes a sequence number for the message and returns it to the sender. After collecting the proposed sequence numbers from all the receivers, the sender chooses the maximum number among the proposed ones as the final sequence number for the message and sends it to the destination processes. Messages are delivered according to the final sequence numbers. Apparently, the distributed schemes incur a large amount of communication traffic and have higher latency for generating a sequence number. Distributed schemes also lack fault-tolerance.

In this paper, we address the issue of incorporating fault tolerance into total ordering message delivery mechanisms in group communication. In a recent paper [2], Baldoni used multiple sequencers to implement a fault-tolerant service for generating sequence numbers. In that scheme, a particular sequencer replica, the primary sequencer, handles all the requests coming from clients. Other sequencer replicas act as backup ones. When a primary sequencer receives a client request, it generates a sequence number for the request and updates the backup sequencers with the sequence number. After it receives acknowledgments from all the backup sequencers, it replies to the client with the sequence number. Furthermore, it uses a time-out mechanism to detect failures. If the primary sequencer fails, a leader election procedure is followed to select the new primary sequencer from among the backup sequencers. This mechanism is essentially centralized with the primary sequencer carrying most of the operation loads.

Mobile-device-based distributed systems will become very important in the future. Some of the associated networks include mobile ad hoc networks, sensor networks and home networks. These systems typically consist of resource-constrained, mobile devices; they are normally less reliable and are vulnerable to component or communication failures. Hence, fault-tolerant measures are important for such systems. However, designating a given set of mobile devices to act as the primary sequencer and the backup ones as described in [2] would not be appropriate for mobile distributed systems. Rather, we should allow all the nodes to share the operation loads on an equal basis while providing fault tolerance for system operation.

To this end, we propose a fault tolerant scheme for generating sequence numbers for multicast messages. Our solution is to maintain the information about the most recently used sequence number at multiple participant nodes to enhance fault tolerance. Our protocol is distributed and is based on the notion of quorums [8, 21], which have been used mostly to solve the mutual exclusion problem. In the protocol, each process may initiate the generation of sequence numbers independently for messages emitted by itself. No single process failure may cause the system to crash. In addition, our scheme incurs a moderate amount of communication traffic when generating a sequence number. The message complexity is on the order of the size of a quorum. Many quorum systems had already been proposed in the literature [8, 20, 21]. In particular, the symmetrical ones [16], in which each node plays an equal role in forming a quorum, are quite suitable for the mobile distributed systems mentioned above. Moreover, the quorum sizes of these systems are normally on the order of $O(\sqrt{N})$, where N is the number of nodes in the system. Such quorum sizes are moderate for even mid-sized systems. In comparison with the previous methods, the availability of the system in our approach can be significantly enhanced.

The proposed scheme has to satisfy the following two properties:

- **Unique:** Let m and m' be two distinct messages that are multicast to the group, respectively. If any process p is in the group, it is true that m and m' have different delivery sequence numbers at p.
- **Consecutive:** For any process p in the group, there exist a contiguous natural sequence number for each message received by p; i.e., the sequence numbers of the messages at p do not present “holes.”

The global sequence number generated by the proposed protocol is guaranteed to satisfy both the unique and consecutive properties; therefore, message delivery according to the global sequence number results in total ordering. The correctness of the protocol is also rigorously proved in this paper. In addition, failure handling and crash recovery are also addressed.

The paper is organized as follows. In section 2, the system model and some preliminaries are described. We present the protocol in section 3. We then discuss some properties of the proposed scheme in section 4. Section 5 draws conclusions.

2. SYSTEM MODEL AND PRELIMINARIES

2.1 System Model

We consider a distributed system which consists of N processes, p_1, p_2, …, p_n, which communicate with each other by passing messages via the underlying network. These processes together form a single multicast group (or, simply, a group). A message sent to the group will be received by all of the processes. We define a requester process as a process which generates a sequence number for a message that is sent by the process. Without loss of generality, we make the following assumptions in the context of our protocol discussion.

- **On the network.** The network consists of a set of sites that communicate through channels. Processes residing on the sites can transmit messages in both the point-to-point and broadcast modes.
- **On the communication mode.** Communication is asynchronous. That is, a message (either point-to-point or broadcast) requires a variable amount of transmission time for completion. It is assumed that communication is reliable and exhibits the FIFO ordering property.
- **On fault-tolerance.** The following kinds of errors will be tolerated.
  - ➢ *Crash failures*: A process may fail by halting prematurely. Until it halts, it behaves correctly. A process failure does not involve malicious actions. We do not consider Byzatine failures. A process that is excessively slow or fails to receive a message many times can be regarded as having failed based on a timeout setting (see below). We assume that the requester process does not fail during the generation of a global sequence number.
  - ➢ *Network partitioning*: A network may be partitioned into segments, and the processes in one segment may appear to have failed from the viewpoint of the processes in the other segments. It is assumed that the communication paths between the partitions will be reconnected eventually.

2.2 Quorums

In the proposed scheme, we employ the notion of *quorums* to achieve fault-tolerance. Basically, a quorum is a subset of the N processes in the group. All the quorums of the system forms a *coterie* [21]. They are formally defined as follows.

Definition Let C be a collection of distinct sets $C = \{Q_1, Q_2, \ldots, Q_m\}$, where each Q_i, $1 \leq i \leq m$, is a nonempty subset of the processes in the group. C is called a coterie if and only if the following two properties hold:

1. Intersection property: $Q_i \cap Q_j \neq \emptyset$, $\forall Q_i, Q_j \in C$.
2. Minimality property: $Q_i \not\subset Q_j$, $\forall Q_i, Q_j \in C$.

Any $Q_i \in C$ is called a quorum.

As an example, consider the system consisting of six processes $p_1, p_2, \ldots, p_6$. The following five quorums may form a coterie: $Q_1 = (p_1, p_2, p_4)$, $Q_2 = (p_1, p_3, p_6)$, $Q_3 = (p_1, p_3, p_5)$, $Q_4 = (p_3, p_4, p_5)$ and $Q_5 = (p_4, p_5, p_6)$.

A number of methods have been proposed in the literature [8, 20] for constructing coteries for a distributed system. In our mechanism, one process is adopted to construct a coterie for the system and disseminate it to all the other processes in the group. The quorums in a coterie maintained by all the processes in the group are used to determine the global sequence number.

3. THE PROTOCOL

In this section, we will show how a global sequence number is generated from quorums. We will also address a scheme for achieving recovery when processes in the system crash. In the following, we will describe in details how a consistent sequence number for a message is generated. Note that for any message, the sequence number assigned to it must be unique and consecutive.

3.1 Operations in a Requester Process

In the protocol, it is required that each process maintains a natural number, called a *local sequence number*, which is initialized to zero. The local sequence number of process p_i is denoted as LC_i for $1 \leq i \leq N$. If process p_i wants to send a message m to the group, then p_i becomes a requester process. p_i first selects a quorum from the *coterie* and multicasts a request message Req_i to all the processes of the selected quorum. p_i then waits for acknowledgements from all the members of the quorum. A process p_j will reply to the requester process upon receiving the request message with a proposed sequence number, $LC_j + 1$, if it is available. If some process indicates that it is not available, then p_i will simply give up and instruct the other members to ignore the request. It will try again later. A process is available for servicing a request if it is not currently servicing another request. Suppose that all of the acknowledgement messages indicate available services. Since there exists a quorum in the coterie maintaining a global sequence number at any time, the largest number among the proposed sequence numbers is assigned to m as its final sequence number. p_i then multicasts this sequence number to all of the members of the selected quorum. All these members will simply update their local sequence numbers to the final global sequence number and then release their locks for the request. Finally, p_i will multicast m piggybacked with the global sequence number to all the processes in the group. The following **Procedure** *requester* are used by the requester process to handle the generation of the global sequence number. *Tr* denotes the timeout interval for detecting a process crash. The notation Δ is the maximum time allowed for Req_i to reach any member of a quorum and δ is the maximum amount of time for a point-to-point message Ack_j to reach the requester. Hence, $Tr > (\Delta + \delta)$ in the **Procedure** requester implies that the requester process detects that some process has failed.

```
Procedure requester
/* message: $m$, requester process: $p_i$, chosen quorum: $Q$. */
begin
    choose a quorum $Q$ from the coterie;
    multicast $Req_i$ to $p_j$, $\forall p_j \in Q$;
    wait for $Ack_j$ from $p_j$, $\forall p_j \in Q$;
    if {($Tr > (\Delta + \delta)$) and ($\exists Ack_j$ is not received by $p_i$)}    /* some processes in $Q$ fail */
        then backoff and try again later;
    if ($\exists p_j \in Q$ responds with unavailable service)
        then instruct $p_j$ to ignore the request, $\forall p_j \in Q$, to release the lock and try again later;
                                                        /* Abort the service for $p_i$ */
    else
     begin
           $LC_i$ = max(sequence number proposed by all $p_j$), $\forall p_j \in Q$;
                                                        /* compute the global sn */
           multicast $LC_i$ to $p_j$, $\forall p_j \in Q$;     /* $p_j$ update their local sequence numbers with $LC_i$ */
           multicast $m$ piggybacked with $LC_i$ to the group;
     end
end
```

3.2 Operations in a Quorum Process

When a process, say p_j, of a selected quorum receives the request message Req_i, it responds with an acknowledgment message Ack_j, in which a proposed sequence number for the requester process p_i is included if it is not currently serving another request; otherwise, it responds with an indication that the service is currently not available. The number proposed by process p_j is $LC_j + 1$. As there exits at least one quorum in the coterie that maintains a global sequence number at any time in the group, there must exist at least one process in the selected quorum that has a global sequence number. When p_j sends Ack_j with the proposed number to p_i, it locks its services until it receives notification from the requester process. This is to prevent the assignment of inconsistent global sequence numbers for different messages. If p_j is instructed to ignore the request, it aborts the service for p_i's request and releases the lock. The following **Procedure** *quorum-member* is used by the members of the quorum to handle the receipt of a request message from the requester process.

```
Procedure quorum-member
/* selected quorum: $Q$, member process of $Q$: $p_j$, requester process: $p_i$, message from $p_i$: $M$ */
begin
```

```
if ($M = Req_i$) then
    if ($p_j$ is not currently servicing another request)
        then sends $Ack_j$ to $p_i$ with $LC_j + 1$ and locks the service;
    else
        sends $Ack_j$ to $p_i$ with unavailable service and exit;
if ($M$ is a notification message from $p_i$ asking to ignore the request) then
    abort the service for $p_i$'s request and exit;
if ($M$ contains a final sequence number) then
    update $LC_j$ with the $LC_i$ and unlock the service;
                                        /* Update the local sequence number */
end
```

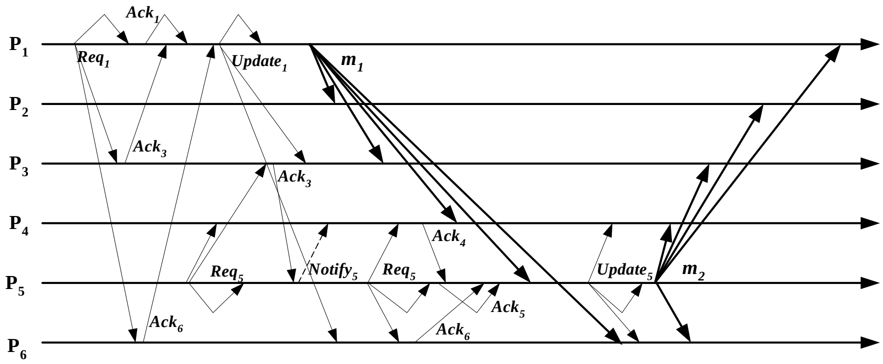

Fig. 1. Generating a sequence number.

For example, consider the system shown in Fig. 1. The local sequence number of the six processes are all set to 0 initially; that is, $LC_i = 0$ for $i = 1 \ldots 6$. Suppose that two processes p_1 and p_5 want to multicast m_1 and m_2, respectively. p_1 chooses a quorum $Q_2 = (p_1, p_3, p_6)$ and multicasts Req_1 to them. At the moment, p_5 chooses a quorum $Q_4 = (p_3, p_4, p_5)$ and multicasts Req_5 to them. After every process in Q_2 receives Req_1, the sequence numbers proposed by p_1, p_3 and p_6 are (0 + 1), (0 + 1) and (0 + 1), respectively. After p_1 has received the acknowledgment messages Ack_1, Ack_3 and Ack_6, it selects the maximum proposed sequence numbers from (0 + 1, 0 + 1, 0 + 1) as the global sequence number. Then p_1 updates LC_1, LC_3 and LC_6 with the global sequence number by multicasting an updated message $Update_1$ containing the final sequence number to quorum Q_2; that is, the values of LC_1, LC_3 and LC_6 are all updated to 1. Meanwhile, p_3 receives Req_5 but is currently servicing Req_1. p_3 aborts p_5's request by sending Ack_3 with abort service to p_5. After p_5 receives Ack_3, it sends a notification message $Notify_5$ with abort service to p_4, which is currently servicing Req_5. At a later time, p_5 tries to choose $Q_5 = (p_4, p_5, p_6)$ and multicasts Req_5 to them. After all the processes in Q_5 receive Req_5, the sequence numbers proposed by p_4, p_5 and p_6 are (0 + 1), (0 + 1) and (1 + 1), respectively. After p_5 has received Ack_4, Ack_5 and Ack_6, it selects the maximum proposed sequence numbers from (0 + 1, 0 + 1, 1 + 1) as the global sequence number and updates LC_4, LC_5 and LC_6 to 2. Finally, p_5 multicasts m_2 with a global sequence number 2 to the group. At this time, the local sequence numbers (LC_1, LC_2, LC_3, LC_4, LC_5, LC_6) are (1, 0, 1, 2, 2, 2).

3.3 Correctness Proof

In the following, we will show that the global sequence numbers generated by the proposed mechanism are indeed unique and consecutive natural numbers, and that totally ordering message delivery is achieved by having messages delivered according to the global sequence numbers.

Lemma 1 Let m_i and m_j be two distinct messages that are multicast to the group. It is true that m_i and m_j are assigned two different sequence numbers (unique property).

Proof: Suppose that processes p_i and p_j are different requester processes that multicast m_i and m_j, respectively. According to the procedure requester, the request messages Req_i and Req_j are sent to two selected quorums in the coterie. According to the definition of a quorum, there exists at least one process that is common to the two quorums. This guarantees that one of these requests is serviced after the other request has been done. Therefore, different sequence numbers are assigned to m_i and m_j, respectively. ❐

Lemma 2 Global sequence numbers are assigned in a consecutive manner (consecutive property).

Proof: In our protocol, the most recently used global sequence number is always maintained by all of the members of a quorum. The intersection property of the quorums ensures that any other quorum will contain at least one member process whose local sequence number represents the most recently used (largest) global sequence number. Hence, the global sequence number obtained by a requester process must be the next larger number that can be used. Since only one requester process is serviced at one time, the sequence numbers assigned are contiguous; that is the sequence numbers of messages at any process in the group do not present "holes." ❐

Theorem The mechanism achieves global total ordering message delivery.

Proof: By Lemma 1 and Lemma 2, the sequence numbers of all the messages sent to the group are unique and consecutive. Therefore, the messages delivered according to the order of the sequence number results in total ordering in our mechanism. ❐

3.4 Failure Handling and Crash Recovery

Processes may fail (or crash) from time to time. In addition, a crashed process may recover. In our protocol, the most recently used global sequence number is always held by all the processes of a quorum. As long as the number of failure processes is smaller than the quorum, the system may continue to function correctly despite failures. However, this is contingent upon that there existing at least one completely functioning quorum in the system. Several types of systems, such as triangular grid quorum systems [16], are (k − 1)-fault tolerant in that they can tolerate up to $k - 1$ failures without all quorums being rendered useless, where k is the uniform size of the quorums. In the following discussion, we will assume that such construction techniques are used to generate the coterie.

When more than $k - 1$ processes in the system crash, there is the possibility that no functioning quorum exists. In this case, a new coterie must be constructed before the system may resume operation. We require that the system construct the coterie once it detects that there are $k - 1$ failures. The most recently used global sequence number can be retrieved from the existing quorum.

Now consider the situation where a process that previously crashed recovers from its failure. If the coterie has not changed since the crash, the recovered process may proceed with the existing coterie. It only needs to reset its local sequence number to zero so that no other multicast message can accidentally use the sequence number proposed by the recovered process as the next global sequence number. No further action is required. If the coterie has been changed since the process crashed earlier, the return of the process must be added to the formation of the coterie. In this case, the operation should be halted before the new coterie is constructed. To ensure that the system resumes with the correct sequence number, all the functioning processes must obtain a copy of the latest global sequence number. This can be achieved by having the recovered process send a special request to a functioning quorum to ask for the latest global sequence number that has been assigned. It then broadcasts this number to all of the living processes in the system. The sequence number is assumed for all the local sequence numbers of the processes when the system resumes operation.

In addition, how a requester process effectively finds a functional quorum in a coterie with faulty processes is an important issue. To handle the task of locating of a functional quorum in the presence of faults, we may resort to the following two techniques:

1. In the presence of faults, a process may recognize that certain faults have occurred previously and avoid selecting quorums that include these faults. In other words, our scheme may require that a process spend more time for finding a functional quorum the first time it experiences unsuccessful search for a quorum due to faults. It may, however, avoid going to these faulty components the next time it requests a global sequence number for a multicast message.
2. If a requester fails to find a fully functioning quorum, it may be notified by those that replied to its request about the failure status of the system known to the repliers. For example, suppose that process A is the requester. It first sends a request to the processes of a chosen quorum. When process B (a fault-free one in the chosen quorum) replies to A, it can notify A about the failure status it has knowledge of so that A may use the information to facilitate subsequent retrials, if they are needed.

Note that our protocol is not intended for use in an environment in which the number of failures is large and processes frequently fail. In fact, providing fault tolerance would be too costly for any method under severe fault conditions.

4. DISCUSSION

In the existing centralized methods, the most recently used global sequence number is maintained by a single site (the fixed sequencer site or the token holder). In contrast, our proposed sequence number is always held by all of the processes of a quorum. As long as there exists a quorum whose members are all alive, the system may continue to function correctly despite failures. In addition, the information carried in the request and the acknowledgment messages are no more than a few integers. Furthermore, the operations needed to process a request message are simple. At any destination process, a message can be delivered as soon as it becomes deliverable with our scheme. This feature not only shortens the period of latency experienced by a message but also simplifies buffer management at the receiving processes. It would be desirable for the load involved in servicing requests be evenly distributed among all the processes in the system. This property is related to the way in which the quorum system is used. Load-balanced quorum systems, in which each process assumes the same responsibility, have been extensively studied [20, 21]. In these systems, each process is included in the same number of quorums, and all quorums have equal size. One may achieve an evenly distributed load by using a load-balanced coterie and having each requester process select quorums with equal probability.

The number of messages that need to be transmitted in order to complete the transmission of a message is called the *message complexity*. For the purpose of analysis, we assume that our system is based on the use of a broadcast network, such as Ethernet, in which a message sent to multiple sites only costs one message in the network and takes the same order of times as the message is sent to a single site. Assume that multiple sequencer sites are used to generate sequence numbers. In [2], Baldon adopted a primary-backup scheme in sequencer replicas to implement a fault-tolerant sequencer for generating sequence numbers. Assume that there are k sequencer replicas. $k + 3$ messages are required to multicast a message to the group, send one message to the primary sequencer, send one message from the primary sequencer to all backup sequencers, send another $(k - 1)$ acknowledgment messages back to the primary sequencer from $(k - 1)$ backup sequencers, send a third message with the sequence number sent to the send by the primary sequencer, and finally broadcast the multicast message to the destinations.(all quite confusing) The message complexity of the multiple sequencer scheme is, thus, $O(k)$. In addition, if the primary sequencer fails, a leader election procedure is performed to select the new primary sequencer from among the backup sequencers. The system cannot function until the new leader sequencer is completely elected. If the primary site and the backup sites all fail, the primary/backup mechanism may not function at all. In our scheme, as long as there exists at least one functional quorum, the system can continue to work correctly despite failures. From the fault tolerance point of view, the primary/backup approach may face more stringent conditions.

Consider that the distributed scheme ABCAST [6] requires three rounds of message exchanges: one message sent by the sender to destinations, N messages carrying proposed sequence numbers sent back to the sender from all N destinations and, finally, a third message broadcast to the destinations for commitment, where, again, N is the number of processes in the group. In total, this approach requires $N + 2$ messages to multicast a message to the group. The message complexity of ABCAST is, thus, $O(N)$. Our scheme also requires three rounds of message transmission for each request: one message sent from the requester process to the member of a quorum, followed by $\sqrt{N}$ acknowledgment messages sent by all the members of a quorum and, finally, a third message broadcast to all the processes in the group. Here, as in most cases, we assume that the size of the load-balanced quorums is $\sqrt{N}$, and that no process in the quorum fails. In total, our scheme requires $\sqrt{N}+2$ messages to multicast a message to the group in the best case. The message complexity is $\sqrt{N}$. Therefore, the proposed scheme incurs a smaller amount of communication traffic than ABCAST dose. In addition, unlike ABCAST, our scheme provides a fault-tolerant solution for totally ordered group communication.

5. CONCLUSION

In this paper, we have presented a fault-tolerant scheme for generating global sequence numbers for multicast messages in order to achieve totally ordered message delivery. The proposed scheme is based on the use of quorum systems. The sequence numbers generated in the system have been shown to be unique, consecutive natural numbers. Further, the availability and fault-tolerance of services can be significantly enhanced in comparison with the existing sequence-number-based methods. In addition, the message complexity of the proposed scheme compares favorably with the existing distributed method and multiple sequencer scheme.

ACKNOWLEDGMENTS

We wish to thank the anonymous reviewers for their insightful comments. This work was supported in part by the National Science Council of Taiwan, R.O.C., under grant No. NSC 91-2213-E011-050.

REFERENCES

1. Y. Amir, L. E. Moser, P. M. Melliar-Smith, V. Agrawala, and P. Ciarfella, "The totem single-ring ordering and membership protocol," *ACM Transactions on Computer Systems*, Vol. 13, 1995, pp. 311-342.
2. R. Baldoni, C. Marchetti, and S. Tucci-Piergiovanni, "A fault-tolerant sequencer for timed asynchronous systems," in *Proceedings of the 8th International Euro-Par Conference*, 2002, pp. 578-588.
3. K. P. Birman, "The process group approach to reliable distributed computing," *Communications of the ACM*, Vol. 36, 1993, pp.37-53.
4. K. P. Birman, "Building secure and reliable network applications," in *Proceedings of the Worldwide Computing and Its Applications* (*WWCA*), 1997, pp.15-28.
5. K. P. Birman and T. A. Joseph, "Reliable communication in the presence of failures," *ACM Transactions on Computer Systems*, Vol. 5, 1987, pp. 47-76.
6. K. P. Birman, A. Schiper, and P. Stephenson, "Lightweight causal and atomic group multicast," *ACM Transactions on Computer Systems*, Vol. 9, 1991, pp. 272- 314.

7. J. M. Chang and N. F. Maxemchuck, “Reliable broadcast protocols,” *ACM Transactions on Computer Systems*, Vol. 2, 1984, pp. 251-273.
8. S. Y. Cheung, M. H. Ammar, and M. Ahamad, “The grid protocol: a high performance scheme for maintaining replicated data,” *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, 1992, pp. 582-592.
9. G. M. Chiu and C. M. Hsiao, “Load-balanced generating sequence-numbers protocol for total ordering in overlapping groups,” in *Proceedings of he International Conference on Dependable Systems and Networks*, Fast Abstracts, 2002, pp. 54-55.
10. G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems – Concepts and Design*, Addison-Wesley Ltd., 2001.
11. X. D’efago, A. Schiper, and P. Urb’an, “Totally ordered broadcast and multicast algorithms: a comprehensive survey,” Technical Report No. DSC/2000/036, ’Ecole Polytechnique F’ed’erale de Lausanne, Switzerland, 2000.
12. W. Jia, J. Cao, X. Jia, and C. H. Lee, “Design and analysis of an efficient and reliable atomic mulitcast protocol,” *Computer Communications*, 1998, pp.37-53.
13. M. F. Kaashoek and A. S. Tanenbaum, “Group communication in the amoeba distributed operating system,” in *Proceedings of the 11th International Conference on Distributed Computing Systems*, 1991, pp. 222-230.
14. M. F. Kaashoek and A. S. Tanenbaum, “An evaluation of the amoeba group communication systems,” in *Proceedings of the 16th International Conference on Distributed Computing Systems*, 1996, pp. 436-447.
15. T. Kindberg, “A sequencing service for group communication,” Technical Report No. 698, Department of Computer Science, Queen Mary & Westfield College, University of London, 1995.
16. Y. C. Kuo and S. T. Huang, “A geometric approach for constructing coteries and *k*-coteries,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, 1997, pp. 402-411.
17. L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” *Communications of the ACM*, Vol. 21, 1978, pp. 558-565.
18. L. Liang, S. T. Chanson, and G. W. Neufeld, “Process groups and group communications: classifications and requirements,” *IEEE Transactions on Computers*, Vol. 23, 1990, pp. 56-66.
19. C. M. Lin, G. M. Chiu, and C. H. Cho, “An efficient quorum-based scheme for managing replicated data in distributed systems,” in *Proceedings of the International Conference on Parallel Processing*, 1999, pp. 328-335.
20. C. M. Lin, G. M. Chiu, and C. H. Cho, “A new quorum-based scheme for managing replicated data in distributed systems,” *IEEE Transactions on Computers*, Vol. 51, 2002, pp. 1442-1447.
21. W. S. Luk and T. T. Wong, “Two new quorum based algorithms for distributed mutual exclusion,” in *Proceedings of the International Conference on Distributed Computing Systems*, 1997, pp. 100-106.

Chih-Ming Hsiao (蕭志明) received the B.S. degree in computer science from Tatung Institute of Technology, Taiwan, in 1994, and the M.S. degree and the Ph.D. degree in Electrical Engineering from National Taiwan University of Science and Technology, Taiwan, in 1996 and 2005, respectively. He is currently an assistant professor in the Department of Computer Science and Information Engineering at St. John's University, Taiwan. His research interests include distributed system and mobile computing.

Ge-Ming Chiu (邱舉明) received the B.S. degree from National Cheng Kung University, Taiwan, in 1976, the M.S. degree from Texas Tech University in 1981, and the Ph.D. degree from the University of Southern California in 1991, all in Electrical Engineering. He is currently a professor in the Department of Computer Science and Information Engineering at National Taiwan University of Science and Technology, Taipei, Taiwan. His research interests include distributed computing, mobile computing, fault-tolerant computing, and parallel processing. Dr. Chiu is a member of the IEEE Computer Society.