

Agenda

- **Last time**
 - Midterm discussion
 - More Chapter 11 (time and global states)
- **This time**
 - Finish Chapter 11 Time and global states
 - Chpt 12: Coordination and Agreement
- **Next time (Tue Apr 10)**
 - Chpt 12: Coordination and Agreement
- **Remember: make-up class Thurs Apr 19/Fri Apr 20**

CS451: Distributed Systems (Spring 2007)

Before we start: Schedule

Sun	Tues	Thurs	Fri
	Time/global (11) 3	Coordination/agreement (12) PA#4 "Comparable project" due 5	
	Coordination/agreement Assignment#5 Out 10	Transactions (13) 12	
	Transactions (13) Dis transactions (14) PA#4 due 17	Dis transactions (14) Replication (15) Replication (15) 19	Replication (15)
	Google day PA#5 due 24	No class – Marty out of town 26	
	Course wrap-up 1	3	
	8	10	Final – 2-5pm 11

CS451: Distributed Systems (Spring 2007)

Apr 24: Google Day!

- **Possible papers**
 - The Anatomy of a Large-Scale Hypertextual Web Search Engine (Computer Networks, 1998)
 - Web Search for a Planet: The Google Cluster Architecture (IEEE Micro, 2003)
 - MapReduce: Simplified Data Processing on Large Clusters (OSDI 2004)
 - Google File System (SOSP 2003)
 - Bigtable: A Distributed Storage System for Structured Data (OSDI 2006)
- **Interesting, but not explicitly covered:**
 - How to design a Good API and Why it matters (OOPSLA'06)

CS451: Distributed Systems (Spring 2007)

Chapter 11: Time and Global States: Roamap

- **Problem definition:** what would happen if there's no attempt to synchronize clocks in a distributed system?
- **Ground rules:** Introduce a general model of computing (process, state, "happens before", HW clock, etc.)
- "approximate synchronization" of clocks
 - (internal) Synchronization in a synchronous system
 - Christian's method (external)
 - Berkeley alg. (internal)
 - NTP (external)
- **Logical time and logical clocks**
 - Happened-before
 - Logical clocks ← we are here
- **Global states**

CS451: Distributed Systems (Spring 2007)

From last time

- **We can establish a partial ordering of events in a distributed system based on "happened before"**
 - For any two events a and b , if we cannot establish that $a \rightarrow b$ or $b \rightarrow a$, then a and b are said to be "concurrent"
- **We can add a new other constraints and create a total ordering through the definition of a "logical clock"....**

CS451: Distributed Systems (Spring 2007)

Time & Logical Clocks (5)

- Logical clock L_i of process P_i = (monotonic) function that assigns a numeric value $L_i(a)$ to each event a of P_i
- No particular relationship between logical and physical clocks
- Set of all logical clocks of all processes: can be represented by one function L
- Logical clock system is said to be **correct**, if it is consistent with the relation \rightarrow , satisfies the condition:

For all events a and b , if $a \rightarrow b$ then $L(a) < L(b)$
- Implementing logical clocks (Lamport, 1978): use of timestamps

CS451: Distributed Systems (Spring 2007)

Time & Logical Clocks (6)

- Process P_i : updates its logical clock and transmits its value within messages as follows

LC1: L_i is incremented before each event is issued at P_i

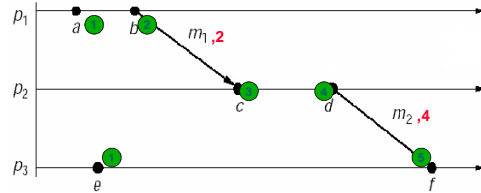
LC2: (a) When P_i sends a message m , it piggybacks on m a timestamp $T_m = L_i$

(b) On receiving (m, T_m) , a process P_j computes $L_j = \text{Max}(L_j, T_m)$ and then applies LC1 before timestamping the event receive(m)

$a \rightarrow b \Rightarrow L(a) < L(b)$ (converse not true, $L(a) < L(b) \not\Rightarrow a \rightarrow b$)

CS451: Distributed Systems (Spring 2007)

Time & Logical Clocks (7)



Note: $L(b) > L(e)$ but $b \parallel e$

CS451: Distributed Systems (Spring 2007)

Total Ordering of Events (1)

- Using of logical clocks to define a total order
- Simple method: classify messages according to their timestamps
- Problem:** logical clock system may assign the same timestamp to different events

Solution: use a conventional order of processes

- Total relation \Rightarrow : for all events a and b occurred in two processes P_i and P_j

$a \Rightarrow b$ IFF $\begin{cases} 1) L_i(a) > L_j(b), \text{ or} \\ 2) L_i(a) = L_j(b) \text{ and } P_i < P_j \end{cases}$ in the conventional order of processes

CS451: Distributed Systems (Spring 2007)

Global States (1)

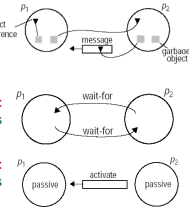
- Objective:** find out whether a particular property is true of a distributed system as it executes

Distributed garbage collection: An object is considered as garbage if there are no longer references to it anywhere

Distributed deadlock detection: detect if a collection of processes waits for another to send it a message

Distributed termination detection: detect that a distributed algorithm has terminated

Distributed debugging



CS451: Distributed Systems (Spring 2007)

Global States (2)

- If we had a single clock, we could snapshot all the processes at the same time and thus have our global state

- System ξ : composed of N processes P_i ($i = 1, \dots, N$)

"Event" = either internal event in process, or sending a msg, or receiving a msg

History(P_i) = $h_i = \langle e_i^0, e_i^1, e_i^2, \dots \rangle$

$h_i^k = \langle e_i^0, e_i^1, e_i^2, \dots, e_i^k \rangle$: finite history

S_i^k = state of process P_i immediately before the k^{th} event occurs, S_i^0 = initial state of P_i

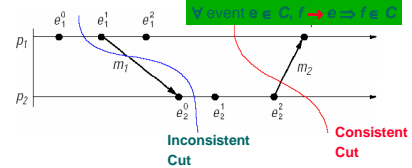
$H = H_1 \cup H_2 \cup \dots \cup H_N$: global history

$S = (S_1, S_2, \dots, S_N)$: global state of system ξ

CS451: Distributed Systems (Spring 2007)

Global States (3)

- Cut:** subset of the global history $C = h_1^c \cup h_2^c \cup \dots \cup h_N^c$ of a system's execution
- Cut C is consistent:** for each event it contains, it also contains all the events that happened-before that event:



- A global consistent state corresponds to a consistent cut

CS451: Distributed Systems (Spring 2007)

Global States (4)

- **Snapshot algorithm (Chandy & Lamport 1985):** determines global states of distributed systems
- **Basic hypotheses:**
 - Neither channels nor processes fail. Reliable communication
 - Channels are unidirectional and provide FIFO-ordered message delivery
 - Graph of processes and channels is strongly connected (there is a path between any two processes)
 - Any process can initiate a global snapshot at any time
 - Processes: may continue their execution and send and receive normal messages while the snapshot takes place

CS451: Distributed Systems (Spring 2007)

Global States (5)

- **Snapshot algorithm (cont'd):**

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) **then**
 [1] it records its process state now;
 [2] records the state of C as the empty set;
 [3] turns on recording of messages arriving over other incoming channels;
else
 [1] P_i records the state of C as the set of messages it has received over C since it saved its state.

CS451: Distributed Systems (Spring 2007)

Global States (6)

- **Snapshot algorithm (cont'd):**

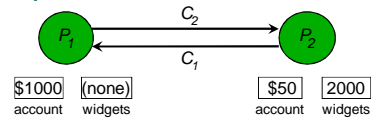
Marker sending rule for process P_i

After P_i has recorded its state, for each outgoing channel C:
 [1] P_i sends one marker message over C
 (before it sends any other message over C).

CS451: Distributed Systems (Spring 2007)

Global States (7)

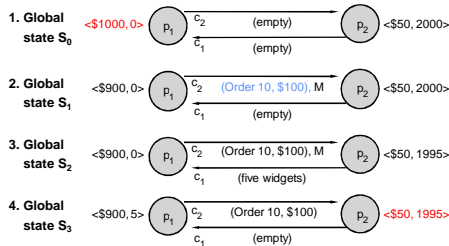
- **Example:**



- **Initial state:** P_2 has received order for 5 widgets @ \$10/each; P_2 has not filled this order yet
- **Now let's record the global state...** (initiated by P_1 , which acts as if it receives a marker over an imaginary link...)

CS451: Distributed Systems (Spring 2007)

Global States (8)



Final state = $\{p_1: \langle \$1000, 0 \rangle; p_2: \langle \$50, 1995 \rangle; c_1: \langle \text{five widgets} \rangle; c_2: \langle \rangle\}$

NOTE: system was never actually in this state!
 Can you prove: [a] that this algorithm terminates?, [b] that the cut is consistent?

CS451: Distributed Systems (Spring 2007)