To appear in Information Processing Letters

RECORDING DISTRIBUTED SNAPSHOTS BASED ON CAUSAL ORDER OF MESSAGE DELIVERY

Arup ACHARYA and B. R. BADRINATH
Department of Computer Science, Rutgers University, New Brunswick, NJ 08903

This paper presents a simple and efficient algorithm to record a global snapshot of a distributed system, where all messages are sent and delivered in *causal order*. For a system with *N* processes, the algorithm requires *N* control messages to record a distributed snapshot of the system. It is shown that the snapshot recorded represents a *consistent* global state of the system.

Keywords : Distributed system, Global Snapshot, Consistent Snapshot, Causal order.

1 Introduction

A *distributed system* is a collection of processes that communicate by sending messages. It is represented by a directed graph, where nodes represent processes and arcs represent directed communication paths *(virtual channels)* between processes.

The global state of a distributed system consists of the local states of each process, and the state of every channel. Due to the absence of a common clock and the inherent asynchrony in a distributed system, it is impossible to record all the local states at the same instant of physical time. Hence, the terms *global state* or *global snapshot* refer to a set of local states, one state per process/channel, that are recorded in a coordinated fashion at possibly different instants of physical time, but yet capture enough information about the system to be useful.

Existing algorithms to record distributed snapshots [1, 4, 8, 13] show that they are intended for systems with message delivery semantics no stronger than FIFO. On the other hand, there exist a number of recent distributed applications [2, 3, 5, 6, 7], wherein messages are sent and delivered according to a message-ordering protocol that implements a communication abstraction stronger than FIFO message delivery. A natural question to ask therefore, is whether it is any easier to record distributed snapshots in such systems that support more powerful message-ordering abstractions.

Work was supported in part by the National Science Foundation under grant IRI-9010174.

This paper addresses the problem of recording distributed snapshots in a system that supports *causal order* of message delivery. There exist several practical and efficient protocols that deliver messages in causal order [3, 10, 11, 12]. The contribution of this paper is to show that causal order of message delivery considerably eases the problem of recording distributed snapshots. We present a simple algorithm that relies on causal delivery of messages to record a distributed snapshot of a system with N processes. This algorithm requires N control messages. The global snapshot recorded is shown to be a *consistent* snapshot of the system.

The next section defines causal order of message delivery, the components of a global snapshot and the properties it must satisfy to be consistent. Section 3 presents the algorithm to capture a distributed snapshot. In section 4, the recorded snapshot is shown to be *consistent* and compared with the algorithm of Chandy and Lamport [4].

2 Distributed Snapshots

Consider a collection of N processes that communicate with each other by sending messages. There exists a directed logical channel $C_{i,j}$ from a process P_i to any other process P_j. All messages are sent and delivered in causal order. The *global snapshot GS* of the collection of processes P_i, $i \in \{1...N\}$ consists of :

1. $\forall i \in \{1...N\}$
   $local\_state_i \in GS$, where $local\_state_i := <s_i^0\ E_i>$,

i.e, the global snapshot includes a recorded local state $local\_state_i$ for each process P_i. For process P_i with an initial state s_i^0, the state after a sequence of events E_i, is represented as $<s_i^0\ E_i>$. An event is either emission (including *multicast*[2]) or delivery of a message, or a local computation which involves no communication. On occurence of an event, the process state changes from its current state to one of a *set* of possible states, the choice being made non-deterministically.

2. $\forall i \in \{1...N\}, \forall j \in \{1...N\}, state(C_{i,j}) \in GS$
   i.e, the global snapshot also includes the state of every channel $C_{i,j}$, which consists of the *sequence* of messages sent from P_i that have *not been delivered* at P_j.

An event *e* is said to *belong* to a global snapshot *GS*, ($e \in GS$), if the local state recorded for some process P_k is $local\_state_k := <s_k^0 E_k>$, such that $e \in E_k$.

A global snapshot is a *consistent* snapshot of the system, if for any message *m*,

$recv(m) \in GS \Rightarrow send(m) \in GS$

The *causal* order of message delivery is defined as follows. Consider two messages *m1* and *m2*, such that

- *m1* and *m2* are sent to the same destination process, though not necessarily from the same sender process, and
- $send(m1) \rightarrow send(m2)$, where "$\rightarrow$" is Lamport's *happened-before* relation [9].

Then, causal order of message delivery requires that *m2* be delivered after *m1*, i.e., $recv(m1) \rightarrow recv(m2)$. It is easy to see that causal order of message delivery ensures FIFO property of the virtual channels.

3 Recording a consistent distributed snapshot

The snapshot algorithm requires that the underlying causal ordering protocol at each process, P_i, maintain two arrays, $SENT_i[1...N]$ and $RECD_i[1...N]$, where $SENT_i[j]$ contains the number of messages sent from process P_i to P_j, while $RECD_i[j]$ contains the number of messages delivered at P_i, that were sent from P_j. It should be noted that this assumption does not place an additional burden on the underlying causal ordering layers, since this information about the number of messages sent by a process and the number of messages delivered, is maintained as an intrinsic part of causal ordering protocols [3, 10, 11, 12]. For example, the protocols of [3, 11, 12] maintain this information explicitly at every process by the use of *logical clocks*, while the protocol of [10] maintains a directed graph (*context graph*) at each process to represent the causal order of messages. Thus, the same data structures used by these protocols to maintain causal ordering information, can represent the SENT and RECD arrays. On the other hand, each message is appended with sufficient causal ordering information so that it is delivered in the correct order at a recipient. This incurs an overhead in message size and additionally, may introduce a delay in the delivery of a message at a recipient process till all causally preceeding messages have been delivered at that process.

Any process may initiate an execution of the algorithm described below to record a distributed snapshot of the system. Such a process will be referred to as the *initiator*. The messages sent as part of the snapshot protocol are tagged as either *token* or *reply*. The initiator multicasts a *token* message to every process. When a token is delivered at a process, it records its *local snapshot*. The local snapshot recorded consists of the local state of the recording process, and values of its SENT and RECD arrays. Each process replies to the initiator with its local snapshot. It should be noted that all messages delivered in the system are causally ordered, including messages that are exchanged as part of the snapshot protocol.

The_snaphot_algorithm

□ *Token transmission by the initiator process, P_{init} :*

P_{init} multicasts a *token* message to every process in the system, including itself. At P_{init}, the multicast is followed by delivery of the *token* message.

□ *Token delivery at a recipient process P_j, $j \in \{1...N\}$:* On delivery of the *token* message, P_j executes the following actions.

a. P_j records its local snapshot, LS_j, which consists of

- its local state, $local\_state_j$,
- current values of $RECD_j$ and $SENT_j$.

i.e, $LS_j := \{local\_state_j, RECD_j, SENT_j\}$

b. P_j sends *reply*(LS_j) to P_{init}.

□ *Assembling a global snapshot at P_{init} :* The initiator, P_{init}, waits till the delivery of *reply*(LS_j) message from every process P_j. The local states of each process has already been recorded and is available at P_{init}. The channel states are computed by P_{init} as a sequence of message_ids, using the values of SENT and RECD arrays recorded by every process as part

[2] The event of sending a message to multiple processes, is referred to as *multicast*.

of its local snapshot. Thus, the global snapshot *GS* is computed as :

a. $\forall j \in \{1...N\}$, $local\_state_j \in GS$.
$local\_state_j$ is available at P_{init} with the delivery of $reply(LS_j)$ message from P_j.

b. $\forall j \in \{1...N\}$,

$state(C_{init,j}) := \phi$
i.e., the channel $C_{init,j}$ does not contain any in-transit messages. This claim is validated in the next section *(Claim C3)*.

c. $\forall i \in \{1...N\}, \forall j \in \{1...N\}, i \neq init$,

$state(C_{i,j}) := \{(RECD_j[i] + 1),..., SENT_i[j]\}$
i.e., $SENT_i[j]$ represents the number of messages sent by P_i to P_j, before P_i recorded its local snapshot; $RECD_j[i]$ is the number of messages sent from P_i that were delivered at P_j before P_j recorded its local snapshot. Thus, the sequence of messages sent on the channel $C_{i,j}$, whose message-ids range from $RECD_j[i]+1$ to $SENT_i[j]$, are considered to be in-transit on channel $C_{i,j}$ in the global snapshot GS.

Note that the values of the SENT and RECD arrays used above are the recorded values, and not the current values.

It will next be shown that this algorithm captures a *consistent snapshot* of the system at the initiator site.

4 Properties of the distributed snapshot recorded

Claim C1 : *The global snapshot GS computed by the initiator P_{init} is consistent.*
Proof : The global snapshot *GS* is a consistent snapshot of the system, if for any message *m*,

- $recv(m) \in GS \Rightarrow send(m) \in GS$
- or conversely, $send(m) \notin GS \Rightarrow recv(m) \notin GS$

i.e, if the global snapshot does not capture the event of sending a message *m*, then for the snapshot to be consistent, neither should the delivery of *m* be included in the snapshot.

Let the recorded states of two processes, P_k and P_l, that are part of GS, be $local\_state_k$ and $local\_state_l$ respectively, such that $local\_state_k = <s_k^0\ E_k>$ and $local\_state_l = <s_l^0\ E_l>$.

Assume that a message *m* is sent from P_k to P_l, after the local snapshot is recorded at P_k, i.e $send(m) \notin GS$. It remains to be shown that the causal order of message delivery ensures that the local snapshot at P_l is recorded before the delivery of message *m*, i.e., $recv(m) \notin GS$.

Let the copies of the *token* message multicast from P_{init} to all processes, that were delivered at P_k and P_l be, $token_k$ and $token_l$ repectively. By definition of Lamport's *happened-before* relation,

- the event of sending a message *happens-before* the event of its receipt (and its delivery). Thus,
  - $send(token) \rightarrow recv(token_k)$ ..... (I)

Since,

- the local snapshot of a process is recorded on delivery of the *token*, and
- message *m* is sent by P_k after recording its local snapshot, it follows that
  - $recv(token_k) \rightarrow send(m)$ ..... (II)

Since, the *happened-before* relation is transitive, it follows from (I) and (II),

- $send(token) \rightarrow send(m)$

i.e, the event of sending the *token* causally precedes the event of sending *m*. Thus the causal order of message delivery will ensure that, at any process which receives a copy of the *token* and the message *m*, the *token* will be delivered before *m*. Therefore, at process P_l,

- $recv(token_l) \rightarrow recv(m)$

The local snapshot is recorded at P_l on delivery of $token_l$. Since *m* is delivered at P_l after the local state has been recorded, the event *recv(m)* does not belong to $local\_state_l$, and thus,

- $recv(m) \notin GS$

Hence, the global snapshot *GS* recorded represents a consistent snapshot of the system. ○

It has been shown that for any message *m*, if its *send* event does not belong to the global snapshot, neither does the event of its delivery, *recv(m)*, i.e. the recorded process states are consistent. It is next shown that the channel states are recorded correctly.

Claim C2: *A message m sent from process P_k to P_l, after the local snapshot has been recorded at P_k cannot be included in the sequence of in-transit messages that constitute the recorded state of channel $C_{k,l}$.*
Proof : The state of $C_{k,l}$ is computed as the sequence of messages-ids $\{RECD_l[k]+1,..., SEND_k[l]\}$.[3] Since, *m*

[3] $RECD_l$ and $SEND_k$ refer to the values that were recorded as part of the local snapshots at P_l and P_k respectively.

was sent by P_k after recording its local snapshot, *m* has a sequence number greater than $SEND_k[l]$, and is therefore, not included in the state of channel $C_{k,l}$. ◯

The snapshot algorithm recorded the state of any channel, outgoing from the initiator, to be empty. That claim is now shown to be valid.

Claim C3 : *The snapshot protocol records a consistent global snapshot such that all outgoing channels, $C_{init,\,j}$ from the initiator process, P_{init} to any process P_j, does not contain any in-transit messages.*
Proof : Consider a message *m* sent by P_{init} to any process P_j, before P_{init} sent a *token* message. Since *send(m)* → *send(token)*, the causal order of message delivery ensures that *m* is delivered before *token* at P_j. Thus, the state of channel $C_{i,j}$, ougoing from P_{init}, will contain no in-transit messages in the global snapshot recorded. ◯

The initiator site sends a token to every site to *record* the local snapshots. Therefore N *token* messsages are required to *record* the distributed snapshot. Each local snapshot is sent back to the initiator site through a *reply* message. Thus, another N messages are required to organise the local snapshots into a distributed snapshot.

It is instructive to compare the protocol presented above with that of the Chandy-Lamport protocol [4]. Their protocol requires the channels to deliver messages in a FIFO order. It incurs an overhead of $|E| = O(N^2)$ messages to capture a consistent snasphot, where *E* is the set of channels. In the Chandy-Lamport protocol, the initiator process sends a *marker* message on all its outgoing channels. A process on receiving a *marker* for the first time (on any of its incoming channels), records its local state, and then sends out a *marker* message on all of its outgoing channels. Thus, a *marker* message is eventually sent out on every channel. The state of a channel is recorded by the process at the receiving end of the channel, as the sequence of messages received on that channel after the process recorded its local state and before it received a *marker* on that channel. So, for every channel, the process at the receiving end of the channel, stores the messages that are considered to be in-transit in the recorded global snasphot.

Our protocol relies on causal order of message delivery, to record a consistent snapshot of a system of N processes with N messages. However, the two protocols differ in recording the states of the channels. In the Chandy-Lamport protocol, the process at the receiving end of every channel, collects the *messages* that are considered to be in-transit in the recorded global snasphot. In our protocol, the state of a channel is computed by the initiator, as the sequence of *message-ids* of the in-transit messages. The initiator does not know directly the contents of those messages. To find the contents, it must rely either on senders having kept copies of the messages they sent, or on receivers collecting those messages as they (subsequently) arrive; the initiator then needs to send a second round of control messages to all the processes, incurring an additional overhead of *O*(N) messages, to collect the contents of the in-transit messages.

5 Conclusion

For distributed systems that obey causal delivery of messages, this paper presented a simple protocol to record a distributed snapshot. It was shown that the distributed snapshot recorded, is a consistent snapshot of the system.

Acknowledgements

The authors wish to thank Prof. J. Misra and the anonymous referees for their comments on an earlier version of the paper.

Bibliography

[1] M. Ahuja. *flush* primitives for asynchronous distributed systems. *Information Processing Letters*, 34(1):5–12, 1990.

[2] K. Birman and T. Joseph. Reliable communications in presence of failures. *ACM Trans. Comput. Systems*, 5(1):47–76, 1987.

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

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

[5] A. Heddaya, M. Hsu, and W. Weihl. Two phase gossip: Managing distributed event histories. *Information Sciences*, 49:35–57, 1989.

[6] N. Krishnakumar and A. J. Bernstein. Bounded ignorance in replicated systems. *Proceedings of the 10th ACM Symposium on Principles of Database Systems*, 1991.

[7] R. Ladin, B. Liskov, and L. Shira. Lazy replication: Exploiting the semantics of distributed services. *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, 1990.

[8] T.H. Lai and T.H. Yang. On distributed snapshots. *Information Processing Letters*, 25(3):153–158, 1987.

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

[10] L. L. Peterson, N.C. Bucholz, and R. D. Schlichting. Preserving and using context information in interprocess communication. *ACM Trans. Comput. Systems*, 7(3), 1989.

[11] M. Raynal, A. Schiper, and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Information Processing Letters*, 39(6), 27 September 1991.

[12] A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. In *Proc. 3rd Intl. Workshop on Distributed Algorithms*, pages 219–232. Springer Verlag LNCS 392, 1989.

[13] K. Venkatesh, T. Radhakrishnan, and H.F. Li. Global state detection in non-fifo networks. *Proceedings of the 7th Intl. Conf. on Distributed Computing Systems*, 1987.