Slides for Chapter 14:
Time and Global States

From Coulouris, Dollimore, Kindberg and Blair
Distributed Systems: Concepts and Design
Edition 5, © Addison-Wesley 2012
Overview of Chapter

- Introduction
- Clocks, events, process states
- Synchronizing physical clocks
- Logical time and logical clocks
- Global states (skip)
- Distributed debugging (skip)
Introduction

• Some applications need to record the time when events occur (e.g. ecommerce, banking)
• May need to determine the relative order in which certain events occurred
• Physical events order based on observer’s frame of reference
• Multiple computer clocks may be skewed and need to be synchronized – no absolute global time for all computers in a distributed systems
Overview of Chapter

• Introduction
• Clocks, events, process states
• Synchronizing physical clocks
• Logical time and logical clocks
• Global states
• Distributed debugging
Clocks, events, process states

• Each process can observe/cause multiple events
• Some events may change process states (data)
• History of a process: the series of events that take place within the process ordered in a total ordering

Clocks:
• Each computer has a physical clock
• Timestamp: date/time that an event occurred
• Clock skew: instantaneous difference between two clocks
• Clock drift: different clocks count time at slightly different speeds
• UTC (Coordinated Universal Time): based on atomic time – international standard for time keeping
Figure 14.1
Skew between computer clocks in a distributed system
Overview of Chapter

• Introduction
• Clocks, events, process states
• Synchronizing physical clocks
• Logical time and logical clocks
• Global states
• Distributed debugging
Synchronizing physical clocks

External synchronization:
• Synchronizes each clock in the distributed system with a UTC source – clocks must be within drift bound D of UTC

Internal synchronization:
• Synchronizes the clocks in the distributed system with one another – any two physical clocks must be within drift bound D of one another
• May drift from UTC but are synchronized together

• Faulty clock: does not stay within specified drift bound
Synchronizing physical clocks

Synchronization in a synchronous system:

• Synchronous system has upper bound on message transmission time $max$ – also $min$ is the minimum message transmission time between two machines

• A message sent from node $p$ at (local clock) time $t$ arrives at another node $q$ at (local clock) time $t'$

• Can now set clock at node $q$ to $t + (max + min)/2$ to synchronize clock at node $q$ with clock in node $p$
Synchronizing physical clocks using a time server node

NTP (Network Time Protocol):
- Synchronize clock at each node with clock at time server
- Assumes asynchronous system

Christian’s algorithm:
- Process p requests time from time server in message $m_r$, receives time value t in message $m_t$
- $P$ records round trip time $T_{\text{round}}$ between sending and receiving
- $P$ sets local clock time to $t + T_{\text{round}} / 2$
- Probabilistic method
- All processes (nodes) synchronize with clock server node
- Used for synchronizing nodes on a local intranet
- Variation called Berkeley algorithm
Figure 14.2
Clock synchronization using a time server

\[ p \rightarrow m_r \rightarrow m_t \rightarrow S \]
Synchronizing physical clocks using NTP

NTP (Network Time Protocol):
- Synchronize clocks of client nodes on the internet with UTC
- Employs statistical techniques to factor in network latency

Other features:
- Can survive lengthy losses of connectivity
- Enables clients to synchronize frequently to offset drift
- Protects against interference with time services

Three modes of synchronization:
- Multicast mode (on a high speed LAN)
- Procedure call mode (similar to Christian’s algorithm)
- Symmetric mode (used by time servers)
Figure 14.3
An example synchronization subnet in an NTP implementation

Note: Arrows denote synchronization control, numbers denote strata.
Figure 14.4
Messages exchanged between a pair of NTP peers
Overview of Chapter

• Introduction
• Clocks, events, process states
• Synchronizing physical clocks
• Logical time and logical clocks
• Global states
• Distributed debugging
Logical time and logical clocks

Ordering events that occur in different processes (nodes):
• Within each process $p_i$ events are ordered ($\rightarrow i$)
• Sending a message from one process occurs before the message is received at another process

Happened-before relationship $\rightarrow$ based on above two observations:
• If $e \rightarrow i e'$, then $e \rightarrow e'$
• For any message $m$, $send(m) \rightarrow receive(m)$
• If $e \rightarrow e'$ and $e' \rightarrow e''$, then $e \rightarrow e''$

In next figure:
• $a \rightarrow b$, $c \rightarrow d$ (same process); $b \rightarrow c$, $d \rightarrow f$ (send/receive)
• $a || e$ (concurrent; cannot tell which occurred first)
Figure 14.5
Events occurring at three processes
Logical time and logical clocks

Logical clock (based on Lamport timestamp):

- Timestamp of event e at pi denoted Li(e)
- Timestamp of event e at p denoted L(e)

Timestamp rules:

- Li := Li + 1 after each event at pi
- When pi sends m it piggybacks timestamp t of *send* as t = Li and sends (m, t)
- When pj receives (m, t) it computes Lj = max (Lj, t) then increments Lj as timestamp of *receive*
Figure 14.6
Lamport timestamps for the events shown in Figure 14.5
Logical time and logical clocks

Totally ordered logical clock:
• Events can have same clock at two processes pi, pj
• By appending process id \((t, i)\), a total order is achieved
• Order of processes not significant

Vector clocks:
• Keep a vector \(V_i[j]\), \(j = 1, 2, \ldots, n\) of all time points of processes that have communicated with pi
• Process vector always piggybacked with a sent message
• Receiving process can use piggybacked vector to update its vector clock
Figure 14.7
Vector timestamps for the events shown in Figure 14.5
Overview of Chapter

• Introduction
• Clocks, events, process states
• Synchronizing physical clocks
• Logical time and logical clocks
• Global states (skip)
• Distributed debugging
Figure 14.8
Detecting global properties

(a) Garbage collection

(b) Deadlock

(c) Termination
Figure 14.9
Cuts

Inconsistent cut

Consistent cut

Physical time
Marker receiving rule for process $p_i$

On $p_i$’s receipt of a marker message over channel $c$:

- If ($p_i$ 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 incoming channels;

- Else
  $p_i$ records the state of $c$ as the set of messages it has received over $c$
  since it saved its state.

Marker sending rule for process $p_i$

After $p_i$ has recorded its state, for each outgoing channel $c$:

- $p_i$ sends one marker message over $c$
  (before it sends any other message over $c$).
Figure 14.11
Two processes and their initial states

Diagram:

- $p_1$ with $\$1000$ in account, $(\text{none})$ in widgets
- $p_2$ with $\$50$ in account, $2000$ widgets
- Communication channels $c_1$ and $c_2$
Figure 14.12
The execution of the processes in Figure 14.11

1. Global state $S_0$

\begin{align*}
\text{<}$1000, 0$\text{>} & \quad \text{p}_1 & \quad \text{c}_2 & \quad \text{(empty)} \\
& & & & \quad \text{p}_2 & \quad \text{<}$50, 2000$\text{>}
\end{align*}

\begin{align*}
\text{c}_1 & \quad \text{(empty)}
\end{align*}

2. Global state $S_1$

\begin{align*}
\text{<}$900, 0$\text{>} & \quad \text{p}_1 & \quad \text{c}_2 & \quad \text{(Order 10, $100$), M} \\
& & & & \quad \text{p}_2 & \quad \text{<}$50, 2000$\text{>}
\end{align*}

\begin{align*}
\text{c}_1 & \quad \text{(empty)}
\end{align*}

3. Global state $S_2$

\begin{align*}
\text{<}$900, 0$\text{>} & \quad \text{p}_1 & \quad \text{c}_2 & \quad \text{(Order 10, $100$), M} \\
& & & & \quad \text{p}_2 & \quad \text{<}$50, 1995$\text{>}
\end{align*}

\begin{align*}
\text{c}_1 & \quad \text{(five widgets)}
\end{align*}

4. Global state $S_3$

\begin{align*}
\text{<}$900, 5$\text{>} & \quad \text{p}_1 & \quad \text{c}_2 & \quad \text{(Order 10, $100$)} \\
& & & & \quad \text{p}_2 & \quad \text{<}$50, 1995$\text{>}
\end{align*}

\begin{align*}
\text{c}_1 & \quad \text{(empty)}
\end{align*}

$(M = \text{marker message})$
Figure 14.13
Reachability between states in the snapshot algorithm

- Actual execution: $e_0, e_1, \ldots$
- Recording begins
- Recording ends
- Pre-snap: $e'_0, e'_1, \ldots e'_{R-1}$
- Snap: $S_{\text{snap}}$
- Post-snap: $e'_R, e'_{R+1}, \ldots$

$S_{\text{init}} \rightarrow$ recording begins $\rightarrow S_{\text{snap}} \rightarrow$ recording ends $\rightarrow S_{\text{final}}$
Overview of Chapter

- Introduction
- Clocks, events, process states
- Synchronizing physical clocks
- Logical time and logical clocks
- Global states
- Distributed debugging (skip)
Figure 14.14
Vector timestamps and variable values for the execution of Figure 14.9
Figure 14.15
The lattice of global states for the execution of Figure 14.14

\[ S_{ij} = \text{global state after } i \text{ events at process 1 and } j \text{ events at process 2} \]
Figure 14.16
Algorithms to evaluate possibly and definitely

1. Evaluating possibly $\phi$ for global history $H$ of $N$ processes
   
   $L := 0$;
   
   $States := \{ (s_1^0, s_2^0, ..., s_N^0) \}$;
   
   while ($\phi(S) = False$ for all $S \in States$)
     
     $L := L + 1$;
     
     $Reachable := \{ S' : S'$ reachable in $H$ from some $S \in States \land level(S') = L \}$;
     
     $States := Reachable$
   
   end while
   
   output "possibly $\phi$"

2. Evaluating definitely $\phi$ for global history $H$ of $N$ processes

   $L := 0$;
   
   if ($\phi(s_1^0, s_2^0, ..., s_N^0)$) then $States := \{ \}$ else $States := \{ (s_1^0, s_2^0, ..., s_N^0) \}$;
   
   while ($States \neq \{ \}$)
     
     $L := L + 1$;
     
     $Reachable := \{ S' : S'$ reachable in $H$ from some $S \in States \land level(S') = L \}$;
     
     $States := \{ S \in Reachable : \phi(S) = False \}$
   
   end while
   
   output "definitely $\phi$"
Figure 14.17
Evaluating \textit{definitely}

\begin{itemize}
\item Level 0
\item 1
\item 2
\item 3
\item 4
\item 5
\end{itemize}

\[
F = (\phi(S) = \text{False}); \quad T = (\phi(S) = \text{True})
\]