Distributed Computing Concepts - Global State in Distributed Systems

Prof. Nalini Venkatasubramanian

230 Distributed Systems - Week 3

-includes slides/examples from
Indy Gupta (UIUC), Coulouris(book) and
Kshemkalyani&Singhal (book slides)
Why Global State?

- Distributed applications/services execute concurrently on multiple machines.
- A **Snapshot** of the distributed application, i.e. a **global picture is useful**
  - *Checkpointing*: can restart distributed application on failure
  - *Garbage collection* of objects: objects at servers that don’t have any other objects (at any servers) with pointers to them
  - Deadlock detection: Useful in database transaction systems
  - Termination of computation: Useful in batch computing systems like Folding@Home, SETI@Home
What constitutes global state?

• **Global Snapshot** = **Global State**

  Individual state of *each process* in the distributed system
  + Individual state of *each communication channel* in the distributed system

• Capture the *instantaneous state* of *each process*
• Capture the instantaneous *state* of *each communication channel*, i.e., *messages* in transit on the channels
Simulate A Global State

- The notions of global time and global state are closely related.
- But, merely synchronizing clocks and taking local snapshots is not enough.
- Need to account for messages in transit.

- A process can (without freezing the whole computation) compute the best possible approximation of a global state [Chandy & Lamport 85].

- A global state that could have occurred:
  - No process in the system can decide whether the state did really occur.
  - Guarantee stable properties (i.e. once they become true, they remain true).
Equivalent Event Diagram
Rubber Band Transformation

P1

P2

P3

P4

e11

e12

e21

e22

e31

e41

e42

cut

Time

Consistent Cuts

- A cut (or time slice) is a zigzag line cutting a time diagram into 2 parts (past and future)
  - \( E \) is augmented with a cut event \( c_i \) for each process \( P_i : E' = E \cup \{ c_i, \ldots, c_n \} \).

- A cut \( C \) of an event set \( E \) is a finite subset \( C \subseteq E : e \in C \land e' < e \rightarrow e' \in C \).

- A cut \( C_1 \) is later than \( C_2 \) if \( C_1 \supseteq C_2 \).

- A consistent cut \( C \) of an event set \( E \) is a finite subset \( C \subseteq E : e \in C \land e' < e \rightarrow e' \in C \).
  - i.e. a cut is consistent if every message received was previously sent (but not necessarily vice versa!)
Cuts (Summary)

Instant of local observation

P1

P2

P3

initial value

ideal (vertical) cut

consistent cut

inconsistent cut

not attainable

equivalent to a vertical cut (rubber band transformation)

can’t be made vertical (message from the future)

Time
Consistent Cuts

● Some Theorems

● For a consistent cut consisting of cut events \( c_i, ..., c_n \), no pair of cut events is causally related. i.e. \( \forall c_i, c_j \sim (c_i < c_j) \wedge \sim (c_j < c_i) \)

● For any time diagram with a consistent cut consisting of cut events \( c_i, ..., c_n \), there is an equivalent time diagram where \( c_i, ..., c_n \) occur simultaneously. i.e. where the cut line forms a straight vertical line

● All cut events of a consistent cut can occur simultaneously
Global States of Consistent Cuts

- The global state of a distributed system is a collection of the local states of the processes and the channels.
- A *global state* computed along a consistent cut is **correct**.
- The *global state* of a consistent cut comprises the local state of each process at the time the cut event happens and the set of all messages sent but not yet received.
- The *snapshot problem* consists in designing an efficient protocol which yields only consistent cuts and to collect the local state information:
  - Messages crossing the cut must be captured.
  - Chandy & Lamport presented an algorithm assuming that message transmission is FIFO.
Distributed Global Snapshot: Requirements

- Snapshot should not interfere with normal application actions, and it should not require application to stop sending messages.

- Each process is able to record its own state:
  - Process state: Application-defined state or, in the worst case:
    - its heap, registers, program counter, code, etc. (essentially the coredump)

- Global state is collected in a distributed manner.

- Any process may initiate the snapshot:
  - Assume just one snapshot run for now.
System Model for Global Snapshots

● The system consists of a collection of n processes p1, p2, ..., pn that are connected by channels.
● There are no globally shared memory and physical global clock and processes communicate by passing messages through communication channels.
● $C_{ij}$ denotes the channel from process $p_i$ to process $p_j$ and its state is denoted by $SC_{ij}$.
● The actions performed by a process are modeled as three types of events:
  ● Internal events, the message send event and the message receive event.
  ● For a message $m_{ij}$ that is sent by process $p_i$ to process $p_j$, let $send(m_{ij})$ and $rec(m_{ij})$ denote its send and receive events.
Process States and Messages in transit

- At any instant, the state of process $p_i$, denoted by $L_{Si}$, is a result of the sequence of all the events executed by $p_i$ till that instant.
- For an event $e$ and a process state $L_{Si}$, $e \in L_{Si}$ iff $e$ belongs to the sequence of events that have taken process $p_i$ to state $L_{Si}$.
- For an event $e$ and a process state $L_{Si}$, $e$ (not in) $L_{Si}$ iff $e$ does not belong to the sequence of events that have taken process $p_i$ to state $L_{Si}$.
- For a channel $C_{ij}$, the following set of messages can be defined based on the local states of the processes $p_i$ and $p_j$
  
  $Transit: transit(L_{Si}, L_{Sj}) = \{m_{ij} |send(m_{ij}) \in L_{Si} \lor rec(m_{ij}) \not\in L_{Sj}\}$
Chandy-Lamport Distributed Snapshot Algorithm

- Assumes FIFO communication in channels
- Uses a control message, called a marker to separate messages in the channels.
  - After a site has recorded its snapshot, it sends a marker, along all of its outgoing channels before sending out any more messages.
  - The marker separates the messages in the channel into those to be included in the snapshot from those not to be recorded in the snapshot.
- A process must record its snapshot no later than when it receives a marker on any of its incoming channels.
- The algorithm terminates after each process has received a marker on all of its incoming channels.
- All the local snapshots get disseminated to all other processes and all the processes can determine the global state.
Chandy-Lamport Distributed Snapshot Algorithm

**Marker receiving rule for Process Pi**

If (Pi has not yet recorded its state) it
   records its process state now
   records the state of c as the empty set
   turns on recording of messages arriving over other channels

else
   Pi records the state of c as the set of messages received over c
   since it saved its state

**Marker sending rule for Process Pi**

After Pi has recorded its state, for each outgoing channel c:
   Pi sends one marker message over c
   (before it sends any other message over c)
1. P1 initiates snapshot: records its state (S1); sends Markers to P2 & P3; turns on recording for channels C21 and C31

2- P2 receives Marker over C12, records its state (S2), sets state(C12) = {} sends Marker to P1 & P3; turns on recording for channel C32

3- P1 receives Marker over C21, sets state(C21) = {a}

4- P3 receives Marker over C13, records its state (S3), sets state(C13) = {} sends Marker to P1 & P2; turns on recording for channel C23

5- P2 receives Marker over C32, sets state(C32) = {b}

6- P3 receives Marker over C23, sets state(C23) = {}

7- P1 receives Marker over C31, sets state(C31) = {}}
Snapshot Example

From: Indranil Gupta (CS425 - Distributed Systems course, UIUC)
P1 is Initiator:
• Record local state S1,
• Send out markers
• Turn on recording on channels $C_{21}, C_{31}$
S1, Record $C_{21}$, $C_{31}$

- First Marker!
- Record own state as S3
- Mark $C_{13}$ state as empty
- Turn on recording on other incoming $C_{23}$
- Send out Markers
S1, Record $C_{21}$, $C_{31}$

- S3
- $C_{13} = <>$
- Record $C_{23}$
S1, Record C_{21}, C_{31}

State of channel C_{31} = <>

Duplicate Marker!

P1: A, B, C, D, E

P2: E, F, G, H, I, J

P3: H, I, J

- S3
- C_{13} = <>
- Record C_{23}
S1, Record $C_{21}, C_{31}$

- $C_{31} = \langle \rangle$

- First Marker!
- Record own state as S2
- Mark $C_{32}$ state as empty
- Turn on recording on $C_{12}$
- Send out Markers
S1, Record $C_{21}, C_{31}$  
$C_{31} = <>$

- S3
- $C_{13} = <>$
- Record $C_{23}$

- S2
- $C_{32} = <>$
- Record $C_{12}$
S1, Record $C_{21}, C_{31}$

$C_{31} = <>$

- S3
- $C_{13} = <>$
- Record $C_{23}$

- S2
- $C_{32} = <>$
- Record $C_{22}$

- Duplicate
- $C_{12} = <>$
- Duplicate!

- $C_{21} = \langle \text{message G [D] } \rangle$

- $S1, \text{Record } C_{21}, C_{31}$

- $C_{31} = <>$

- $S2$

- $C_{32} = <>$

- $C_{12} = <>$

- Record $C_{12}$
S1, Record $G_{2t}, G_{3t}$

$C_{21} = \langle \text{message } G \rightarrow D \rangle$

$C_{31} = <>$

$C_{32} = <>$

$C_{13} = <>$

$C_{12} = <>$

Record $G_{23}$

Record $G_{32}$

Duplicate!
Algorithm has Terminated

\[ C_{21} = \text{<message } G \rightarrow D > \]

\[ C_{31} = <> \]

\[ C_{32} = <> \]

\[ C_{12} = <> \]

\[ C_{23} = <> \]
Collect the Global Snapshot Pieces

C_{21} = \langle \text{message G} \rightarrow \text{D} \rangle

C_{31} = \langle \rangle

C_{21} = \langle \rangle

C_{32} = \langle \rangle

C_{12} = \langle \rangle

C_{23} = \langle \rangle
Our Global Snapshot Example …
... is causally correct

Consistent Cut captured by our Global Snapshot Example
Chandy-Lamport Extensions: Spezialetti-Kerns and others

- Exploit concurrently initiated snapshots to reduce overhead of local snapshot exchange
- Snapshot Recording
  - Markers carry identifier of initiator – first initiator recorded in a per process “master” variable.
  - **Region** - all the processes whose master field has same initiator.
  - Identifiers of concurrent initiators recorded in “*id-border-set*.”
- Snapshot Dissemination
  - Forest of spanning trees is implicitly created in the system. Every Initiator is root of a spanning tree; nodes relay snapshots of rooted subtree to parent in spanning tree
  - Each initiator assembles snapshot for processes in its region and exchanges with initiators in adjacent regions.
- Others: multiple repeated snapshots; wave algorithm
Computing Global States without FIFO Assumption

- In a non-FIFO system, a marker cannot be used to delineate messages into those to be recorded in the global state from those not to be recorded in the global state.
- In a non-FIFO system, either some degree of inhibition or piggybacking of control information on computation messages to capture out-of-sequence messages is required.
Computing Global States without FIFO Assumption - Lai-Yang Algorithm

- Uses a \textit{coloring} scheme that works as follows
  - White (before snapshot); Red (after snapshot)
  - Every process is initially white and turns \textcolor{red}{red} while taking a snapshot. The equivalent of the “Marker Sending Rule” (virtual broadcast) is executed when a process turns \textcolor{red}{red}.
  - Every message sent by a white (\textcolor{red}{red}) process is colored white (\textcolor{red}{red}).
  - Thus, a white (\textcolor{red}{red}) message is a message that was sent before (after) the sender of that message recorded its local snapshot.
  - Every white process takes its snapshot at its convenience, but no later than the instant it receives a \textcolor{red}{red} message.
Computing Global States without FIFO Assumption - Lai-Yang Algorithm (cont.)

- Every white process records a history of all white messages sent or received by it along each channel.
- When a process turns red, it sends these histories along with its snapshot to the initiator process that collects the global snapshot.
- Determining Messages in transit (i.e. White messages received by red process)
  - The initiator process evaluates transit($LS_i$, $LS_j$) to compute the state of a channel $C_{ij}$ as given below:
    - $SC_{ij} = \{\text{white messages sent by } p_i \text{ on } C_{ij} - \text{white messages received by } p_j \text{ on } C_{ij}\}$
    - $= \{\text{send}(M_{ij})| send(m_{ij}) \in LS_i\} - \{\text{rec}(m_{ij})| rec(m_{ij}) \in LS_j\}$. 
Computing Global States without FIFO Assumption: Termination

- First method
  - Each process $I$ keeps a counter $c_{ntri}$ that indicates the difference between the number of white messages it has sent and received before recording its snapshot, i.e., number of messages still in transit.
  - It reports this value to the initiator along with its snapshot and forwards all white messages it receives henceforth, to the initiator.
  - Snapshot collection terminates when the initiator has received $\sum_i c_{ntri}$ number of forwarded white messages.

- Second method
  - Each red message sent by a process piggybacks the value of the number of white messages sent on that channel before the local state recording. Each process keeps a counter for the number of white messages received on each channel.
  - Termination – Process receives as many white messages on each channel as the value piggybacked on red messages received on that channel.
Computing Global States without FIFO Assumption: Mattern’s Algorithm

- Uses Vector Clocks
  - All process agree on some future virtual time \( s \) or a set of virtual time instants \( s_1, \ldots, s_n \) which are mutually concurrent and did not yet occur
  - A process takes its local snapshot at virtual time \( s \)
  - After time \( s \) the local snapshots are collected to construct a global snapshot
    - \( P_i \) ticks and then fixes its next time \( s = C_i + (0, \ldots, 0, 1, 0, \ldots, 0) \) to be the common snapshot time
    - \( P_i \) broadcasts \( s \)
    - \( P_i \) blocks waiting for all the acknowledgements
    - \( P_i \) ticks again (setting \( C_i = s \)), takes its snapshot and broadcast a dummy message (i.e. force everybody else to advance their clocks to a value \( \geq s \))
    - Each process takes its snapshot and sends it to \( P_i \) when its local clock becomes \( \geq s \)
Computing Global States without FIFO Assumption (Mattern cont)

- Inventing a n+1 virtual process whose clock is managed by $P_i$
- $P_i$ can use its clock and because the virtual clock $C_{n+1}$ ticks only when $P_i$ initiates a new run of snapshot:
  - The first n components of the vector can be omitted
  - The first broadcast phase is unnecessary
  - Counter modulo 2
- Termination
  - Distributed termination detection algorithm [Mattern 87]

Optional video: has detailed example illustrating the challenge of capturing global snapshots.  
https://www.youtube.com/watch?v=ao58xine3jM