Distributed Snapshots: Determining Global States of Distributed Systems

Prof. Phillip Gibbons

Spring 2020, Lecture 5
Today’s Reminders / Announcements

• Spent many hours working on the semester schedule...
  – Have all the papers selected
  – Good mix of HoF papers & recent best papers
  – Still need to post to the syllabus page

• As a result, did not get to the Summaries yet...
“Distributed Snapshots: Determining Global States of Distributed Systems”
K. Mani Chandy & Leslie Lamport 1985

- Leslie Lamport (SRI, DEC SRC, MSR)
  - National Academy of Science

- Mani Chandy (UT Austin, Caltech)
  - National Academy of Engineering

- Most cited papers: todays (3570), book article (3264)
- Who else has an article in same book (but w/8 cites)?
“Distributed Snapshots: Determining Global States of Distributed Systems”
K. Mani Chandy & Leslie Lamport 1985

SigOps HoF citation (2013):
This paper takes the idea of consistency for distributed predicate evaluation, formalizes it, distinguishes between stable and dynamic predicates, and shows precise conditions for correct detection of stable conditions. The fundamental techniques in the paper are the secret sauce in many distributed algorithms for deadlock detection, termination detection, consistent checkpointing for fault tolerance, global predicate detection for debugging and monitoring, and distributed simulation.
Leslie Lamport on Today’s Paper

“The distributed snapshot algorithm described in this paper came about when I visited Chandy, who was then at the University of Texas at Austin.

He posed the problem to me over dinner, but we had both had too much wine to think about it right then.

The next morning, in the shower, I came up with the solution.

When I arrived at Chandy's office, he was waiting for me with the same solution.

I consider the algorithm to be a straightforward application of the basic ideas from [27].”

[27] is “Time, Clocks, and the Ordering of Events in a Distributed System”
Global State Detection
System Model

- Finite labeled, directed graph in which vertices represent processes & edges represent channels

- Channels have infinite buffers, in-order delivery, arbitrary but finite delays, are uni-directional & error-free
Events

An event is defined by:

– process p
– state s of p immediately before the event
– state s’ of p immediately after the event
– channel c (if any) whose state is altered by the event
– message M (if any) sent/received along c
Example

Fig. 4. Global states and transitions of the single-token conservation system.
Inconsistent Global State

- state of p: in-p (p has token)
- state of q: in-c
- state of c: has token
- state of c’: empty

state transitions to in-c

- state of p: in-c
- state of q: in-c
- state of c: empty
- state of c’: empty

Problem: global state shows
- 2 tokens in system
- 0 tokens in system
Global-State-Detection Algorithm

• **Marker-Sending Rule for p**
  For each channel c outgoing from p:
  – p records state, then sends a marker as its next message on c

• **Marker-Receiving Rule for q**
  On receiving a marker along a channel c:
  – If q has not recorded its state then
    q records its state; q records the state c as empty
  – Else q records state of c as the sequence of messages received along c after q’s state was recorded yet before q received the marker along c

Termination: As long as at least 1 process spontaneously records its state & no marker remains stuck in a channel & the graph is strongly connected, then all processes record their states in finite time
Example

1. In S0, p records state=A, puts marker on c
2. p puts M on c (S1)
   q puts M’ on c’ (S2)
   p receives M’ (S3)
3. Marker received by q;
   q records state=D, c=empty,
   q puts marker on c’
4. Marker received by p;
   p records c’=<M’>

Never Happened!
So...In What Way is the Recorded Global State “Meaningful”? 

• It *could* have occurred

• There is a computation where
  – Sequence of states before the DS algorithm starts is unchanged
  – Sequence of states after the DS algorithm ends is unchanged
  – Sequence of events in between may (only) be reordered
  – Recorded global state is one of the states in between

• But why is that useful???
Applying to Prior Example

Never Happened!

But Could’a Happened

q sends M’

p sends M
Theorem

- There is a computation seq’ derived from seq where
  - Sequence of states before/after DS starts/ends is unchanged
  - Sequence of events in between may (only) be reordered
  - Recorded global state S* is one of the states in between

- Prerecording event: occurs at p before p records its state
  Postrecording event: ...after...

- seq’ is seq permuted such that all prerecording events occur before any postrecording events
1. In $S_0$, $p$ records state=$A$, puts marker on $c$

2. $p$ puts $M$ on $c$ ($S_1$)
   $q$ puts $M'$ on $c'$ ($S_2$)
   $p$ receives $M'$ ($S_3$)

3. Marker received by $q$;
   $q$ records state=$D$, $c$=empty,
   $q$ puts marker on $c'$

4. Marker received by $p$;
   $p$ records $c'=<M'>$
Example: Swapping Post and Pre

- p sends M
- q sends M'
- p sends M

States:
- A
- B
- C
- D

Global States:
- S0
- S1
- S2
- S3

Transitions:
- postrecording
- prerecording

Initial global state S0

Red box indicates S*
Discussion: Summary Question #1

• **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.
Collecting the Global State

- Each $p$ repeatedly floods along all outgoing channel what it knows about the global state
Stability Detection

• Input: Any stable property \( y \)
  Stable: \( y(S) \) implies \( y(S') \) for all global states \( S' \) reachable from \( S \)

• Return:
  – FALSE implies property \( y \) did not hold when DS algorithm starts
  – TRUE implies property \( y \) holds when DS algorithm ends

Note: If \( y \) starts holding after DS start, ok to return FALSE

• SD Algorithm:
  – Record a global state \( S^* \); Return \( y(S^*) \)

• Correctness:
  – \( y(S^*)=\text{TRUE} \) implies \( y(\text{DS end state})=\text{TRUE} \) [reachable, \( y \) stable]
  – \( y(\text{DS start state})=\text{TRUE} \) implies \( y(S^*)=\text{TRUE} \) [reachable, \( y \) stable]
• **Describe the paper's single most glaring deficiency.** Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.
Theorem & Proof Sketch

• There is a computation seq’ derived from seq where
  – Sequence of states before/after DS starts/ends is unchanged
  – Sequence of events in between may (only) be reordered
  – Recorded global state S* is one of the states in between

• Prerecording event: occurs at p before p records its state
  Postrecording event: ...after...

• seq’ is seq permuted such that all prerecording events
  occur before any postrecording events

• Must show:
  – seq’ is a legal computation
  – S* is the global state in seq’ at the transition point
Swapping Post and Pre

• Why legal to swap $e_{j-1}$ (post) and $e_j$ (pre) ?
  – On different processes, say $p$ and $q$
  – No message $M'$ sent at $e_{j-1}$ received at $e_j$
    Why? Since $e_{j-1}$ is post, marker already sent
      If $M'$ received then $q$ already received marker
      & recorded state, so $e_j$ would be post
  – State of $q$ not altered by occurrence of $e_{j-1}$ since at $p$
  – If $e_j$ is a receive $M$ along $c$ event, then $M$ already at head of $c$
    before $e_{j-1}$
  – Thus, $e_j$ can occur in global state $S_{j-1}$

\[ S_{j-1} \]

\[ e_j: q \text{ sends } M' \]
Swapping Post and Pre

• Why legal to swap $e_{j-1}$ (post) and $e_j$ (pre) ?
  – State of p not altered by occurrence of $e_j$
  – Thus, $e_{j-1}$ can occur immediately after $e_j$
  – Moreover, state after $e_1, ..., e_{j-2}, e_j, e_{j-1}$ is same as $e_1, ..., e_{j-1}, e_j$

• Repeatedly swap until all pre before any post

• $S^*$ is the same as state at pre-to-post transition
  – Follows from Marker-Send and Marker-Receive rules

QED
• Describe what conclusion you draw from the paper as to how to build systems in the future. Most of the assigned papers are significant to the systems community and have had some lasting impact on the area.
Monday’s Papers: Bug Detection

“Eraser: A Dynamic Data Race Detector for Multi-Threaded Programs”
Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, Thomas Anderson 1997

“Efficient and Scalable Thread-Safety Violation Detection”
Guangpu Li, Shan Lu, Madanlal Musuvathi, Suman Nath, Rohan Padhye 2019