ELSEVIER

Information Processing Letters 50 (1994) 311-316

Information Processing Letters

An optimal algorithm for distributed snapshots with causal message ordering

Sridhar Alagar [*,1], S. Venkatesan [1]

Computer Science Program, EC 31, University of Texas at Dallas, Richardson, TX 75083, USA

Communicated by J. Misra; received 1 March 1994

Abstract

We present an optimal distributed algorithm to record a global state of a distributed system with causally ordered message delivery. The message complexity of our algorithm is $O(n)$ bits where n is the number of processes in the system.

Keywords: Distributed systems; Global states; Causal ordering; Message complexity

1. Introduction

A *global state* of a distributed system is a "consistent" collection of local states of the processes and the channels. Since the information is distributed among the processes and there is no common clock, the processes have to coordinate among themselves to record a global state. Due to the inherent asynchrony in the distributed system, the recorded global state may not have occurred during the execution. However, the recorded global state can be useful in many applications, particularly in detecting stable properties like "program has terminated", "processes are deadlocked", etc. [4].

Chandy and Lamport [4] define the notion of global states and present an algorithm to record a global state in distributed systems with FIFO communication channels. The message complexity of their algorithm is $O(m)$ bits where m is the number of channels. Several algorithms exist in the literature for recording global states both in FIFO and NON-FIFO networks [2,5,9].

Acharya and Badrinath [1] consider the problem of recording a global state in a distributed system that maintains "causal order" in message delivery. There are several algorithms in the literature that implement causal ordered message delivery [3,7,8]. Since causal order property is stronger than FIFO property, global states might be recorded more efficiently in these systems than in systems with FIFO channels.

In this paper, we present a message-optimal algorithm to record global states. The message complexity is $O(n)$ bits. Optimality is achieved

* Corresponding author. Email: sridhar@utdallas.edu.

[1] This research was supported in part by NSF under Grant No. CCR-9110177, by the Texas Advanced Technology Program under Grant No. 9741-036, and by grants from Alcatel Network Systems.

0020-0190/94/$07.00 © 1994 Elsevier Science B.V. All rights reserved
SSDI 0020-0190(94)00055-4

by using some "information" already available for maintaining causal order message delivery. We assume that the underlying causal ordering protocol can be modified to make this information available.

2. Model and definitions

A distributed system is a collection of processes connected by a set of communication channels. Every pair of processes is connected by a point to point logical channel. The channels are bidirectional. Communication channels take an arbitrary but finite amount of time to deliver the messages (i.e., the channels are asynchronous). The execution speeds of the processes are also asynchronous. Communication among the processes is by message passing only. Let $P = \{P_1, \ldots, P_n\}$ be the set of processes. Let C_{ij} represent the channel between P_i and P_j.

An *event* in a process is an action that changes the state of the process. An event may be a send event resulting in sending of a message to one or multiple processes, or a receive event resulting in receiving of a message from a process, or an internal event in which no sending or receiving of a message is involved. Note that *multicasting* (sending a message to multiple processes) is a single atomic send event.

A *process state* is a sequence consisting of the initial state followed by a sequence of events occurred in that process. A *channel state* is a sequence of messages sent along the channel that are not yet received. A *global state* of a distributed system is a collection of process states and channel states. We say that an event e *is in* a global state G if e is in a process state of the global state G. Similarly, a message m *is in* a global state G if m is in a channel state of the global state G. For a message m, let $send(m)$ be the event that corresponds to the sending of m and $recv(m)$ be the event that corresponds to the receipt of m. A global state G is *consistent* if, for a message m sent by P_i to P_j,

m is in G or $recv(m)$ is in G
$\Rightarrow$ $send(m)$ is in G.

A global state G is *complete* if, for a message m sent by P_i to P_j,

$send(m)$ is in G
$\Rightarrow$ m is in G or $recv(m)$ is in G.

The ordering of events in a distributed system is based on the "happened before" relation, denoted by $\rightarrow$, introduced by Lamport [6]. *Causal ordering* of message delivery is preserved if, for any two messages m and m' that have the same destination and $send(m) \rightarrow send(m')$, $recv(m) \rightarrow recv(m')$. An example of causal ordering is presented in Fig. 1(a). Process P_1 sends a message m to P_3, and process P_2 sends a message m' to P_3. Since $send(m) \rightarrow send(m')$, causal ordering requires P_3 to receive m before m'. Note that causal ordering implies FIFO, but FIFO need not imply causal ordering. In Fig. 1(b) FIFO property is satisfied but causal ordering is violated. For any two messages m and m' sent to the same process such that $send(m) \rightarrow send(m')$, causal ordering requires that $recv(m) \rightarrow recv(m')$, whereas FIFO requires this only if m and m' were sent by the same processes. Thus causal ordering is a stronger property than FIFO. In our model, all messages are delivered in causal order.

The performance of the snapshot algorithm is measured by the *communication complexity*. The communication complexity (worst case) is the maximum number of bits transmitted by the snapshot algorithm.

3. Snapshot algorithm

Our snapshot algorithm makes use of the information already available for maintaining causal order of message delivery. For ease of exposition, we refer to the protocol that ensures causal order message delivery as the *causal order protocol*. The snapshot algorithm is initiated by a process by multicasting a *marker* message to all the processes. When the causal order protocol at a process receives a *marker* it saves the information appended to the *marker* to maintain causal ordering. Subsequently, on receiving an application message, the causal order

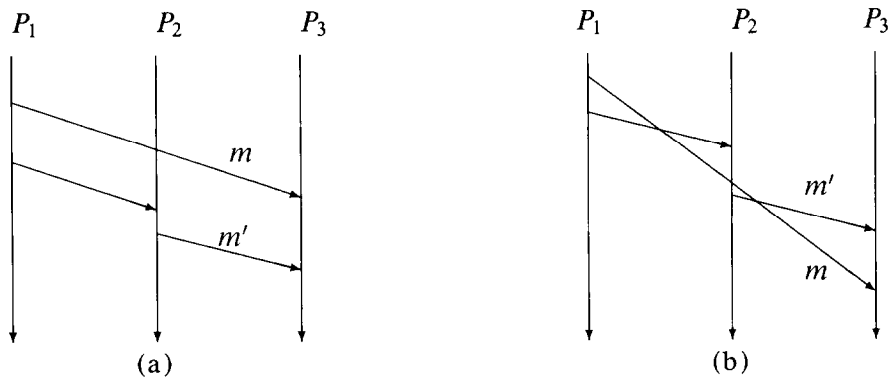

Fig. 1. In (a) causal ordering is maintained. In (b) it is violated.

protocol checks (using the information saved) whether the sending of the *marker* happened before the sending of the application message. If so, it marks the application message *new* and delivers the message to the process. Otherwise it marks the application message *old* and delivers the message to the process. We assume that this modification can be incorporated into the protocol that maintains causal ordering.

The above-mentioned modification can be done if the implementation of causal order protocol is based on vector clocks [8] or based on message counting [7]. Consider the implementation based on message counting. Each process P_i maintains two arrays $DELIV_i[1,\ldots,n]$ and $SENT_i[1,\ldots,n,1,\ldots,n]$. $DELIV_i[j]$ denotes the number of messages from P_j delivered to P_i. $SENT_i[j,k]$ is the number of messages sent by P_j to P_k that P_i knows of. Whenever a message M is sent by P_i, $SENT_i$ is sent with M. Now consider that the causal order protocol at P_j receives the *marker* message, $\langle marker, \mathrm{ST}\rangle$. The causal order protocol saves the value $\mathrm{ST}[i,j]$. On receiving a subsequent message $\langle M, \mathrm{ST}_M\rangle$, if $\mathrm{ST}_M[i,j] \geqslant \mathrm{ST}[i,j]$, then M is marked *new*. Otherwise M is marked *old*. Note that such modification may not be possible in the piggybacking implementation of ISIS CBCAST.

We next present the snapshot algorithm. A process that initiates the algorithm is called an *initiator*. For simplicity, we assume that only one process initiates the algorithm. (Our algorithm can be extended for multiple initiators.)

Steps executed by the initiator.

- The initiator multicasts a *marker* message to all the processes and waits for a *done* message from each process.
- After receiving a *done* message from each process, the initiator multicasts a *terminate* message to all the processes.

Steps executed by a process P_i (including the initiator).

- On receiving a *marker* message, process P_i records its local state, initializes the state of all incoming channels to *null*, and sends a *done* message to the initiator.
- On receiving an application message from P_j, process P_i checks whether the message is marked *old*. If so, process P_i records the message as a part of the channel C_{ji}'s state.
- On receiving a *terminate* message from the initiator, process P_i terminates the snapshot algorithm.

4. Correctness and complexity

Let $marker_i$ and $terminate_i$ denote, respectively, the copies of the *marker* and the *terminate* messages received by P_i. Let $done_i$ denote the *done* message sent by P_i.

Lemma 1. *A message m sent by P_i to P_j after P_i records its local state is not in the global state recorded by the snapshot algorithm.*

Proof. Recall that the *marker* message is multicast by the initiator to all the processes. Let P_i send a message m to P_j after P_i records its state. Clearly, $recv(marker_i) \rightarrow send(m)$. We know that $send(marker) \rightarrow recv(marker_i)$. Therefore, by transitivity, $send(marker) \rightarrow send(m)$. Causal order message delivery ensures that $recv(marker_j) \rightarrow recv(m)$. Therefore, P_j has recorded its state before receiving m. Also, the causal order protocol marks m *new*. Thus, m will not be in the channel C_{ij}'s recorded state. □

Lemma 2. *If a message m is sent by P_i to P_j before P_i records its local state, then $recv(m)$ is in the recorded state of P_j or m is in the recorded state of channel C_{ij}.*

Proof. Consider a message m sent by P_i to P_j before P_i records its local state. If P_j has received m (sent by P_i) before recording the local state, then $recv(m)$ is in the local state of P_j. Assume that P_j receives m after recording its local state. We know that $send(m) \rightarrow recv(marker_i)$. Since P_i sends a *done* message to the initiator after recording its state, $send(m) \rightarrow send(done_i)$. Only after receiving *done* messages from all the processes the initiator multicasts *terminate* message to all the processes. Therefore, $send(m) \rightarrow send(terminate)$. Causal order message delivery ensures that $recv(m) \rightarrow recv(terminate_j)$. The message m will be marked *old* by the causal order protocol, because $send(marker) \not\rightarrow send(m)$. The snapshot algorithm at a process records an application message marked *old* as part of the state of the channel through which it received the message. Therefore, m is recorded in the state of C_{ij}. □

The algorithm terminates since each process eventually receives the *terminate* message from the initiator. When a process receives the *terminate* message it has recorded its local state and the state of all its incoming channels. The causal ordering protocol can stop marking the messages as *new* or *old* when the snapshot algorithm terminates.

Theorem 3. *The global state recorded is consistent and complete.*

Proof. From Lemma 1, the recorded global state is consistent, and from Lemma 2 it is complete. □

Theorem 4. *The message complexity of the snapshot algorithm is $O(n)$ bits, and the algorithm is optimal with respect to message complexity.*

Proof. The initiator sends $2n$ messages (n *marker* and n *terminate* messages) overall, and all other processes send one *done* message each. Clearly, the message complexity is $O(n)$ bits, since each message is of size $O(1)$ bits. The initiator has to inform all the processes to record their respective local states. This requires at least one message bit per process. Thus the lower bound for the message complexity is $\Omega(n)$ bits. Our algorithm, thus, is optimal with respect to the message complexity. □

5. Comparison

We briefly review the Acharya–Badrinath algorithm before we compare it with our algorithm. Their algorithm requires that the underlying causal ordering protocol at each process P_i maintain two arrays, $SENT_i[1,\ldots,n]$ and $RECD_i[1,\ldots,n]$. At any instant, $SENT_i[j]$ denotes the number of messages P_i has sent to P_j till that instant, and $RECD_i[j]$ denotes the number of messages P_i has received from P_j till that instant. An initiator process initiates the snapshot algorithm by multicasting a *token* message to all the processes. After receiving the *token* message, process P_i records its state and replies to the initiator by sending it local state and the two arrays $SENT_i$ and $RECD_i$. The initiator, after receiving replies from all the processes, computes the channels state as sequence of message ids. A channel C_{ij}'s state is given by $\{(RECD_j[i] + 1),\ldots,SENT_i[j]\}$.

The Acharya–Badrinath algorithm is two-phased, one for sending *token*, and the other for sending *reply*. Our algorithm is three-phased, one for sending *marker*, second for sending *done*, and the third for sending *terminate*. However, in the Acharya–Badrinath algorithm, the global state is assembled at the initiator and the state of a channel is recorded as the sequence of message-ids of the in-transit messages. Since the initiator does not know the contents of those messages, their algorithm relies on either the senders keeping a log of the messages sent, or the receivers storing the messages (may include the messages that is not in a recorded channel state) as they arrive [1]. The initiator has to send another $O(n)$ control messages to inform the processes about the states of their incoming channels. Thus, their algorithm is also three-phased if the contents of the channels' states have to be known. In our algorithm, a process records the state of all of its incoming channels similar to Chandy–Lamport algorithm [4]. A process does not have to store a message that is not in a recorded channel state. Also, the computation performed by the initiator in our algorithm is much less compared to their algorithm where the initiator computes ($O(n^2)$ comparisons) the states of all the channels.

The Acharya–Badrinath algorithm uses $O(n)$ messages, but the *reply* messages are long. (The size of a message can be $O(n \log n)$ bits assuming that the number of messages sent by a process is polynomial in n.) The message complexity of our algorithm is $O(n)$ bits. The size of messages we use is always $O(1)$ bits. We achieve low message complexity and reduce the computation overhead performed at the initiator by relying on the ability of causal ordering protocol to mark the application messages as *new* or *old*. This may require some modification to the "code" of the causal ordering protocol. The Acharya–Badrinath algorithm does not require this modification, but it requires that the underlying causal ordering protocol maintain *SENT* and *RECD* arrays [1]. These are the only differences between our assumptions and their assumptions.

6. Conclusion

We have presented an algorithm to record a global state of a distributed system. Our algorithm makes use of the information already available for maintaining causally ordered message delivery. Every process locally records its state and the states of all of its incoming channels. The message complexity of our algorithm is $O(n)$ bits. Our algorithm can be extended for multiple initiators. When a process initiates the process of recording a snapshot, the initiator stamps the *marker* with its *id*. The additional steps performed by the causal order protocol are slightly involved. Now an application message may be marked *new* with respect to one initiator, but it need not be marked *new* with respect to another initiator. So the messages that are marked *new* are also stamped with initiators ids. When a process receives a *new* message stamped with initiators ids, the message is not recorded as part of the channel state in the recordings that correspond to the stamped ids . All other steps of the algorithm are similar to that of the single initiator.

Acknowledgement

We like to thank Professor J. Misra and the anonymous referees for their extensive comments which greatly improved the presentation of this paper.

References

[1] A. Acharya and B. Badrinath, Recording distributed snapshots based on causal order of message delivery, *Inform. Process. Lett.* **44** (1992) 317–321.

[2] M. Ahuja, Flush primitives for asynchronous distributed systems, *Inform. Process. Lett.* **34** (1990) 5–12.

[3] K. Birman, A. Schiper and P. Stephenson, Lightweight causal and atomic group multicast, *ACM Trans. Comput.* **9** (3) (1991) 272–314.

[4] K. Chandy and L. Lamport, Distributed snapshots: Determining global states of distributed systems, *ACM Trans. Comput. Systems* **3** (1) (1985) 63–75.

[5] T. Lai and T. Yang, On distributed snapshots, *Inform. Process. Lett.* **25** (1987) 153–158.

[6] L. Lamport, Time, clocks, and the ordering of events in a distributed system, *Comm. ACM* **21** (7) (1978) 558–565.

[7] M. Raynal, A. Schiper and S. Toueg, Causal ordering abstraction and a simple way to implement it, *Inform. Process. Lett.* **39** (6) (1991) 343–350.

[8] A. Schiper, J. Eggli and A. Sandoz, A new algorithm to implement causal ordering. in: *Proc. 3rd Internat. Workshop on Distributed Algorithms* (Springer, Berlin, 1989) 219–232.

[9] K. Venkatesh, T. Radhakrishnan and H. Li, Global state detection in non-FIFO networks, in: *Proc. 7th Internat. Conf. on Distributed Computing Systems* (IEEE, 1987) 364–370.