Snapshots
LECTURE A
WHAT IS A GLOBAL SNAPSHOT?

Indranil Gupta (Indy)
University of Illinois
Here's a Snapshot

Sommet de Paris
pour le soutien au peuple libyen
Samedi 19 mars 2011
Distributed Snapshot

• More often, each country’s representative is sitting in their respective capital, and sending messages to each other (say emails).
• How do you calculate a “global snapshot” in that distributed system?
• What does a “global snapshot” even mean?
In the Cloud

- In a cloud: each application or service is running on multiple servers
- Servers handling concurrent events and interacting with each other
- The ability to obtain a “global photograph” of the system is important
- Some uses of having a global picture of the system
  - 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's a Global Snapshot?**

- **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

- And the instantaneous state of each communication channel, i.e., *messages* in transit on the channels
Obvious First Solution

• Synchronize clocks of all processes
• Ask all processes to record their states at known time $t$
• Problems?
  – Time synchronization always has error
    • Your bank might inform you, “We lost the state of our distributed cluster due to a 1 ms clock skew in our snapshot algorithm.”
  – Also, does not record the state of messages in the channels
• Again: synchronization not required – causality is enough!
Example
Global Snapshot 0

P_i

C_{ij} [$1000, 100 iPhones]

C_{ji} [empty]

P_j

[$600, 50 Androids]

[Global Snapshot 0]
Global Snapshot 1

[$299, Order Android ]

[$701, 100 iPhones] 

[empty]

[$600, 50 Androids]

[Global Snapshot 1]
Global Snapshot 2

- $701, 100 iPhones
- $101, 50 Androids

- $299, Order Android
- $499, Order iPhone

C_{ij}

P_i

C_{ji}

P_j

[Global Snapshot 2]
Global Snapshot 3

\[ C_{ij} \]

\[ \text{[$1200, 1 iPhone order from \(P_j\), 100 iPhones]} \]

\[ \text{[empty]} \]

\[ C_{ji} \]

\[ \text{[$101, 50 Androids]} \]

\[ \text{[Global Snapshot 3]} \]
[$1200, 99 iPhones]

[empty]

[($299, Order Android), (1 iPhone)]

[$101, 50 Androids]

[Global Snapshot 4]
(1 iPhone)

[Global Snapshot 5]

$1200, 99 iPhones$

$400, 1 Android order from $P_i$

50 Androids
\begin{itemize}
\item \textbf{Global Snapshot 6}
\item \(C_{ij}\) \text{[$1200, 99 iPhones$]}
\item \(C_{ji}\) \text{[empty]}
\item \text{[$400, 1 Android order from \(P_i\), 50 Androids, 1 iPhone$]}
\item \text{[Global Snapshot 6]}
\end{itemize}

... and so on ...
Moving from State to State

• Whenever an event happens anywhere in the system, the global state changes
  – Process receives message
  – Process sends message
  – Process takes a step

• State to state movement obeys causality
  – Next: Causal algorithm for Global Snapshot calculation
Snapshots

Lecture B

Global Snapshot Algorithm

Indranil Gupta (Indy)
University of Illinois
**Problem:** Record a global snapshot (state for each process, and state for each channel)

**System Model:**
- \( N \) processes in the system
- There are two uni-directional communication channels between each ordered process pair: \( P_j \rightarrow P_i \) and \( P_i \rightarrow P_j \)
- Communication channels are FIFO-ordered
  - First in First out
- No failure
- All messages arrive intact, and are not duplicated
  - Other papers later relaxed some of these assumptions
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
  – We’ll assume just one snapshot run for now
Chandy-Lamport Global Snapshot Algorithm

- First, Initiator Pi records its own state
- Initiator process creates special messages called “Marker” messages
  - Not an application message, does not interfere with application messages
- for $j=1$ to $N$ except $i$
  - Pi sends out a Marker message on outgoing channel $C_{ij}$
  - $(N-1)$ channels
- Starts recording the incoming messages on each of the incoming channels at Pi: $C_{ji}$ (for $j=1$ to $N$ except $i$)
Whenever a process $P_i$ receives a Marker message on an incoming channel $C_{ji}$

- **if** (this is the first Marker $P_i$ is seeing)
  - $P_i$ records its own state first
  - Marks the state of channel $C_{ji}$ as “empty”
  - for $j=1$ to $N$ except $i$
    - $P_i$ sends out a Marker message on outgoing channel $C_{ij}$
    - Starts recording the incoming messages on each of the incoming channels at $P_i$: $C_{ji}$ (for $j=1$ to $N$ except $i$)

- **else** // already seen a Marker message
  - Mark the state of channel $C_{ji}$ as all the messages that have arrived on it since recording was turned on for $C_{ji}$
The algorithm terminates when

- All processes have received a Marker
  - To record their own state
- All processes have received a Marker on all the $(N-1)$ incoming channels at each
  - To record the state of all channels

Then, (if needed), a central server collects all these partial state pieces to obtain the full global snapshot
Example

Instruction or Step

Message
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 $S_3$
- Mark $C_{13}$ state as empty
- Turn on recording on other incoming $C_{23}$
- Send out Markers
S1, Record C21, C31

- S3
- C13 = <>
- Record C23
S1, Record C₂₁, C₃₁

State of channel C₃₁ = <>

Duplicate Marker!

- S3
- C₁₃ = <>
- Record C₂₃
S1, Record $C_{21}, C_{31}$

$C_{31} = <>$

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

$C_{31} = <>$

S2

$C_{32} = <>$

Duplicate!

$C_{12} = <>$

Record $C_{13} = <>$

$C_{23}$
\begin{itemize}
  \item S1, Record $C_{21}, C_{31}$
  \item $C_{21} = \langle \text{message } G \rightarrow D \rangle$
  \item $C_{31} = \langle \rangle$
  \item S2
  \item $C_{32} = \langle \rangle$
  \item $C_{12} = \langle \rangle$
  \item Record $C_{13}$
  \item $C_{13} = \langle \rangle$
  \item S3
  \item $C_{21} = \langle \rangle$
  \item Duplicate!
\end{itemize}
S1, Record $C_{21}, C_{31}$

$C_{31} = \langle \rangle$

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

$S2$

$C_{12} = \langle \rangle$

$C_{32} = \langle \rangle$

$S3$

$C_{13} = \langle \rangle$

Record $C_{23}$

Record $C_{72}$

Duplicate!
Algorithm has Terminated

- S1
- $C_{13} = <$>
- $C_{31} = <$>
- $C_{21} = <message G \rightarrow D >$
- S2
- $C_{32} = <$>
- $C_{12} = <$>
- $C_{23} = <$>
Collect the Global Snapshot Pieces

S1

C21 = <message G→D>
C31 = <>

S2

C32 = <>

S3

C13 = <>

C12 = <>
C23 = <>
• Global Snapshot calculated by Chandy-Lamport algorithm is **causally correct**
  – What?
Snapshots

LECTURE C

CONSISTENT CUTS

Indranil Gupta (Indy)
University of Illinois
Cuts

- **Cut** = time frontier at each process and at each channel
- Events at the process/channel that happen before the cut are “in the cut”
  - And happening after the cut are “out of the cut”
**Consistent Cuts**

**Consistent Cut**: a cut that obeys causality

- A cut $C$ is a consistent cut if and only if:
  - for (each pair of events $e$, $f$ in the system)
  - Such that event $e$ is in the cut $C$, and if $f \rightarrow e$ ($f$ happens-before $e$)
    - Then: Event $f$ is also in the cut $C$
**Example**

Consistent Cut

Inconsistent Cut

G $\rightarrow$ D, but only D is in cut
**OUR GLOBAL SNAPSHOT EXAMPLE ...**

- **P1**
  - Points: A, B, C, D, E
  - Time: S1
  - Conditions: C_{13} = < >, C_{31} = < >

- **P2**
  - Points: E, F, G
  - Time: S2
  - Conditions: C_{32} = < >, C_{12} = < >, C_{23} = < >

- **P3**
  - Points: H, I, J
  - Time: S3
  - Conditions: C_{13} = < >
... IS CAUSALLY CORRECT

Consistent Cut captured by our Global Snapshot Example

- S3
- \( C_{13} = <> \)
- S2
- \( C_{32} = <> \)
- \( C_{12} = <> \)
- \( C_{23} = <> \)
In fact...

• Any run of the Chandy-Lamport Global Snapshot algorithm creates a consistent cut
Let’s quickly look at the proof

• Let $e_i$ and $e_j$ be events occurring at $P_i$ and $P_j$, respectively such that
  
  – $e_i \rightarrow e_j$ (\(e_i\) happens before \(e_j\))

• The snapshot algorithm ensures that

  \[\text{if } e_j \text{ is in the cut then } e_i \text{ is also in the cut.}\]

• That is: if $e_j \rightarrow <P_j \text{ records its state}>$, then
  
  – it must be true that $e_i \rightarrow <P_i \text{ records its state}>$. 

---

**Chandy-Lamport Global Snapshot Algorithm creates a consistent cut**
Chandy-Lamport Global Snapshot Algorithm creates a consistent cut

- if \( e_j \rightarrow <Pj \text{ records its state}> \), then it must be true that \( e_i \rightarrow <Pi \text{ records its state}> \).
  - By contradiction, suppose \( e_j \rightarrow <Pj \text{ records its state}> \) and \( <Pi \text{ records its state}> \rightarrow e_i \)
  - Consider the path of app messages (through other processes) that go from \( e_i \rightarrow e_j \)
  - Due to FIFO ordering, markers on each link in above path will precede regular app messages
  - Thus, since \( <Pi \text{ records its state}> \rightarrow e_i \), it must be true that \( Pj \) received a marker before \( e_j \)
  - Thus \( e_j \) is not in the cut => contradiction
• What is the Chandy-Lamport algorithm used for?
Snapshots

LECTURE D
SAFETY AND LIVENESS

Indranil Gupta (Indy)
University of Illinois
"Correctness" in Distributed Systems

- Can be seen in two ways
- Liveness and Safety
- Often confused – it’s important to distinguish from each other
Liveness = guarantee that something **good** will happen, **eventually**

- Eventually == does not imply a time bound, but if you let the system run long enough, then …
**Liveness: Examples**

- **Liveness** = guarantee that something **good** will happen, eventually
  - Eventually == does not imply a time bound, but if you let the system run long enough, then …

- **Examples in Real World**
  - Guarantee that “at least one of the atheletes in the 100m final will win gold” is liveness
  - A criminal will eventually be jailed

- **Examples in a Distributed System**
  - Distributed computation: Guarantee that it will terminate
  - “Completeness” in failure detectors: every failure is eventually detected by some non-faulty process
  - In Consensus: All processes eventually decide on a value
Safety

- Safety = guarantee that something bad will never happen
Safety: Examples

• Safety = guarantee that something bad will never happen

• Examples in Real World
  – A peace treaty between two nations provides safety
    • War will never happen
  – An innocent person will never be jailed

• Examples in a Distributed System
  – There is no deadlock in a distributed transaction system
  – No object is orphaned in a distributed object system
  – “Accuracy” in failure detectors
  – In Consensus: No two processes decide on different values
Can't we Guarantee both?

• Can be difficult to satisfy both liveness and safety in an asynchronous distributed system!
  – Failure Detector: Completeness (Liveness) and Accuracy (Safety) cannot both be guaranteed by a failure detector in an asynchronous distributed system
  – Consensus: Decisions (Liveness) and correct decisions (Safety) cannot both be guaranteed by any consensus protocol in an asynchronous distributed system
  – Very difficult for legal systems (anywhere in the world) to guaranteed that all criminals are jailed (Liveness) and no innocents are jailed (Safety)
In the language of Global States

• Recall that a distributed system moves from one global state to another global state, via causal steps

• Liveness w.r.t. a property $Pr$ in a given state $S$ means
  – $S$ satisfies $Pr$, or there is some causal path of global states from $S$ to $S'$ where $S'$ satisfies $Pr$

• Safety w.r.t. a property $Pr$ in a given state $S$ means
  $S$ satisfies $Pr$, and all global states $S'$ reachable from $S$ also satisfy $Pr$
Using Global Snapshot Algorithm

- Chandy-Lamport algorithm can be used to detect global properties that are **stable**
  - Stable = once true, stays true forever afterwards
- **Stable Liveness examples**
  - Computation has terminated
- **Stable Non-Safety examples**
  - There is a deadlock
  - An object is orphaned (no pointers point to it)
- **All stable global properties can be detected using the Chandy-Lamport algorithm**
  - Due to its causal correctness
The ability to calculate global snapshots in a distributed system is very important.

But don’t want to interrupt running distributed application.

Chandy-Lamport algorithm calculates global snapshot.

Obey causality (creates a consistent cut).

Can be used to detect stable global properties.

Safety vs. Liveness.