Information Processing Letters 25 (1987) 153-158
North-Holland

29 May 1987

ON DISTRIBUTED SNAPSHOTS

Ten H. LAI and Tao H. YANG

Department of Computer and Information Sciences, The Ohio State University, 2036 Neil Avenue Mall, Columbus, OH 43210, U.S.A.

Communicated by David Gries
Received 15 July 1986
Revised 5 November 1986

We develop an efficient snapshot algorithm that needs no control messages and does not require channels to be first-in-first-out. We also show that several stable properties (e.g., termination, deadlock) can be detected with uncoordinated distributed snapshots. For such properties, our algorithm can be further simplified.

Keywords: Distributed system, stable property, snapshot algorithm, deadlock detection, termination detection

1. Introduction

Chandy and Lamport [1] proposed an elegant technique, called *distributed snapshots*, for detecting stability in a distributed system:

(1) Every process takes a local snapshot by recording its own state as well as the states of all channels incident upon it.

(2) The local snapshots are collected and assembled to form a global snapshot of the system, from which it can be decided whether the system has reached a stable state.

To ensure that the scheme works correctly, Chandy and Lamport proposed that processes be somewhat coordinated in taking local snapshots so that the resulting global snapshot is 'meaningful'. They described a distributed algorithm for taking a meaningful global snapshot. The algorithm relies on channels being first-in-first-out (i.e., messages being delivered in the order sent), and it requires $O(|C|)$ control messages, where C is the set of channels in the system.

This paper presents a message-efficient algorithm for processes to take local snapshots. The algorithm does not require channels to be first-in-first-out, and it requires no control messages at all.

As mentioned, Chandy and Lamport suggested 'meaningful' global snapshots for detecting stability in a system. A natural question then arises: Can an 'uncoordinated' gobal snapshot, in which processes take local snapshots without any coordination among them, be useful for stability detection? We answer it in the affirmative by showing that several stable properties (e.g., termination, deadlock) can be detected with uncoordinated snapshots.

This paper is concerned only with the problem of taking local snapshots. The issue of forming a global snapshot is discussed in [9]. Interesting derivatives of the Chandy-Lamport algorithm can be found in [4,8].

2. Model of a distributed system

We adopt the model of [1] with two modifications: (i) interprocess communications are not necessarily first-in-first-out, and (ii) a computation of a system is defined as a function of time.

(Throughout this paper, whenever time is used, it denotes global time, and this is for purpose of presentation only—the distributed system itself does not have global time.) Below we give a brief description of the model; the reader is referred to [1] for more information.

A *distributed system* is a strongly-connected directed graph, where nodes represent processes and arcs represent unidirectional channels. Processes communicate with one another exclusively by sending and receiving messages. Messages sent along a channel are assumed to be delivered correctly, with arbitrary but finite delay, but *not* necessarily in the order being sent.

Each channel is associated with a set of messages, called its *state*, that may grow and shrink. When a message is sent (received) along a channel, the message is added to (removed from) the set associated with the channel. A *process* consists of a set of states, an initial state in the set, and a set of events. An *event* of a process p is a 5-tuple $\langle p, s, s', \mu, c\rangle$, meaning that process p changes its state from s to s′ and sends (receives) message μ along outgoing (incoming) channel c which is incident upon p; μ and c are *null* symbols if no message is involved in the event. A *global state* of a distributed system is a set of process and channel states—one state per process/channel. The *initial* global state is one in which each process state is an initial state and each channel is empty. An event $\langle p, s, s', \mu, c\rangle$ can occur in global state S iff (a) the state of p in S is s, and (b) if c is an incoming channel, then μ is contained in the state of c in global state S. If event e can occur in global state S, then *next*(S, e) denotes the global state immediately after the occurrence of e in global state S.

Let E be the union of all event sets in the system. Let f be a function mapping the time-line $(0, \infty)$ to $E \cup \{\Lambda\}$ such that $f(t) = \Lambda$ for all but a finite number of t, where Λ is a special symbol not in E denoting 'no event'. Let t_i, $1 \leqslant i \leqslant n$, be all t such that $f(t) \neq \Lambda$. Assume $0 = t_0 < t_1 < \cdots < t_n < t_{n+1} = \infty$. We say that f is a *computation of the system* iff event $f(t_i)$ can occur in gobal state S_{i-1}, where S_0 is the initial global state and $S_i = \mathit{next}(S_{i-1}, f(t_i))$, $1 \leqslant i \leqslant n$. We occasionally write a computation as $\{S_i : 0 \leqslant i \leqslant n\}$.

Global state S′ is *reachable* from global state S (denoted as $S \to S'$) iff there is a computation $\{S_i : 0 \leqslant i \leqslant n\}$ such that $S = S_j$ and $S' = S_k$ for some j, k, $0 \leqslant j \leqslant k \leqslant n$.

Let y be a predicate function on the set of all global states of a distributed system. The predicate is said to be a *stable property* of the system iff y(S) implies y(S′) whenever $S_0 \to S \to S'$.

3. Snapshots

Consider a fixed computation f of a distributed system with process set P and channel set C. Let $t_0, \ldots, t_{n+1}$ and $S_0, \ldots, S_n$ be as before. All definitions below are made relative to computation f.

For $t \in (t_i, t_{i+1})$, $0 \leqslant i \leqslant n$, define

$GS(t) = S_i$ (i.e., S_i is the global state of the system at time t),

state(p, t) = the state of p in global state S_i,

sent(c, t) = the set of all messages sent along channel c by time t,

received(c, t) = the set of all messages received along channel c by time t.

A *local snapshot* of process p taken at time t consists of:

- *state*(p, t),
- *sent*(c, t), for every outgoing channel c incident upon p,
- *received*(c, t), for every incoming channel c incident upon p.

A *distributed snapshot* of a system taken at times $\{t_p : p \in P\}$ is a set of local snapshots with p's snapshot taken at time t_p; that is, the following set:

$$\{\mathit{state}(p, t_p) : p \in P\}$$
$$\cup \{\mathit{sent}(c, t_p), \mathit{received}(c, t_q) : c = (p, q) \in C\}.$$

A *global snapshot* of a system taken at times $\{t_p : p \in P\}$ is the following set:

$$\{\mathit{state}(p, t_p) : p \in P\}$$
$$\cup \{\mathit{sent}(c, t_p) - \mathit{received}(c, t_q) : c = (p, q) \in C\}.$$

Note that a global snapshot is an 'assembled' distributed snapshot, and is always a global state.

A global snapshot GSN (respectively, distributed snapshot DSN) is occasionally written as $GSN(t_p : p \in P)$ (respectively, $DSN(t_p : p \in P)$) to emphasize the timing of local snapshots.

A global snapshot $GSN(t_p : p \in P)$ taken between times τ_1 and τ_2 (i.e., $\tau_1 < t_p < \tau_2$ $\forall p \in P$) is *meaningful* iff GSN is reachable from global state $GS(\tau_1)$ and global state $GS(\tau_2)$ is reachable from GSN. It is *feasible* iff $received(c, t_q) \subseteq sent(c, t_p)$ for every channel $c = (p, q) \in C$.

The following theorem was established in [1]. We give a simpler proof here.

3.1. Theorem ([1]). *All feasible global snapshots are meaningful.*

Proof. Let $GSN(t_p : p \in P)$ be a feasible global snapshot taken between times τ_1 and τ_2, during computation f. Let $\delta = \tau_2 - \tau_1$. Construct computation f′ from f as follows: f′ is the same as f except that every post-snapshotting event in f is now postponed for δ units of time. That is,

$$f'(t) = \begin{cases} f(t) & \text{if } f(t) \text{ is an event at } p \in P \\ & \quad \text{and } t \leqslant t_p, \\ f(t-\delta) & \text{if } f(t-\delta) \text{ is an event at } p \in P \\ & \quad \text{and } t_p < t - \delta, \\ \Lambda & \text{otherwise.} \end{cases}$$

Fig. 1 displays an example of f′. Since GSN is feasible, f′ is easily seen to be a computation of the system. One may readily check that

$$GS(\tau_1, f) = GS(\tau_1, f'),$$

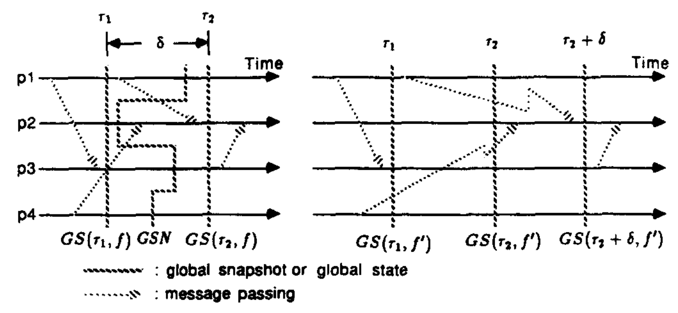

Fig. 1. *Left*: computation f. *Right*: computation f′.

$$GSN = GS(\tau_2, f'),$$

$$GS(\tau_2, f) = GS(\tau_2 + \delta, f'),$$

where $GS(\tau, g)$ denotes the global state of the system at time τ during computation g. So, global snapshot GSN is meaningful. □

4. An efficient snapshot algorithm

The snapshot algorithm of [1] not only takes a distributed snapshot, but further *distributively* refines it to a global snapshot. It works basically as below.

Step 1. Every process takes a local snapshot and immediately sends a *marker* along every outgoing channel; this must be done *before* or *upon* receiving the first arriving marker; let t_p denote the time this is done at node p.

Step 2. Upon receiving a marker along channel $c = (p, q) \in C$, process q computes $M(c) = sent(c, t_p) - received(c, t_q)$.

Step 3. The set $\{state(p, t_p) : p \in P\} \cup \{M(c) : c \in C\}$ is a meaningful global snapshot.

Note that a marker sent along channel $c = (p, q)$ serves two purposes: (i) to ensure that t_p and t_q are such that $received(c, t_q) \subseteq sent(c, t_p)$, and (ii) for process p to implicitly inform process q of the value of $sent(c, t_p)$. (If channel c is FIFO and q receives p's marker at time t'_q, then $received(c, t'_q) = sent(c, t_p)$.)

We now derive an algorithm that uses no markers and that does not require channels to be FIFO. The idea is simple: let the distributed snapshot be transformed into a global snapshot *at one node* (i.e., nondistributively), thereby releasing a marker from the duty of (ii). Then achieve (i) without markers as follows. Consider any channel $c = (p, q)$. If after time t_p process p will never send messages to process q, then clearly $received(c, t_q) \subseteq sent(c, t_p)$ and no marker is necessary for channel c. Otherwise, let the marker 'piggyback' on every outgoing post-snapshotting message. These ideas lead to the snapshot algorithm below.

Snapshot Algorithm. The processes operate according to the following rules.

***Rule* 1.** Every process is initially white and turns red while taking a local snapshot.

***Rule* 2.** Every message sent by a white (red) process is colored white (red).

***Rule* 3.** Every white process takes a snapshot at its convenience—but no later than a red message is possible received. (Thus, the arrival of a red message at a white process will invoke the process to take a snapshot before receiving the message.)

After the local snapshots are collected to a node, they are combined to form a global snapshot, which is readily seen, by Theorem 3.1, to be meaningful.

Our algorithm needs no control messages at all in taking local snapshots and does not require channels to be FIFO. It requires more space, however, in sending local snapshots to other nodes; for they now each contain the complete message history of a node instead of, as in the algorithm of [1], just messages in transit. Fortunately, this drawback can be overcome in important applications such as termination and deadlock detection (see Section 6), where the *number* of messages in *sent*(c, t_p) (*received*(c, t_q)) rather than the set itself is of concern.

Note that our algorithm heavily relies on every process being able to spontaneously take a local snapshot even if no red messages ever arrive—this would not be a problem if the system is weakly fair in the sense that every node eventually executes the snapshot algorithm. (In contrast, in [1], only one process is required to have this 'spontaneity' nature.) If only one process can spontaneously initiate the algorithm, a *signal* may be passed along a spanning tree to ask every process to take a snapshot [6,10]; in that case, our algorithm requires $O(|P|)$ control messages.

In applications, a system usually has to repeatedly take global snapshots until the stable property in concern occurs and is detected. It is a straightforward exercise to modify the above algorithm so that it can take a series of distributed snapshots. Besides, one may reduce the size of a local snapshot by resetting *sent*( ) and *received*( ) to $\emptyset$ after a local snapshot is taken, assuming that the latest global snapshot is still available at the node responsible for forming global snapshots.

5. Strongly stable properties

We now consider the question raised in Section 1: Is there any stable property that can be detected with uncoordinated distributed snapshots? And, more interestingly: It there any stable property that can be detected by a nonmeaningful global snapshot?

A property y of system D = (P, C) is *strongly stable* iff $y(GS(\tau_1))$ implies y(GSN) and y(GSN) implies $y(GS(\tau_2))$ for all global snapshots GSN taken between times τ_1 and τ_2 during all computations of D.

A strongly stable property is obviously a stable property. The converse is in general not true. For instance, the property "Dijkstra's self-stabilization system has reached a stable state" [3,5] is stable, but not strongly stable.

If a stable property in concern is known to be strongly stable, then it can be detected with uncoordinated snapshots; as a result, our snapshot algorithm becomes tremendously simple:

Every process takes a snapshot at its convenience.

Note that a strongly stable property y may or may not be detected by a nonmeaningful global state. The latter can happen if, for example, y(GSN) = **false** for all nonmeaningful GSNs.

In what follows we show that the property 'local-deadlock' as defined in [2] is a strongly stable property that can be detected even with a nonmeaningful global snapshot.

Consider a distributed system D = (P, C) in which every process is either *active* or *idle*, and every process is associated with a set of processes called its *dependent set*. The active/idle status of a process together with its dependent set constitutes the *state* of the process. An idle process becomes active upon receiving a message from any process in its dependent set; otherwise, it stays idle without changing its dependent set. An active process is free to send or receive messages, and may become idle at any moment.

The system is said to be *locally deadlocked* in global state S if there is a nonempty set $Q \subseteq P$ such that the following are true in S:

(i) All processes in Q are idle.

(ii) The dependent set of every process in Q is a subset of Q.

(iii) Every channel between processes in Q is empty.

In that case, Q is said to be *deadlocked.*

5.1. Lemma. *During a computation, if* $Q \subseteq P$ *is deadlocked in global snapshot* $GSN(t_p : p \in P)$, *then no process* $q \in Q$ *may resume active after time* t_q.

Proof. Otherwise, let $R = \{r \in Q : r$ ever resumes active after time $t_r\}$, and let q be the process in R that resume active earliest. The message that makes q active was sent by some process $p \in Q$ *before* time t_p, or else the 'earliest' nature of q would be contradicted. But then the channel from p to q is nonempty in GSN, contradicting the 'deadlock' nature of Q. □

5.2. Theorem. *That a distributed system is locally deadlocked is a strongly stable property and may be detected even by a nonmeaningful global snapshot.*

Proof. Let $GSN(t_p : p \in P)$ be any global snapshot taken between τ_1 and τ_2. Assume the system to be locally deadlocked at time τ_1. By definition, there is a set of processes Q that is deadlocked at time τ_1. It is clear that no processes in Q, and no channels between processes of Q, may change their states after time τ_1. Hence, the system is locally deadlocked in GSN.

Conversely, assume the system to be locally deadlocked in GSN. Then there exists a deadlocked subset Q (relative to GSN). For contradiction, assume the system is not locally deadlocked at time τ_2. In particular, Q is not deadlocked at τ_2. By definition, at least one of the following holds at time τ_2: (i) some process in Q is still active, (ii) the dependent set of some process in Q is not contained in Q, and (iii) some messages are still in transit between processes of Q. In all cases, there must be a process in Q that resumes active after taking its local snapshot, in contradiction to Lemma 5.1. So, the system must be locally deadlocked at time τ_2. By definition, the property in concern is strongly stable.

We now show that a local deadlock may be detected by a nonmeaningful global snapshot. Fig.

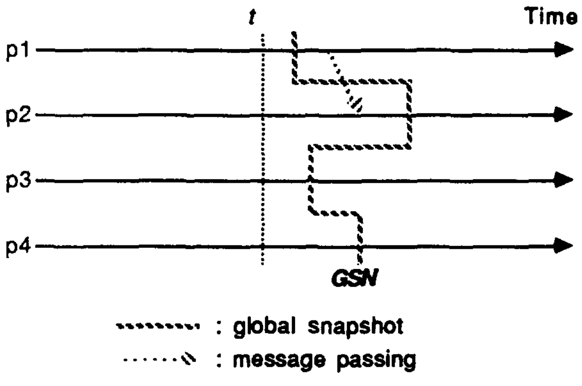

Fig. 2.

2 displays a system of four processes and a nonfeasible global snapshot GSN. It is not hard to actually define the system so that GSN is also nonmeaningful. Let $\{p_3, p_4\}$ be deadlocked at time t. This fact evidently can be detected by GSN. So, the theorem is proved. □

Besides local-deadlock, "the entire system is deadlocked" and "the system is terminated" [6] are also strongly stable properties.

6. Remarks on deadlock and termination detection

This section justifies our claim that several applications (e.g., deadlock and termination detection) need only message counters (instead of complete message histories) for each channel.

6.1. Theorem. *Let* $GSN(t_p : p \in P)$ *be any global snapshot. A nonempty set* $Q \subseteq P$ *is deadlocked in* GSN *iff the following are true in* GSN: (i) *every process in* Q *is idle*, (ii) *the dependent set of every process in* Q *is a subset of* Q, *and* (iii) $|sent(c, t_p)| = |received(c, t_q)|$ *for every channel* $c = (p, q) \in Q \times Q$.

Proof. Consider any distributed snapshot $GSN(t_p : p \in P)$ and nonempty set $Q \subseteq P$.

("$\Rightarrow$"): Assume Q deadlocked in GSN. By definition, (i) and (ii) hold and, for every channel $c = (p, q) \in Q \times Q$, $sent(c, t_p) \subseteq received(c, t_q)$. If $sent(c, t_p) = received(c, t_q)$ for every $c = (p, q) \in Q \times Q$, we are done. So, assume $sent(c, t_p) \neq$

received(c, t_q) for some $c = (p, q) \in Q \times Q$. That is, at least one message in *received*(c, t_q) is sent by p after time t_p, contradicting Lemma 5.1.

("$\Leftarrow$"): Now assume that (i), (ii), and (iii) hold in GSN. For contradiction, assume Q is not deadlocked in GSN. That is, assume $\textit{sent}(c, t_p) - \textit{received}(c, t_q) \neq \emptyset$ for some channel $c = (p, q)$. Since $|\textit{sent}(c, t_p)| = |\textit{received}(c, t_q)|$, it follows that $\textit{received}(c, t_q) - \textit{sent}(c, t_p) \neq \emptyset$ and hence p sends at least one message after t_p. So, the set $R = \{r \in Q : r \text{ ever resumes active after time } t_r\}$ is not empty. Let q′ be the process in R that resumes active earliest, and p′ be the sender of the message, say μ, that makes q′ active. If message μ is sent after time $t_{p'}$, then p′ resumes active earlier than q′. If μ is sent before time t_p, then since $|\textit{sent}(c', t_{p'})| = |\textit{received}(c', t_{q'})|$, where $c' = (p', q')$, q′ must receive before $t_{q'}$ a message that is sent by p′ after time $t_{p'}$. In both cases, the 'earliest' nature of q′ is contradicted. □

Thus, in local-deadlock detection, instead of recording the 'set' of messages sent (received) along a channel, it suffices to count the 'number' of messages sent (received).

For termination detection, it is even possible to use only one message counter without distinguishing between channels [7].

Acknowledgment

The authors express their gratitude to an anonymous referee for helpful comments and suggestions.

References

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

[2] K.M. Chandy, J. Misra and L.M. Haas, Distributed deadlock detection, ACM Trans. Comput. Syst. 1 (2) (1983) 144–156.

[3] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, Comm. ACM 17 (1974) 643–644.

[4] E.W. Dijkstra, The distributed snapshot of K.M. Chandy and L. Lamport, Tech. Rept. EWD 864a, Univ. of Texas, Austin, TX, 1983.

[5] E.W. Dijkstra, A belated proof of self-stabilization, Distributed Comput. 1 (1986) 5–6.

[6] N. Francez and M. Rodeh, Achieving distributed termination without freezing, IEEE Trans. Software Engrg. SE-8 (1982) 287–292.

[7] T.H. Lai, Termination detection for dynamically distributed systems with non-first-in-first-out communication, J. Parallel & Distributed Computing 3 (1986) 577–599.

[8] C. Morgan, Global and logical time in distributed algorithms, Inform. Process. Lett. 20 (1985) 189–194.

[9] M. Spezialetti and P. Kearns, Efficient distributed snapshots, Proc. 6th Internat. Conf. on Distributed Computing Systems (1986) 382–388.

[10] R. Topor, Termination detection for distributed computations, Inform. Process. Lett. 18 (1984) 33–36.