J. Parallel Distrib. Comput. 73 (2013) 621–629

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Efficient distributed snapshots in an anonymous asynchronous message-passing system

Ajay D. Kshemkalyani [a,*], Mukesh Singhal [b]

[a] Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607, USA

[b] Computer Science and Engineering, University of California at Merced, Merced, CA 95343, USA

ARTICLE INFO

Article history:
Received 7 April 2012
Received in revised form
18 December 2012
Accepted 11 January 2013
Available online 18 January 2013

Keywords:
Distributed computing
Anonymous systems
Global snapshot
Causality
Global state
Termination detection

ABSTRACT

We present a global snapshot algorithm with concurrent initiators, with termination detection in an anonymous asynchronous distributed message-passing system having FIFO channels. In anonymous systems, process identifiers are not available and an algorithm cannot use process identifiers in its operation. Such systems arise in several domains due to a variety of reasons. In the proposed snapshot algorithm for anonymous systems, each instance of algorithm initiation is identified by a random number (nonce); however, this is not used as an address in any form of communication. In the algorithm, each process can determine an instant when the local snapshot recordings at all the processes have terminated. This is a challenging problem when an algorithm cannot use process identifiers and a process does not know the number of processes in the system or the diameter of the network and cannot use a predefined topology overlay on the network, because there is no easy way to identify the global termination condition. The message complexity of our algorithm is (cn^2), where c is the number of concurrent initiators and n is the number of processes in the system, which is much better than that of the algorithm by Chalopin et al. (2012) [6]. Further, the algorithm by Chalopin et al. also requires knowledge of the network diameter.

1. Introduction

Snapshot recording in distributed systems finds applications in several important problems, such as checkpointing and rollback recovery, detection of stable properties, and debugging of distributed programs. Existing algorithms for snapshot recording explicitly use process identifiers for their operation and hence they do not work for anonymous systems. In this paper, we present a global snapshot recording algorithm for anonymous systems that works with concurrent initiators, and also detects termination of the global state recording activities.

In an anonymous system, an algorithm cannot use process identifiers [3]. In such systems, a process may be unwilling to disclose its process identifier due to privacy reasons or processes in the distributed system may be coming together from different domains, and there is no guarantee that the process identifiers are unique. In some cases, the system may be dynamic with users/processes continuously joining or leaving the system and it may not be possible to know who the current users/processes are. Some distributed applications like web servers and peer-to-peer file systems require preserving the anonymity of users, and forbid the use of any form of identity in the distributed computation [9]. In systems such as sensor networks, the mass-produced sensing agents are generally not equipped with identifiers.

The global snapshot problem requires recording the states of all the processes and all in-transit messages in channels in a consistent manner [7]. A snapshot consists of $\langle \bigcup_i \{LS_i\}, \bigcup_{i,j} \{SC_{i,j}\} \rangle$, where LS_i is the local state of process P_i and $SC_{i,j}$ is the state of the channel $C_{i,j}$ from process P_i to process P_j. The recording should be consistent so as to reflect a system state that could have been reached in the execution. Chandy and Lamport presented the seminal algorithm to record a consistent snapshot [7]. It was followed by many algorithms (e.g., [1,2,4,10,13–15,18]) that modified the system assumptions; see [11,12] for a survey. However, there are only two algorithms that can deal with concurrent initiators — the Spezialetti–Kearns algorithm [19] and the very recent Chalopin et al. algorithm [6]. The Chalopin et al. algorithm is designed to record snapshots in an anonymous system.

There are two typical phases in obtaining a global snapshot in traditional snapshot recording algorithms: locally recording the snapshot at every process (which is the main challenge) and distributing the recorded global snapshot to all the initiators.

* Corresponding author.
E-mail addresses: ajay@uic.edu (A.D. Kshemkalyani), msinghal@ucmerced.edu (M. Singhal).

0743-7315/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2013.01.003

Table 1
Comparison of algorithms. n is the number of processes, c is the number of concurrent initiators, β is the diameter of the network.

Feature	Chandy–Lamport algorithm [7]	Spezialetti–Kearns algorithm [19]	Chalopin et al. algorithm [6]	Proposed algorithm
Message complexity	$O(n^2)$	$O(n^2)$	$O(cn^2 + \beta n^3)$	$O(cn^2)$
Message space complexity	$O(n^2)$	$O(cn^2)$	$O(cn^2 + \beta n^3)$	$O(c^2n^2)$
Concurrent initiators allowed	No	Yes	Yes	Yes
Global termination detection by all nodes	No	No	Yes	Yes
Anonymity	No	No	Yes	Yes
Other assumptions	None	None	Diameter β must be known to all nodes	None

In the Chandy–Lamport algorithm, the cost of the first phase is $O(n^2)$ messages, while the cost of the second phase is $O(n^3)$ messages, where n is the number of processes in the system. All processes are informed of the state of every other process in the second phase to ensure that all processes get the desired information about the global snapshot. This algorithm cannot handle concurrent initiators.

In the Spezialetti–Kearns algorithm [19], in the first phase, each process takes a local snapshot (local state and incident channel states) and collects information about neighboring concurrent initiators using process identifiers. The cost of this phase is $O(n^2)$ messages. In the second phase, the collected information is distributed to all initiator processes using process identifiers for direct and transitive communication. When there are c concurrent initiators, the cost of the second phase is $O(cn^2)$ messages. The Spezialetti–Kearns algorithm uses process identifiers explicitly, and hence, it is not suitable for anonymous systems.

In this paper, besides requiring processes to collectively record a global snapshot, we require that all the processes identify an instant in physical time when the snapshot recording has completed at all the processes in the system. This is useful for applications when processes want to output the result of the snapshot recording to the environment only after global termination of the snapshot recording algorithm. In another situation, an application may want to overwrite its latest checkpoint with the recorded local snapshot state only when it knows that all the processes have taken their local snapshot.

Problem Statement. Record a global snapshot with concurrent initiators in an anonymous asynchronous distributed message-passing system, where each process can determine an instant when the local snapshot recordings at all the processes have terminated, using a distributed algorithm. We assume that there is no information available about the number of processes in the system or the diameter of the network, and no predefined static topology overlay is available.

This is really the concurrent snapshot initiation problem, and the snapshot termination detection problem bundled together as a single problem. Termination detection of distributed computations has been studied independently in many works [5,12,17], but not in conjunction with the snapshot recording problem, except in the algorithm by Chalopin et al. [6]. The Chalopin et al. algorithm records a global snapshot under concurrent initiators and also determines the global termination of the snapshot recording activity in an anonymous system, by superimposing the termination detection algorithm of Szymanski et al. [20] on the Chandy–Lamport snapshot recording algorithm. Their solution has two main drawbacks — it requires each node to know the diameter β of the network, and it has a high message complexity of $O(n^3\beta)$. We note that the authors [6] did not provide any statement or analysis of their algorithm's message complexity.

The proposed algorithm to record a global snapshot with concurrent initiators and detect termination of the snapshot recording activities in an anonymous system works as follows. The algorithm is divided into two phases: in the first phase, each process records a local snapshot and in the second phase, the termination of all the local snapshot recordings is detected. In particular, when a process invokes the request for a snapshot, it generates a unique one-time random number, called nonce, and propagates this nonce with its request. Each other process in the system gets associated to exactly one nonce and all processes associated to the same nonce will dynamically form a spanning tree on-the-fly, whose root is the process that invoked the snapshot. Once each process has collected certain information on all incoming links/channels, its local snapshot is determined. Then, for each spanning tree, the information propagates from the leafs to the root to collect certain information about the termination of the global snapshot recordings. Creating the spanning tree allows processes to propagate the information towards the process that invoked the snapshot without knowing it. Once the root node has assembled this information, it generates a "terminated" message to propagate this information in the system. Systemwide, all the nodes participate in an ingenious termination detection algorithm to detect global termination using the information on the "terminated" messages, about all the nonces they are aware of in their local views.

Thus, the paper makes two main contributions:

1. The first is the idea to use nonce to ensure that processes correctly and efficiently handle the messages corresponding to all concurrent snapshot initiations and track information for global termination detection. Specifically, when a node has already received a snapshot recording message containing a nonce, (a) it does not propagate the snapshot recording messages corresponding to other nonce/initiators, but (b) it tracks the presence of these other nonces received from concurrent initiators for use in global termination detection.
2. The second contribution is to present a novel algorithm to detect global termination that works even though processes are anonymous, and without using any information about the number of processes in the system or the network diameter or any predefined static topology overlay.

Table 1 compares the performance and features of the algorithms surveyed above. The message complexity gives the total number of messages. The message space complexity gives the total space requirement (in integers) of all the messages in the solution. For a fair comparison of all the algorithms, we consider only the cost of recording the local snapshots in the Chandy–Lamport algorithm, and the cost of recording the local snapshots and collecting the information about the various concurrent initiators in the Spezialetti–Kearns algorithms. Thus, we do not consider their costs of assembling and/or distributing the recorded local snapshots. For the Chalopin et al. algorithm and the proposed algorithm, costs include the costs of recording the local snapshots and of global termination detection.

The rest of the paper is organized as follows: In Section 2, we present a system model and define notations. In Section 3, we discuss the challenges in solving the problem and present an outline

of our algorithm. In Section 4, we present a snapshot recording and global termination detection algorithm for anonymous asynchronous distributed message-passing systems. In Section 5, we prove the correctness of the algorithm. In Section 6, we analyze and discuss its performance. Section 7 contains concluding remarks.

2. A system model

We assume an anonymous asynchronous distributed message-passing system where processes may run identical code and cannot be identified in any communication by process identifiers. There does not exist a common memory or a common clock and all communication occurs by message passing only. A process in the system can communicate only on the ports of its incident channels to its immediate neighbors, which are identified using local port numbers. A process can only use port numbers of its incident channels to identify neighbors. The ports at a process v are denoted as $C_1, C_2, \ldots, C_{degree_v}$, where $degree_v$ is the number of immediate neighbors of process v. In addition, there is a special port C_0 at each process that is used for sending a message to itself. Messages are sent on C_0 only when explicitly stated so.

The system is modeled as a graph (N, E), where N is the set of processes and E is the set of links or channels. We assume bidirectional channels connecting the processes. Each port is thus bidirectional. We also assume failure-free execution like the Chandy–Lamport algorithm and all other snapshot recording algorithms — thus, processes and links do not crash during algorithm execution. Communication over the channels is FIFO. A process does not know the number of processes in the system or the diameter of the network.

We also assume that during the execution of the algorithm, processes do not join or leave the system. If a process were to join the system during the execution of an instance (invocation) of the algorithm, it will interfere with the correct operation because the process's data structures may not be properly initialized. If a process were to leave the system during the execution of an instance (invocation) of the algorithm, it will interfere with the correct operation because it may break the spanning tree that the algorithm dynamically built or messages may not get forwarded properly. A node that wants to join or leave can wait until the execution is over.

Recording of a local snapshot at a process involves recording of the local state of the process, and recording of the states of all incoming channels at that process. A global snapshot is a consistent collection of all local snapshots [7]. For sake of presentation, we assume that every initiator invokes global snapshot collection only once; however, our algorithm can be naturally extended to recording repeated snapshots in the face of concurrent initiators by using a new set of variables for every subsequent instance of the global snapshot initiation and a *snap_no* variable on messages to identify the instance of the global snapshot.

3. Towards global snapshot recording for anonymous systems

3.1. Challenges and basic idea

In anonymous systems, process identifiers are not available for the operation of an algorithm and for the message exchanges with other processes [9]. This complicates the design of snapshot algorithms under multiple concurrent initiators when termination detection of the algorithm by every process is required in anonymous systems. We identify two main challenges.

1. (Recording with concurrent initiators.) How to record the distributed snapshot with concurrent initiators in an anonymous system becomes a challenge because processes cannot use process identifiers of initiators to determine whose snapshot collection recording request a message represents; neither is there the knowledge of the number of concurrent initiators.
2. (Global termination detection.) How to let each process detect global termination of the distributed snapshot recording algorithm in an anonymous system becomes a challenge because an algorithm cannot use process identifiers for processes to communicate with each other, and a process does not know the total number of processes or the number of concurrent initiators in the system or the diameter of the network, and cannot use a predefined static topology overlay. Thus, there is no easy way to identify the global termination condition.

Recording with concurrent initiators

We address the first challenge in the following manner. Each of the unknown number of initiators simply diffuses its markers in the system, akin to the Chandy–Lamport algorithm. With concurrently initiated diffusions, markers from different initiators will "cross" each other in opposite directions along some edges. All markers are treated alike, irrespective of who the initiator was, in the recording of the local states and channel states. Logically though, the system can be viewed as being partitioned into different regions or domains. Within each domain/region, the local snapshots are clearly consistent. The consistency of local state recordings at the two ends of an inter-domain edge is guaranteed because the recordings are concurrent (due to the markers on the inter-domain edge having crossed each other in transit in opposite directions); inter-domain channel states are also recorded correctly (consistent with the local state recordings) because all markers are treated alike. Hence, snapshots of each pair of adjacent regions are also consistent with respect to each other.

Global termination detection

To address the second challenge, a first approach might be to let each process, after its local snapshot recording termination, initiate a diffusion of "terminated" messages in the system informing other processes that it is terminated. This incurs a cost of n^3 messages. However, this approach will not work because each process needs to be uniquely identified, and each process should also know when to stop expecting more "terminated" messages (this requires knowledge of n). A second approach might be for each process to circulate a "terminated" message around a predefined ring and wait until it gets back its own initiated message. However, this approach will not work because each process needs to be uniquely identified, and the approach also uses a predefined overlay topology. We could choose an appropriate termination detection algorithm from the literature (see surveys in [12,17]); however, no algorithm seems suitable because they all require knowledge of n or of the network diameter β or superimpose a static overlay topology such as a ring or a tree, or have high serial time overhead.

We propose a two-pronged approach.

- First, we detect termination of the snapshot recordings within each region/domain, at the initiator within that region. This does not require any added information because we exploit the spanning tree naturally induced by the edge along which the first marker is received by each node. As part of this step, we also collect at each initiator, information about each neighboring region. This requires each region to be colored uniquely. To accomplish this, we let each initiator choose a unique color for itself.

Table 2
Messages used in the algorithm.

Message	Sender	Trigger	Receiver	Phase	Purpose
MARKER	Each node	Initiator, others on receiving first MARKER	Each neighbor	I	Initiate state recording; record states of channels
ACCEPT	Child node in spanning tree	On receiving a MARKER	Parent node which sent it a MARKER	I	Identify spanning tree edges
REJECT	Child node in spanning tree	On receiving a MARKER	Non-parent node which sent it a MARKER	I	Identify non-spanning tree edges
INSWEEP	Non-root tree node	Receive INSWEEP from all children nodes	Parent node	II	Collect colors of neighboring regions
TERMINATE	Each node	Initiator on receiving INSWEEP from all children nodes, others on receiving TERMINATE message	Each neighbor	II	Detect global termination of all local snapshot recordings

- Next, the initiators execute a distributed computation to let each process in the system know of the global termination of the snapshot recordings within each region. The challenge here is to accomplish this without knowing the total number of concurrent initiators (regions), and restricting the available information to the information about the color of a region and colors of its neighboring regions. This idea is discussed further in Section 3.2.

Note that, to handle global termination detection of the snapshot algorithm with concurrent initiations, we do not necessarily need the identifiers of the initiators. We only need a capability to uniquely identify (color) each of the concurrent initiations of the snapshot algorithm. We propose to identify each instance of algorithm initiation by a process by a unique one-time number generated on-the-fly, called a *nonce*. Nonces are randomly selected from a very large state space (e.g., 128 bits) to maintain uniqueness and are frequently used in network security to foil "replay attacks". We do not use a nonce of an initiator as an address in any form of communication in the algorithm.

Though randomly chosen, the nonces are guaranteed to be unique with an extremely high probability and serve the purpose of providing a unique identity (color) to each initiator. When nonces are drawn from a 128-bit field, a simple probabilistic analysis using the well-known "birthday paradox" shows that even with 100 concurrent initiators in a very large-scale system, the probability of a collision is nearly zero [16]. In the extremely low probability that 2 nonces are identical, a process may deduce that global termination has occurred even though it has not occurred.

3.2. Algorithm outline

For global snapshot recording, marker messages which carry the nonce (color) of the initiator are diffused through the system. We handle concurrent snapshot initiations efficiently by suppressing redundant snapshot collections in the following manner: a process does not take a snapshot or propagate a snapshot request initiated by a process if it has already taken a snapshot in response to some other snapshot initiator. When a process receives the first marker for an initiator, it notes the nonce in the marker and it belongs to the domain (or region) of this initiator for this snapshot initiation. Later when this process receives a marker on an incoming port which is carrying the nonce of a different initiator,[1] the process detects a concurrent initiation of the snapshot algorithm by a different initiator and the sender of the marker lies in a different domain. The process does not take a new snapshot for this marker and does not propagate this marker. However, when a process receives a marker for a different initiator on an incoming port, it completes the recording of the state of the corresponding incoming channel. Thus, when the snapshot algorithm is invoked by multiple concurrent initiators, the system gets partitioned into multiple domains/regions and snapshots recorded in multiple domains can be combined to obtain a consistent global snapshot. (See Theorem 1 in Section 5 as to why the global snapshot across multiple domains is consistent.)

Termination detection of the algorithm by every process is a challenge when the process identifiers are not available. We develop a termination detection scheme that entails the following actions: First, an initiator uses marker messages to build a spanning tree on all processes/nodes in its domain. Processes in the domain propagate intra-domain termination related information on the spanning tree to enable the initiator to detect the termination of local snapshot activities at all the processes in its domain. The colors of neighboring domains are also collected along with the intra-domain local snapshot recording termination information. Then a process detects the global termination of the snapshot recording algorithm by having each initiator broadcast to every process the nonce of its domain and the nonces of concurrent initiations in its neighborhood that it knows about. Since we do not know either n (the number of processes in the system) or c (the variable number of concurrent initiators), we are still faced with a big challenge, viz., that of knowing when to stop expecting more broadcast "termination" messages. The information about nonces of domains in which the snapshot recording has completed is used in an ingenious way, using the fixed point theory on sets, to infer global termination of the snapshot recording algorithm.

4. The algorithm

Algorithms 1 and 2 show the two phases of the snapshot recording and termination detection algorithm, respectively. The types of messages used in the 2 phases are described in Table 2.

1. Phase I (Algorithm 1): In Phase I, an initiator generates a unique one-time number (a nonce), which uniquely identifies this initiation, and the processes collectively record a global snapshot by propagating marker messages through the distributed system. Marker messages carry the nonce (color) of the initiator. The purpose of the nonce is to differentiate itself from other concurrent initiations. The nonce is not used to identify processes or to send messages using the nonce as the destination. A spanning tree induced by the markers within a region is also explicitly identified for use in Phase II.

   When a process receives the first marker message, it records its local state, sends out a marker message on each outgoing port, and stores the nonce of the initiator, received in the marker message, in its variable *region_color* to denote that it belongs to the domain/region of this initiator. When a process receives a subsequent marker for a different initiator, it includes the nonce in the marker into its set *Neighborhood_Color_Set*

[1] Such a process lies on the boundaries of two adjacent domains/regions.

to capture this concurrent initiation. *Neighborhood_Color_Set* tracks colors of neighboring regions of the region under consideration. The processing of a marker is along the lines of the Chandy–Lamport algorithm, with the following change: A spanning tree is identified in the sub-graph of nodes having the same *region_color*. The *parent* of a node in the spanning tree is the node (port) from which the first MARKER of this color is received. An ACCEPT or a REJECT message is sent in response to the MARKER message, depending on whether this is or is not the first MARKER of this nonce/color seen, respectively. If an ACCEPT message is received, the sender node is a child of the receiver node in the spanning tree; if a REJECT message is received, the sender node is not a child of the receiver node in the spanning tree. These messages are useful to build the spanning tree used in Phase II (Algorithm 2) to construct *Neighborhood_Color_Set* at the initiator, and to detect termination of the local snapshot recordings at all the nodes within the region. After local snapshot recording at a process has completed, it calls the procedure LOCAL_SNAP_COMPLETE, given in Algorithm 2.

2. Phase II (Algorithm 2): This phase consists of two subphases.
   (a) In the first subphase, an inward sweep from the leaf nodes of the spanning tree within each region (i.e., processes on the boundaries of two adjacent domains/regions) towards the initiator (root) of that region is performed along the spanning tree. As leaf nodes in a spanning tree execute LOCAL_SNAP_COMPLETE, they propagate INSWEEP message towards the root of the spanning tree, and use the variable *Neighborhood_Color_Set* to accumulate the nonces/colors of the neighborhood regions. At the initiator node, *Neighborhood_Color_Set* is collected. In LOCAL_SNAP_COMPLETE, an initiator determines termination of all the snapshot recording activities within its region, and determines the colors of its neighborhood regions.
   (b) In the second subphase, termination of all the snapshots within a region is broadcast in the system using the TERMINATE message, and all the nodes collectively determine when the global snapshot has terminated. For this, the variables *Universal_Color_Set*, *Terminated_Color_Set*, and *Neighborhood_Color_Set* are used. *Universal_Color_Set* tracks all the colors (instances of initiations) seen. This includes concurrent initiations. *Terminated_Color_Set* tracks colors of regions in which snapshot recording has terminated. When a node becomes aware of a newly terminated region with a nonce x', it adds x' on the TERMINATE message to *Terminated_Color_Set*. It also adds x' and *Neighborhood_Color_Set* (contained in the parameter *NCS*) on the TERMINATE message to the *Universal_Color_Set*. *Universal_Color_Set* now contains colors of all terminated regions and the colors of their neighbors. If *Universal_Color_Set* equals *Terminated_Color_Set*, then global termination in all regions is detected. The equality implies that fixed point of "the neighboring regions of the terminated regions have also terminated" has been reached.

There is a major difference from the Spezialetti–Kearns algorithm. In Phase II, the Spezialetti–Kearns algorithm uses the process identifiers of the concurrent initiators to exchange collected snapshots among the concurrent initiators. However, we use an anonymous algorithm wherein we rely on the three sets *Terminated_Color_Set*, *Universal_Color_Set* and *Neighborhood_Color_Set* to detect global termination of the snapshot recording algorithm. Note that we do not use process identifiers or colors to direct communication in the algorithm. All communication is based on the local neighborhood view obtained through the local ports. Different invocations of the algorithm by the same node can use different nonces.

Broadcasting the TERMINATE message

Within each region, TERMINATE messages are broadcast in Algorithm 2 on the spanning tree created in Algorithm 1. Across regions, the TERMINATE messages are sent on all the incident inter-region edges, as identified in the variable *Neighboring_Region_Ports*. More specifically, TERMINATE messages are initiated by the initiator in each region, in LOCAL_SNAP_COMPLETE. An initiator broadcasts the TERMINATE message along spanning tree edges in its region. Each node that receives TERMINATE forwards it along the remaining tree edges in its region, identified by $Children \cup \{parent\} \setminus \{sender\}$, and across inter-region edges, identified by the local port numbers in the local variable *Neighboring_Region_Ports* (line 21 of Algorithm 1). When a node in the adjacent region receives this TERMINATE message, it also forwards it along the (remaining) tree edges in its region and along any inter-region edges. So in this adjacent region, the TERMINATE message propagates along the tree edges, and can hop across to further distant regions.

Although there is a spanning tree within each region, these spanning trees cannot be combined easily to give a global spanning tree. (This can be done by using an anonymous adaptation of the Gallager–Humblet–Spira MST algorithm [8], but that requires additional phases, incurring additional delay.)

Repeated snapshots

Several problems such as checkpointing and rollback recovery and detection of stable properties, require repeated collection of global snapshots which is achieved by serial invocations of the snapshot algorithm. The nonce chosen by each initiator in each serial invocation can be chosen dynamically on-the-fly. Serial invocations of the algorithm can be assigned serial snapshot numbers and a new set of data structures can be associated with each invocation. An additional parameter such as *snap_no* can be used on all messages to facilitate the variable to track the snapshot number. Serial invocations of the algorithm, even by the same initiator, will have distinct sets of data structures at the nodes for the different serial (but possibly overlapping in time) instances of the global snapshot.

Distributing the snapshot

We did not require processes to distribute the local snapshots recorded; however, we required that in the algorithm, all the processes identify an instant in physical time when the snapshot recording has completed at all the processes in the system. If the local snapshots are also to be distributed throughout the system, this can be easily done by our algorithm. All the nodes would select nonces for each instance of the snapshot algorithm. While collecting the termination-related information using the INSWEEP messages in a domain, the locally recorded snapshots would also be collected. These could then be distributed using TERMINATE messages.

5. Correctness proof

There are two components of the proof: consistency of the recorded states, and detection of global termination.

5.1. Consistency

We give a proof of the consistency of the recorded snapshot in Theorem 1.

Theorem 1. *The global snapshot recorded by processes across multiple domains/regions is consistent even when there are multiple concurrent initiators.*

Algorithm 1 Recording distributed snapshots and constructing a spanning tree in each domain in an anonymous system under concurrent initiators. Code at node i.

set of int: *Children*, *Unrelated* := $\emptyset$
set of int: *Neighbors* := set of outgoing ports to neighbors
set of int: *Neighboring_Region_Ports* := $\emptyset$
int: x, *region_color*
int: *parent*
set of int: *Universal_Color_Set*, *Terminated_Color_Set*, *Neighborhood_Color_Set* := $\emptyset$

(message types)
MARKER, ACCEPT, REJECT, INSWEEP, TERMINATE

1. When a node wants to initiate snapshot recording:
2. record local state
3. *region_color* := generate a nonce x
4. send a MARKER(x) on each (outgoing) port
5. *parent* := $\top$
6. *Universal_Color_Set* := *Universal_Color_Set* $\bigcup\{x\}$
7. initialize state(C_k) for all (incoming) ports C_k to empty set

8. When a node receives MARKER(x') on an (incoming) port C_j:
9. **if** P_i has not recorded its state **then**
10. record local state
11. record state(C_j) as the empty set
12. send a MARKER(x') on each (outgoing) port
13. send ACCEPT on (outgoing) port C_j
14. *region_color* := x'
15. *Universal_Color_Set* := *Universal_Color_Set* $\bigcup\{x'\}$
16. *parent* := j
17. **else**
18. record state(C_j) := set of messages received along this (incoming) port after local state was recorded and until this MARKER was received
19. send REJECT on (outgoing) port C_j
20. **if** *region_color* $\neq x'$ **then**
21. *Neighborhood_Color_Set* := *Neighborhood_Color_Set* $\bigcup\{x'\}$
22. *Neighboring_Region_Ports* := *Neighboring_Region_Ports* $\bigcup\{j\}$

23. When a node receives ACCEPT on an (incoming) port C_j:
24. *Children* := *Children* $\bigcup\{j\}$
25. **if** *Children* $\bigcup$ *Unrelated* = *Neighbors* $\setminus$ {*parent*} **then**
26. // node has received a MARKER on all (incoming) ports;
27. LOCAL_SNAP_COMPLETE

28. When a node receives a REJECT on an (incoming) port C_j:
29. *Unrelated* := *Unrelated* $\bigcup\{j\}$
30. **if** *Children* $\bigcup$ *Unrelated* = *Neighbors* $\setminus$ {*parent*} **then**
31. // node has received MARKER on all (incoming) ports;
32. LOCAL_SNAP_COMPLETE

Proof. Each of the unknown number of concurrent initiators simply diffuses its markers containing its nonce in the system. With concurrently initiated diffusions, markers from different initiators will "cross" each other in opposite directions along some edges. All markers are treated alike, irrespective of who the initiator was, in the recording of the local states and channel states. When a process receives the first marker, irrespective of the nonce on the marker, it records its local state and propagates the marker along the other incident channels. Thus, all processes will record their local states.

Next, we show that all messages are recorded correctly and consistently. We term messages sent by a process before the local state recording as prerecording messages and messages sent after the local state recording as postrecording messages. When a process records its local state (on receiving the first marker along some channel), it forwards the marker on all other incident channels.

- All prerecording messages are recorded in either the local state or the channel state at the receiver.

  Due to FIFO nature of each channel, any prerecording message the process sent would reach the receiver before the marker on that channel, and would either (i) be included in the receiver's local state if the marker along this channel is the first marker it receives, or (ii) be included in the channel's (incoming) state if the marker along this channel is not the first marker it receives.
- No postrecording message is recorded in either the process state or the channel state at the receiver.

  Due to FIFO nature of each channel, any postrecording message the process sent would reach the receiver after the marker on that channel. When a postrecording message arrives, the receiver would have already recorded its local state and the state of that (incoming) channel; thus, neither of those states would contain a postrecording message.

A process records a local snapshot as soon as it receives the first marker, and finishes recording a channel state as soon as it receives a marker on that channel. Therefore, it does not record

Algorithm 2 Termination detection of distributed snapshot activities in an anonymous system under concurrent initiators. Code at node i.

```
1.  LOCAL_SNAP_COMPLETE:
2.  if Children = ∅ then
3.      if parent ≠ ⊤ then                                   // non-initiator leaf node
4.          send INSWEEP(Neighborhood_Color_Set) on (outgoing) port C_parent
5.      else                                                 // initiator leaf node
6.          send TERMINATE(x, Neighborhood_Color_Set) to itself on (outgoing) port C_0
7.  else
8.      await INSWEEP from each (incoming) child port C_j, where j ∈ Children
9.      for each INSWEEP(NCS) received do
10.         Neighborhood_Color_Set := Neighborhood_Color_Set ⋃ NCS
11.     if parent ≠ ⊤ then                                   // non-initiator non-leaf node
12.         send INSWEEP(Neighborhood_Color_Set) on (outgoing) port C_parent
13.     else                                                 // initiator non-leaf node
14.         send TERMINATE(x, Neighborhood_Color_Set) to itself on (outgoing) port C_0

15. On receiving TERMINATE(x', NCS) on (incoming) port C_j:
16. if x' ∉ Terminated_Color_Set then
17.     Universal_Color_Set := Universal_Color_Set ⋃ NCS ⋃ {x'}
18.     Terminated_Color_Set := Terminated_Color_Set ⋃ {x'}
19.     if Universal_Color_Set = Terminated_Color_Set then
20.         declare global termination
21.     send TERMINATE(x', NCS) to all (outgoing) ports in
        Children ⋃ {parent} ⋃ Neighboring_Region_Ports \ {j}
```

any postrecording message in the local state or channel state. Thus, all process states and all channel states are recorded consistently, irrespective of whether the channels are intra-domain channels or inter-domain channels. □

As the global snapshot recorded by the algorithm is consistent, it satisfies all the properties satisfied by a snapshot recorded by the Chandy–Lamport algorithm.

5.2. Termination

Lemma 1. *Local snapshot recording has terminated when Children* $\bigcup$ *Unrelated* = *Neighbors* \ {*parent*}.

Proof. Each neighbor is classified as a child node or a unrelated node when an ACCEPT or a REJECT message, respectively, is received from that node. The ACCEPT or REJECT is sent on FIFO channels after a MARKER has been sent on the channel. Thus, a MARKER has been received on each channel already, and the local snapshot recording has terminated when the given condition *Children* $\bigcup$ *Unrelated* = *Neighbors* \ {*parent*} holds. □

Observation 1. *For each snapshot initiation identified by a nonce, the MARKER messages induce a spanning tree, where the parent of a node in the spanning tree is identified by the first port along which a MARKER message is received at the node.*

Lemma 2. *Snapshot recordings at nodes in the subtree rooted at a node have completed when an INSWEEP message is received from all the nodes in Children.*

Proof. A node invokes LOCAL_SNAP_COMPLETE after the local snapshot recording has completed. In LOCAL_SNAP_COMPLETE, leaf nodes initiate sending INSWEEP messages to their parent nodes in the spanning tree induced by the MARKERs. Thus, when a node receives an INSWEEP message from all its child nodes, the local snapshot recordings have completed at the child nodes. In turn, this node now sends an INSWEEP to its parent. The lemma now follows by using an inductive argument. □

Corollary 1. *When an initiator sends a TERMINATE message on its outgoing ports, snapshot recordings at all the nodes in the tree colored by the initiator's nonce have completed.*

Proof. A direct consequence of Lemma 2. □

Lemma 3. *At any node, Terminated_Color_Set* $\subseteq$ *Universal_Color_Set.*

Proof. This follows from the processing that occurs on receiving a TERMINATE message. *Universal_Color_Set* contains colors of regions that are neighboring regions of regions whose snapshot recordings have terminated, in addition to the colors of regions that have terminated. □

Theorem 2. *At any node, Terminated_Color_Set* = *Universal_Color_Set implies that global termination of the snapshot algorithm has occurred.*

Proof. At any node, *Universal_Color_Set* contains colors of all terminated regions and the colors of their neighboring regions. Thus, the set contains colors of *all* regions in which snapshot recording is known to be terminated, and the colors of their neighboring regions (in which snapshot recording is not known to be terminated). If *Universal_Color_Set* equals *Terminated_Color_Set*, then global termination in all regions can be inferred. The condition implies that there are no additional neighboring regions (in which snapshot recording is not known to be terminated) of regions whose snapshot recording is known to have terminated. If there were any neighboring regions where snapshot recording is not known to have terminated, their colors would be reported in the *NCS* parameter on TERMINATE and added to *Universal_Color_Set* but not to *Terminated_Color_Set*.

The equality implies that the fixed point of "the neighboring regions of the terminated regions have also terminated" has been reached. That is, the set of terminated regions equals the set of terminated regions union the set of neighbors of the terminated regions (where snapshot recording is not known to be terminated). □

6. Complexity

Let c be the number of concurrent initiators, n the number of nodes, and e the number of channels (links) in the system. Let n_k denote the number of nodes in region k and let $Neighboring\_Region\_Ports_k$ denote the set *Neighboring_Region_Ports* at node k.

6.1. Message complexity

Phase I: There are e MARKER messages, and there are also e ACCEPT or REJECT messages.

Phase II: There are $\leq n$ INSWEEP messages and $c(\sum_{k=1}^{c}(n_k - 1) + \sum_{k=1}^{n} |Neighboring\_Region\_Ports_k|)$ TERMINATE messages. Per color, there are $\sum_{k=1}^{c}(n_k - 1)$ intra-region TERMINATE messages and $\sum_{k=1}^{n} |Neighboring\_Region\_Ports_k|$ inter-region TERMINATE messages. As the channels are bidirectional, the total number of TERMINATE messages $\leq 2ce$.

Total number of messages: The total number of messages is $2e + n + c(\sum_{k=1}^{c}(n_k - 1) + \sum_{k=1}^{n} |Neighboring\_Region\_Ports_k|)$, which is less than or equal to $2e + n + cn^2$. Asymptotically, this is $\max(O(e), O(c(\sum_{k=1}^{c}(n_k - 1) + \sum_{k=1}^{n} |Neighboring\_Region\_Ports_k|)))$, or simply $O(cn^2)$.

6.2. Message space complexity

MARKER, ACCEPT and REJECT messages are of size $O(1)$, whereas INSWEEP and TERMINATE messages are of size $O(c)$. Therefore, the total message space complexity is $2e + cn + c^2(\sum_{k=1}^{c}(n_k - 1) + \sum_{k=1}^{n} |Neighboring\_Region\_Ports_k|)$. Asymptotically, this is $O(c^2 e)$ or $O(c^2 n^2)$.

6.3. Comments

When $c = 1$, the algorithm complexity defaults to that of the Chandy–Lamport algorithm (which neither can handle concurrent initiators nor detect global termination of the recording activities).

The message complexity of our algorithm is much lower than the $O(n^3\beta)$ complexity of the Chalopin et al. algorithm [6], and the message space complexity of our algorithm is lower than or comparable to that of the Chalopin et al. algorithm. In addition, we do not require processes to have any knowledge of the network diameter unlike the Chalopin et al. algorithm.

7. Concluding remarks

We presented a global snapshot algorithm with concurrent initiators, with termination detection of the snapshot recording in an anonymous asynchronous distributed message-passing system. Each instance of algorithm invocation by an initiator process is identified by a large random number (nonce); however, this nonce is not used as an address in any form of communication. Further, the algorithm does not assume knowledge of the network diameter or the number of processes in the system, and does not assume any predefined static topology overlay.

The algorithm handled concurrent snapshot initiations efficiently by suppressing redundant snapshot collections. The algorithm also performed termination detection of the snapshot recording activities without knowing the number of concurrent initiators, by using nonces in an ingenious way. The algorithm defines three sets at each process for tracking nonces in the local view, and applied the theory of fixed points of sets as the nonces were diffused in the system and updated local views of the sets of nonces and of the set of terminated processes. The fixed point theory of sets applied to the sets of nonces in the local view was used to detect the global termination of the snapshot recording activities.

We showed the correctness of the algorithm and derived its message complexity and message space complexity. When compared with the Chalopin et al. algorithm [6], our algorithm has much lower message complexity and a lower or comparable message space complexity. Furthermore, our algorithm does not assume knowledge of the network diameter, which is required by the Chalopin et al. algorithm. Our algorithm's complexity also compares favorably with other existing snapshot recording/collection algorithms which use process identifiers in their operations (hence are not anonymous) and do not perform termination detection.

Acknowledgments

The authors would like to thank the three anonymous referees for providing valuable comments on an earlier version of this article.

References

[1] A. Acharya, B.R. Badrinath, Recording distributed snapshots based on causal order of message delivery, Inform. Process. Lett. 44 (1992) 317–321.
[2] S. Alagar, S. Venkatesan, An optimal algorithm for distributed snapshots with causal message ordering, Inform. Process. Lett. 50 (1994) 311–316.
[3] D. Angluin, Local and global properties in networks of processors, in : Proceedings of the 12th ACM Symposium on Theory of Computing, 1980, pp. 82–93.
[4] R. Baldoni, G. Cioffi, J.-M. Helary, M. Raynal, Direct dependency-based determination of consistent global checkpoints, Comput. Syst. Sci. Eng. 16 (1) (2001) 43–49.
[5] J. Chalopin, E. Godard, Y. Metivier, Local terminations and distributed computability in anonymous networks, in: Proceedings of DISC, in: LNCS, 2008, pp. 47–62.
[6] J. Chalopin, Y. Metivier, T. Morsellino, On Snapshots and stable properties detection in anonymous fully distributed systems (Extended abstract), in: Proceedings of SIROCCO, in: LNCS, vol. 7355, Springer, 2012, pp. 207–218.
[7] K.M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems, ACM Trans. Comput. Syst. 3 (1) (1985) 63–75.
[8] R.G. Gallager, P.A. Humblet, P.M. Spira, A distributed algorithm for minimum-weight spanning trees, ACM Trans. Program. Lang. Syst. (TOPLAS) 5 (1) (1983) 66–77.
[9] R. Guerraoui, E. Ruppert, What can be implemented anonymously? in: Proceedings of DISC 2005, Springer, 2005, pp. 244–259.
[10] J.-M. Helary, Observing global states of asynchronous distributed applications, in: Proceedings of the 3rd International Workshop on Distributed Algorithms, in: LNCS, vol. 392, Springer Verlag, 1989, pp. 124–134.
[11] A.D. Kshemkalyani, M. Raynal, M. Singhal, An introduction to snapshot algorithms in distributed computing, Distrib. Syst. Eng. J. 2 (4) (1995) 224–233.
[12] A.D. Kshemkalyani, M. Singhal, Distributed Computing: Principles, Algorithms, and Systems, Cambridge University Press, 2008.
[13] A.D. Kshemkalyani, Fast and message-efficient global snapshot algorithms for large-scale distributed systems, IEEE Trans. Parallel Distrib. Syst. 21 (9) (2010) 1281–1289.
[14] T.H. Lai, T.H. Yang, On distributed snapshots, Inform. Process. Lett. 25 (1987) 153–158.
[15] D. Manivannan, R.H.B Netzer, M. Singhal, Finding consistent global checkpoints in a distributed computation, IEEE Trans. Parallel Distrib. Syst. 8 (6) (1997) 623–627.
[16] F.H. Mathis, A generalized birthday problem, SIAM Rev. 33 (2) (1991) 265–270.
[17] F. Mattern, Algorithms for distributed termination detection, Distrib. Comput. 2 (3) (1987) 161–175.
[18] F. Mattern, Efficient algorithms for distributed snapshots and global virtual time approximation, J. Parallel Distrib. Comput. 18 (1993) 423–434.
[19] M. Spezialetti, P. Kearns, Efficient distributed snapshots, in: IEEE ICDCS 1986, pp. 61–68.
[20] B. Szymanski, Y. Shy, N. Prywes, Synchronized distributed termination, IEEE Trans. Software Eng. 11 (10) (1985) 1136–1140.

Ajay D. Kshemkalyani received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Bombay in 1987, and the MS and Ph.D. degrees in computer and information science from The Ohio State University in 1988 and 1991, respectively. He spent six years at IBM Research Triangle Park working on various aspects of computer networks, before joining academia. He is currently a professor in the Department of Computer Science at the University of Illinois at Chicago. His research interests are in distributed computing, distributed algorithms, computer networks, and concurrent systems. In 1999, he received the US National Science Foundation Career Award. He previously served on the editorial board of the Elsevier journal Computer Networks, and is currently an editor of the IEEE Transactions on Parallel and Distributed Systems. He has coauthored a book entitled Distributed Computing: Principles, Algorithms, and Systems (Cambridge University Press, 2008). He is a distinguished scientist of the ACM and a senior member of the IEEE.

Mukesh Singhal is a Chancellor's Professor in the School of Engineering at the University of California at Merced. From 2001 to 2012, he was a Full Professor and Gartener Group Endowed Chair in Network Engineering in the Department of Computer Science at The University of Kentucky, Lexington. From 1986 to 2001, he was a faculty in Computer and Information Science at The Ohio State University.

He received a Bachelor of Engineering degree in Electronics and Communication Engineering with high distinction from Indian Institute of Technology, Roorkee, India, in 1980 and a Ph.D. degree in Computer Science from University of Maryland, College Park, in May 1986. His current research interests include distributed systems, cloud computing, wireless and mobile computing systems, computer security, and performance evaluation. He has published over 220 refereed articles in these areas. He has coauthored books titled "Advanced Concepts in Operating Systems", McGraw-Hill, New York, 1994, "Distributed Computing Systems", Cambridge University Press 2008, "Data and Computer Communications: Networking and Internetworking", CRC Press, 2001, and "Readings in Distributed Computing Systems", IEEE Computer Society Press, 1993. He is a Fellow of IEEE. He was a recipient of 2003 IEEE Technical Achievement Award.

He has served in the editorial board of "IEEE Trans. on Parallel and Distributed Systems", "IEEE Trans. on Data and Knowledge Engineering", and "IEEE Trans. on Computers". From 1998 to 2001, he served as the Program Director of Operating Systems and Compilers program at the National Science Foundation.