A Token-Based Distributed Algorithm for Total Order Atomic Broadcast

Sandip Dey[1] and Arindam Pal[2]

[1] Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India
sandip@csa.iisc.ernet.in
[2] Microsoft Corporation, Hyderabad, India
arindamp@microsoft.com

Abstract. In this paper, we propose a new token-based distributed algorithm for total order atomic broadcast. We have shown that the proposed algorithm requires lesser number of messages compared to the algorithm where broadcast servers use unicasting to send messages to other broadcast servers. The traditional method of broadcasting requires $3(N-1)$ messages to broadcast an application message, where N is the number of broadcast servers present in the system. In this algorithm, the maximum number of token messages required to broadcast an application message is $2N$. For a heavily loaded system, the average number of token messages required to broadcast an application message reduces to 2, which is a substantial improvement over the traditional broadcasting approach.

1 Introduction

Various distributed applications require that a group of processes receive messages from different sources and take actions based on these messages. These actions can be updating a replicated database or executing a method on a replicated object etc. *Broadcasting* is a technique, in which an arbitrary one among N processes sends a message to the other $N-1$ processes. Although broadcasting can always be achieved by sending $N-1$ *unicast* messages and waiting for the $N-1$ acknowledgments, this algorithm is slow, inefficient and wasteful of network bandwidth.

By *atomicity* we mean that, if a message is delivered to one among a group of processes, then it will be delivered to all other processes in the group. By *total ordering* we mean that, if a process receives a message p before a message q, then all other processes in the group will also receive message p before message q. A message delivery rule is considered *safe*, if before a broadcast server delivers a message, it ensures that every other broadcast server has received that message. A message is considered *stable*, if every broadcast server in the system knows that every other broadcast server has received the message.

Various issues of broadcasting are addressed in [1,2,3]. In [4], a centralized broadcast protocol has been described. Here, application processes send messages to a central entity called *sequencer*. If the *sequencer* fails, a new *sequencer* has

S.K. Das and S. Bhattacharya (Eds.): IWDC 2002, LNCS 2571, pp. 343–347, 2002.
© Springer-Verlag Berlin Heidelberg 2002

to be chosen before messages can be broadcasted. So, this algorithm has a single point of failure.

In this paper, we propose a total order, atomic broadcast protocol that allows broadcasting of messages using a unidirectional ring. A set of *broadcast servers* deliver messages on behalf of *application processes*. The broadcast servers are organized in a unidirectional logical ring and a *token* carrying broadcast messages circulates around the ring. The algorithm does not use any broadcast primitive. Instead it uses *unicasting* between broadcast servers to broadcast messages. The algorithm is fully distributed and there is no single point of failure.

The rest of the paper is organized as follows. In section 2, the system model is described. Section 3 presents the algorithm. In section 4, we have stated and proved the properties of the algorithm. Finally, section 5 concludes the paper.

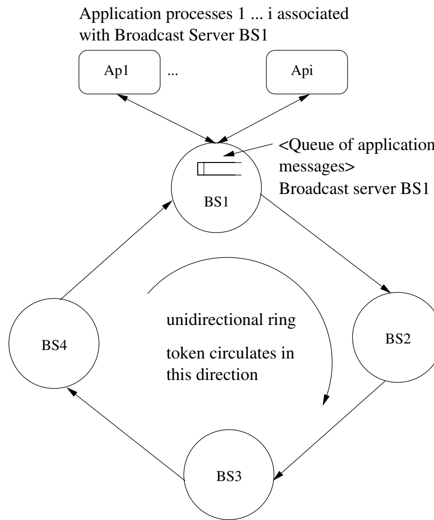

Fig. 1. System Model

2 System Model

The system consists of a set of broadcast servers and a set of application processes, where each application process is associated with a unique broadcast server. We consider an asynchronous distributed environment where processes communicate with each other by explicit message passing only. Send primitive is non-blocking and receive primitive is blocking in nature. The broadcast servers are failure-free and no messages are lost in transit. The communication channels

between a broadcast server and its associated application processes are assumed to be first-in first-out (FIFO). The system model is depicted in Fig. 1.

3 The Algorithm

Our algorithm implements a broadcast protocol, which in essence uses unicasting to communicate between processes. Broadcast servers take the responsibility of broadcasting messages received from a group of application processes. Each broadcast server knows its identity, the identity of the neighboring broadcast server and the total number of broadcast servers present in the system. These informations are required to connect the broadcast servers in a unidirectional logical ring. Once the ring is established, each broadcast server starts a new thread called *ring handler* which takes care of broadcasting issues. The main *broadcast server* thread waits for a *connection* from some application process. When a *connection* is received, it invokes an *application client handler* thread which handles the communication with that particular application process. The main server thread continues waiting for other connections. So the broadcast servers are concurrent in nature, i.e., they can handle several client requests simultaneously.

An application client may dynamically join a new broadcast server. When a new application client starts execution, it first establishes a *connection* with a broadcast server. The main *application client* thread then invokes a new thread called *broadcast message receiver* which waits for application messages from the associated broadcast server. These application messages are generated by other application processes in the system and are now being broadcasted. The main *application client* thread runs simultaneously with the *broadcast message receiver* thread and generates application messages to be broadcasted. Each application client keeps an application message counter. A value of i for this counter indicates that this is the ith application message generated by this application client. Application messages are generated randomly i.e., after generating a message the application client sleeps for a random time and then generates another message. All these application messages are sent to the associated broadcast server.

A broadcast server stores messages from its application clients in a queue. There is no limit on the length of the queue. Whenever a broadcast server receives a message from an application client, it puts the message in the queue. A token circulates around the logical ring and it carries with it N number of application messages, where N is the number of broadcast servers present in the ring. The *ring handler* thread running at each broadcast server checks whether the message queue is empty or not. If the queue is empty, the broadcast server is currently not *interested* to broadcast. If the broadcast server is *interested*, has the token and the token has no previous message (still undelivered) from this broadcast server, it picks a message from the queue and puts the message in the token. This message gets a sequence number equal to the value of the sequence number in the token and the sequence number in the token is incremented. In other words, the token is carrying a unique sequence number generator with it. The

circulation count for this message is initialized to 0, indicating that it has been just put into the token. The broadcast server then forwards the token to its neighbor. Whenever a broadcast server receives an application message for the first time, i.e., with a *circulation count* equal to 0, it copies the application message in a local buffer. This is important to satisfy the *safe* delivery rule. If the broadcast server is *interested* but the token has a previous message from this broadcast server which is not delivered to all the application processes yet, the broadcast sever just increments the *circulation count* of its message in the token and forwards the token to its neighbor. After forwarding the token it simply waits for the token to arrive again. When an application message circulates twice around the ring, it becomes stable. Once an application message becomes stable, it can be removed from the token. Each broadcast server delivers all the application messages with *circulation count* equal to 1, to associated application processes. The token has N slots for N application messages for each of the N broadcast servers. When a message needs to be delivered to the application clients, the *ring handler* thread starts a *broadcast message sender* thread which delivers the message to all the application clients associated with the broadcast server.

4 Properties of the Algorithm

Theorem 1. *The algorithm ensures atomicity and total ordering of messages.*

Proof. A broadcast server delivers an application message to all its associated application clients. Also, safe delivery rule ensures that every other broadcast server has received that application message. So, if an application process receives an application message, others will also receive it. This ensures atomicity.

Now, if a broadcast server delivers a message p before a message q, then every other broadcast server does the same because messages are delivered by a broadcast server only when the token completes one full circulation with respect to that broadcast server. Hence, every application process associated with a broadcast server will receive message p before q. This way the total ordering of messages is ensured.

Theorem 2. *The maximum number of token messages required to broadcast an application message is $2N$, where N is the number of broadcast servers in the system. For a heavily loaded system, the average number of token messages required to broadcast an application message reduces to* 2.

Proof. A message is stable when the token completes two full circulations. When a message is stable it has been delivered by all the broadcast servers. Hence the maximum number of token messages required to broadcast a message is $2N$.

For a heavily loaded system, each broadcast server will put one message in its slot. Hence, in two full circulations of the token N messages will be delivered. In other words, $2N$ token messages will be required to deliver N messages. Hence, the average number of token messages required to broadcast a message is 2.

Compare this with the number of messages required for a complete network of N broadcast servers. Here, a broadcast server which wants to broadcast a message will send it to $N-1$ other broadcast servers. They will send acknowledgement messages back to the original broadcast server. The original broadcast server will send replies to all of them and every broadcast server will know that every other broadcast server in the system has received the message. Thus to satisfy the safe delivery rule $3(N-1)$ messages are required. Hence, even for a lightly loaded system, our algorithm requires $3(N-1)-2N = (N-1)$ messages less for broadcasting.

5 Conclusion

In this paper, we have designed a fully distributed algorithm for total order atomic broadcast and have shown that our algorithm requires lesser number of messages compared to the algorithm where broadcast servers use unicasting to send messages to all other broadcast servers.

Here, we have not considered any fault-tolerance issue. However, our algorithm can be modified to handle token loss. This can be achieved by using extra sequence numbers in the token and keeping a timer at each broadcast server. Two consecutive application messages having non-consecutive sequence numbers will indicate a message loss.

References

1. H. Garcia-Molina, N. Lynch, B. Blaustein, C. Kaufman, and O. Schmueli. Notes on a Reliable Broadcast Protocol. Technical report, Computer Corporation of America, July 1985.
2. J. M. Chang and N. F. Maxemchuk. Reliable Broadcast Protocols. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.
3. P .M. Melliar-Smith, L. E. Moser, and V. Agrawala. Broadcast Protocols for Distributed Systems. *IEEE Transactions on Parallel and Disributed Systems*, 1(1):17–25, January 1990.
4. M. F. Kaashoek, A. S. Tanenbaum, S. F. Hummel, and H. E. Bal. An Efficient Reliable Broadcast Protocol. *Operating Systems Review*, 23(4):5–19, October 1989.