

Lock and Time

THOAI NAM

**Faculty of Information Technology
HCMC University of Technology**

Using some slides of Prashant Shenoy,
UMass Computer Science



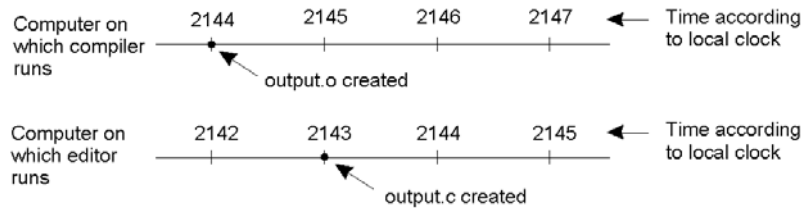
Chapter 3: Lock and Time

- Time ordering and clock synchronization
- Virtual time (logical clock)
- Distributed snapshot (global state)
- Consistent/Inconsistent global state
- Rollback Recovery



Clock Synchronization

- Time is unambiguous in centralized systems
 - System clock keeps time, all entities use this for time
- Distributed systems: each node has own system clock
 - Crystal-based clocks are less accurate (1 part in million)
 - *Problem*: An event that occurred after another may be



Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Physical Clocks: A Primer

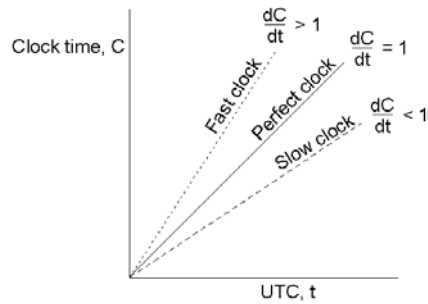
- Accurate clocks are atomic oscillators
 - $1s \sim 9,192,631,770$ transitions of the cesium 133 atom
- Most clocks are less accurate (e.g., mechanical watches)
 - Computers use crystal-based blocks (one part in million)
 - Results in *clock drift*
- How do you tell time?
 - Use astronomical metrics (solar day)
- Universal coordinated time (*UTC*) – international standard based on atomic time
 - Add leap seconds to be consistent with astronomical time
 - UTC broadcast on radio (satellite and earth)
 - Receivers accurate to 0.1 – 10 ms
- Need to synchronize machines with a master or with one another

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Clock Synchronization

- Each clock has a maximum drift rate ρ
 - » $1-\rho \leq dC/dt \leq 1+\rho$
 - Two clocks may drift by $2\rho \Delta t$ in time Δt
 - To limit drift to $\delta \Rightarrow$ resynchronize every $\delta/2\rho$ second

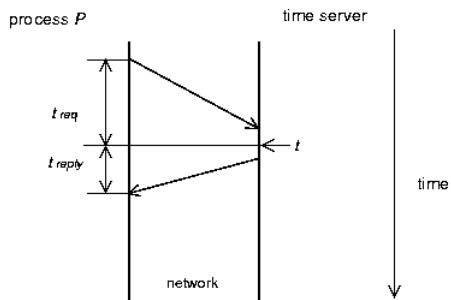


Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Cristian's Algorithm

- Synchronize machines to a *time server* with a UTC receiver
- Machine P requests time from server every $\delta/2\rho$ seconds
 - Receives time t from server, P sets clock to $t+t_{reply}$ where t_{reply} is the time to send reply to P
 - Use $(t_{req}+t_{reply})/2$ as an estimate of t_{reply}
 - Improve accuracy by making a series of



measurements

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



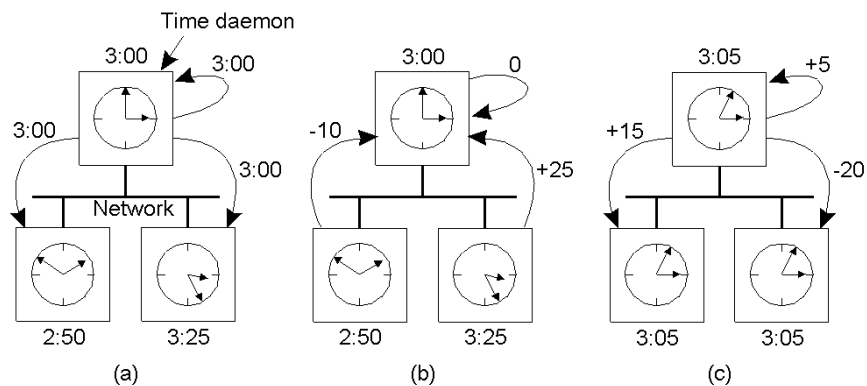
Berkeley Algorithm

- Used in systems without UTC receiver
 - Keep clocks synchronized with one another
 - One computer is *master*, other are *slaves*
 - Master periodically polls slaves for their times
 - » Average times and return differences to slaves
 - » Communication delays compensated as in Cristian's algorithm
 - Failure of master => election of a new master

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Berkeley Algorithm



- a) The time daemon asks all the other machines for their clock values
- b) The machines answer
- c) The time daemon tells everyone how to adjust their clock

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Distributed Approaches

- Both approaches studied thus far are centralized
- Decentralized algorithms: use resynchronization intervals
 - Broadcast time at the start of the interval
 - Collect all other broadcast that arrive in a period S
 - Use average value of all reported times
 - Can throw away few highest and lowest values
- Approaches in use today
 - *rdate*: synchronizes a machine with a specified machine
 - Network Time Protocol (NTP)
 - » Uses advanced techniques for accuracies of 1-50 ms

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Logical Clocks

- For many problems, internal consistency of clocks is important
 - Absolute time is less important
 - Use *logical* clocks
- Key idea:
 - Clock synchronization need not be absolute
 - If two machines do not interact, no need to synchronize them
 - More importantly, processes need to agree on the *order* in which events occur rather than the *time* at which they occurred

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Event Ordering

- *Problem*: define a total ordering of all events that occur in a system
- Events in a single processor machine are totally ordered
- In a distributed system:
 - No global clock, local clocks may be unsynchronized
 - Can not order events on different machines using local times
- Key idea [Lamport]
 - Processes exchange messages
 - Message must be sent before received
 - Send/receive used to order events (and synchronize clocks)

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Happened-Before Relation

- If A and B are events in the same process and A executed before B , then $A \rightarrow B$
- If A represents sending of a message and B is the receipt of this message, then $A \rightarrow B$
- Relation is transitive:
 - $A \rightarrow B$ and $B \rightarrow C \Rightarrow A \rightarrow C$
- Relation is undefined across processes that do not exchange messages
 - Partial ordering on events

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



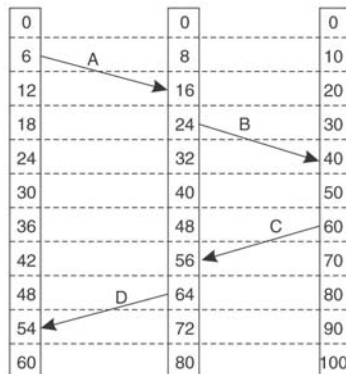
Event Ordering Using HB

- Goal: define the notion of time of an event such that
 - If $A \rightarrow B$ then $C(A) < C(B)$
 - If A and B are concurrent, then $C(A) <, =$ or $> C(B)$
- Solution:
 - Each processor maintains a logical clock LC_i
 - Whenever an event occurs locally at i , $LC_i = LC_i + 1$
 - When i sends message to j , piggyback LC_i
 - When j receives message from i
 - » ~~If $LC_j < LC_i$ then $LC_j = LC_i + 1$ else do nothing~~

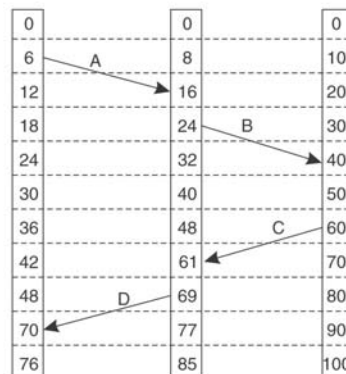
Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Lamport's Logical Clocks



(a)



(b)

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



More Canonical Problems

- Causality
 - Vector timestamps

- Global state and termination detection

- Election algorithms

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Causality

- Lamport's logical clocks
 - If $A \rightarrow B$ then $C(A) < C(B)$
 - Reverse is not true!!
 - » Nothing can be said about events by comparing time-stamps!
 - » If $C(A) < C(B)$, then ??
- Need to maintain *causality*
 - Causal delivery: If $\text{send}(m) \rightarrow \text{send}(n) \Rightarrow \text{deliver}(m) \rightarrow \text{deliver}(n)$
 - Capture causal relationships between groups of processes
 - Need a time-stamping mechanism such that:
 - » If $T(A) < T(B)$ then A should have causally preceded B

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Vector Clocks

- Each process i maintains a vector V_i
 - $V_i[i]$: number of events that have occurred at process i
 - $V_i[j]$: number of events occurred at process j that process i knows
 - Update vector clocks as follows
 - Local event: increment $V_i[i]$
 - Send a message: piggyback entire vector V
 - Receipt of a message:
 - » $V_j[i] = V_j[i] + 1$
 - » Receiver is told about how many events the sender knows occurred at another process k
 $V_j[k] = \max(V_j[k], V_i[k])$
 - *Homework*: convince yourself that if $V(A) < V(B)$, then A causally precedes B
-

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



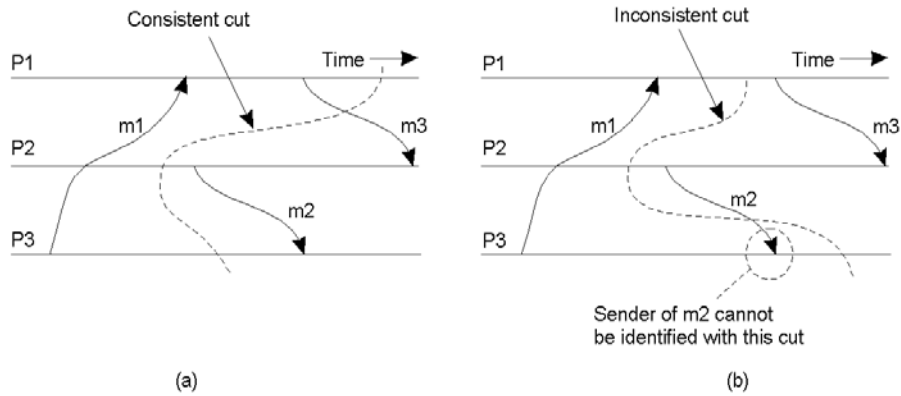
Global State

- Global state of a distributed system
 - Local state of each process
 - Messages sent but not received (state of the queues)
 - Many applications need to know the state of the system
 - Failure recovery, distributed deadlock detection
 - Problem: how can you figure out the state of a distributed system?
 - Each process is independent
 - No global clock or synchronization
 - Distributed snapshot: a consistent global state
-

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Consistent/Inconsistent Cuts



- a) A consistent cut
- b) An inconsistent cut

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Distributed Snapshot Algorithm

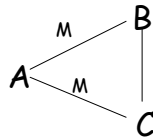
- Assume each process communicates with another process using unidirectional point-to-point channels (e.g, TCP connections)
- Any process can initiate the algorithm
 - Checkpoint local state
 - Send marker on every outgoing channel
- On receiving a marker
 - Checkpoint state if first marker and send marker on outgoing channels, save messages on all other channels until:
 - Subsequent marker on a channel: stop saving state for that channel

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Distributed Snapshot

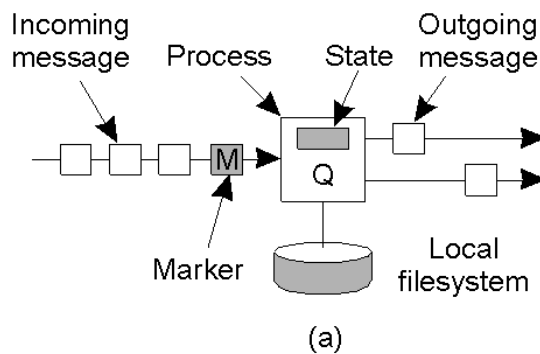
- A process finishes when
 - It receives a marker on each incoming channel and processes them all
 - State: local state plus state of all channels
 - Send state to initiator
- Any process can initiate snapshot
 - Multiple snapshots may be in progress
 - » Each is separate, and each is distinguished by tagging the marker with the initiator ID (and sequence number)



Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Snapshot Algorithm Example (1)

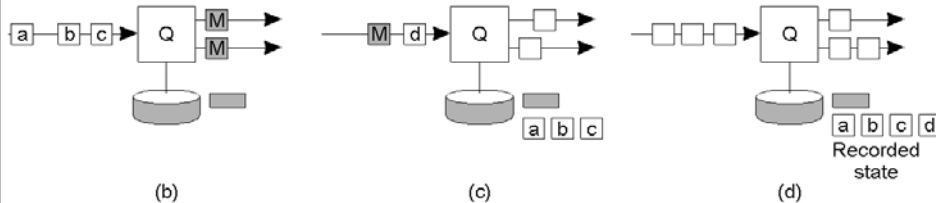


(a) Organization of a process and channels for a distributed snapshot

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Snapshot Algorithm Example (2)



- (b) Process Q receives a marker for the first time and records its local state
- (c) Q records all incoming message
- (d) Q receives a marker for its incoming channel and finishes recording the state of the incoming channel

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



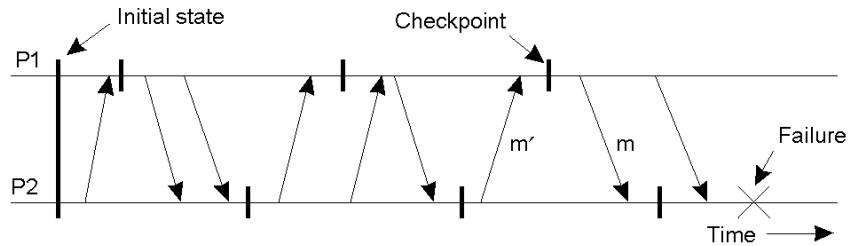
Recovery

- Techniques thus far allow failure handling
- Recovery: operations that must be performed after a failure to recover to a correct state
- Techniques:
 - Checkpointing:
 - » Periodically checkpoint state
 - » Upon a crash roll back to a previous checkpoint with a *consistent state*

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Independent Checkpointing



- Each processes periodically checkpoints independently of other processes
- Upon a failure, work backwards to locate a consistent cut
- Problem: if most recent checkpoints form inconsistent cut, will need to keep rolling back until a consistent cut is found
- Cascading rollbacks can lead to a domino effect.

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Coordinated Checkpointing

- Take a distributed snapshot
- Upon a failure, roll back to the latest snapshot
 - All process restart from the latest snapshot

Khoa Công Nghệ Thông Tin – Đại Học Bách Khoa Tp.HCM



Message Logging

- Checkpointing is expensive
 - All processes restart from previous consistent cut
 - Taking a snapshot is expensive
 - Infrequent snapshots => all computations after previous snapshot will need to be redone [wasteful]
- Combine checkpointing (expensive) with message logging (cheap)
 - Take infrequent checkpoints
 - Log all messages between checkpoints to local stable storage
 - To recover: simply replay messages from previous checkpoint

» Avoids recomputations from previous