Distributed Systems - Theoretical Foundations (pt. 2) Agreement Protocols

CS239
Advanced Operating Systems
October 31, 2000
Dr. Richard Guy

Lecture Source Materials

Singhal, Ch. 5: **Theoretical Foundations**

Chandy & Lamport: **Distributed Snapshots: Determining Global States of Distributed Systems**, ACM TOCS 3:1, Feb. `85,63-75.

Singhal, Ch. 8: **Agreement Protocols**

Lamport, Shostak, & Pease, **The Byzantine Generals Problem,** ACM TOPLAS, July 1982.

Outline

- Theoretical Foundations (part 2 of 2)
  - Summary of part 1
  - Causal ordering of messages
  - Global state
- Agreement Protocols

Theoretical Foundations

- Motivation
- Model
- Inherent limitations of distributed system
- Virtual clocks
- Causal ordering of messages
- Global state

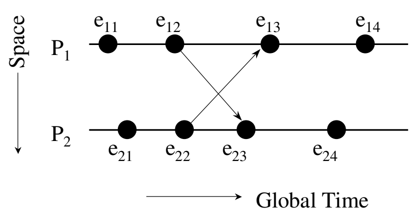
Key question

- Is it possible to obtain a coherent global state of a distributed system, given the absence of a global clock, no shared memory, and the existence of arbitrary delay communication channels?

"Virtual clock" solutions

- Lamport's logical clocks
  - limitations of Lamport's clocks
- Fidge/Mattern vector clocks

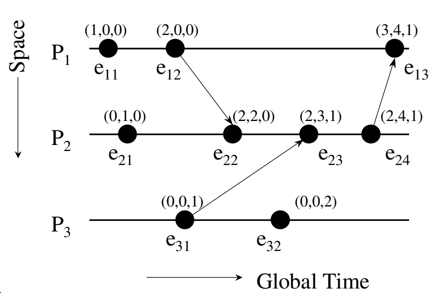
Space-time diagram



Limitations of Lamport's clock

- If $a \rightarrow b$, then $C(a) < C(b)$
- Is the reverse true?
  – if $C(a) < C(b)$, is $a \rightarrow b$ always true? NO!
- Lamport's clock cannot distinguish between clock advancement from local vs remote events
- hard to reason based on event timestamps

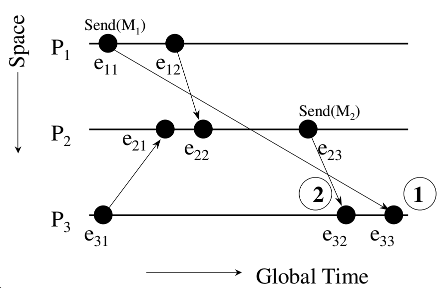
Space-time diagram



Clocks revisited

- Are vector clocks adequate?
  - better than Lamport's clocks
  - usable in practice?

Causal ordering example



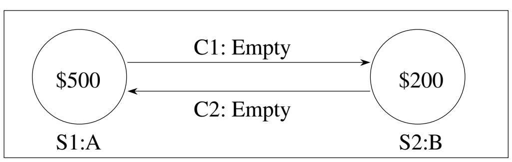
Causal ordering of messages

- Matching receive order with send order
  - typically, the desired semantics
- Protocol key concepts
  - delay (buffer) delivery until preceeding messge is known to have been delivered
  - use message-event-specific timestamp, with increment-by-one clock, to identify messages
  - Birman, et al, assumes broadcast messages
  - both protocols require reliable message delivery

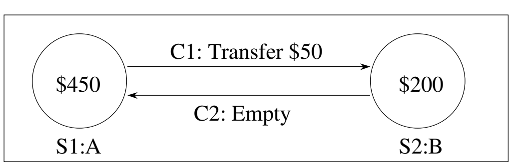
Global state

- Task: to record and collect a coherent global state in a distributed system
- Key problem: correctly recording “state” of communication channels

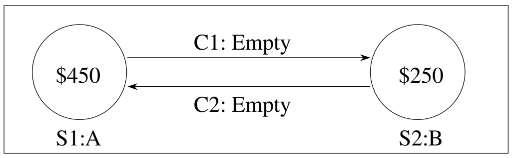
Coherent global state #1



Coherent global state #2



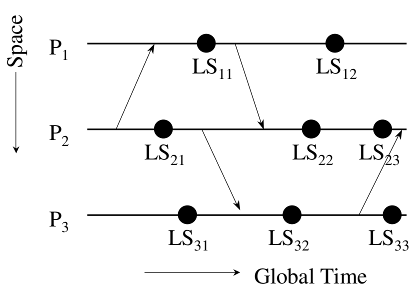
Coherent global state #3



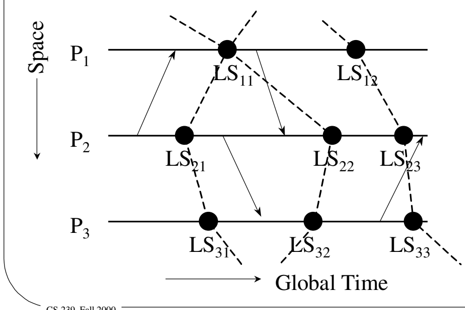
Global state defiinitions

- Local state (defined for each local time)
  - send( ), receive( )
  - transit set(i,j): sent but not received
  - inconsistent set(i,j): received but not sent
- Global state
  - consistent: for all received, corresponding send
  - transitless: all sent have been received
  - strongly consistent = consistent and transitless

Example



(In)consistent global state



Chandy-Lamport Algorithm

- Key idea: use special computation-independent marker to delimit/flush channel
- processes record local state and channel state, upon receiving marker(s)
- after receiving markers on all incoming channels, send recorded state to initiator

Algorithm details, part 1

- Marker sending/receiving rules
- *Marker sending rule for process P*
  - *P* records its local state
  - *P* sends marker along all outgoing channels, if it hasn't yet done so

Algorithm details, part 2

- *Marker receiving rule for process Q*
  - If *Q* has not yet recorded its state
    - record state of *C* as an empty sequence
    - follow the marker sending rule
  - else
    - record state of *C* as the sequence of messages received on *C* after *Q*'s state was recorded (earlier) and before *Q* received the marker along *C*.

Algorithm comments

+ Multiple initiations allowed
  - single process
  - multiple processes
+ Particularly useful for stable state detection
  - deadlock detection
  - computation termination

– Assumes FIFO channels

Theoretical Foundations Summary

- "No global clock" is important because...
- "No shared memory" is important because...
- The virtual clock approach helps because...
- Causal message ordering is useful because...
- Global state determination is used in…

Skipping material in...

- Ch. 6: Distributed Mutual Exclusion
- Ch. 7: Distribued Deadlock Detection

Agreement Protocols Outline

- Motivation
- Definition
- System Model
- Problem Classification
- Byzantine Agreement
- Applications

Motivation

- Cooperation or competition
  - distributed commit
  - unique resource reservation/usage
- Easy, if failure-free
- Harder (impossible?) with failures
  - fail-stop would be nice, but...
  - malicious process prevent agreement?

Approach

- processors exchange messages
- faulty process may
  - fail to respond/send as expected
  - send malicious messages
- nonfaulty don't know who is faulty
- all (nonfaulty) execute same protocol

System Model

- *n* processors, of which at most *m* are faulty
- system is logically fully connected
  - message passing directly between all processors
- receiver always knows identity of sender
- links don't break; only processors may fail
- agreement is binary: 0 or 1 are final values

Additional model parameters

- Synchronous vs asynchronous computation
  - rounds
  - asynchronous proven not solvable!!!
- Failure model (crash, omission, malicious)
- Authenticated vs non-authenticated "second-hand" messages
- Computational complexity
  - time, message traffic, storage overhead

Problem Classification

	Byzantine agreement	Consensus	Interactive consistency
Who initiates the value(s)?	One	All	All
Final agreement	Single	Single	Vector

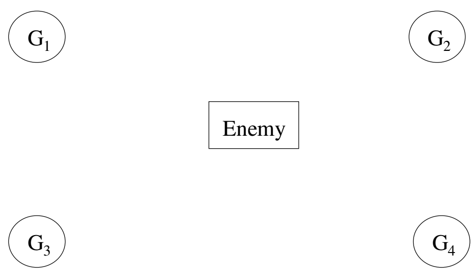
Classification comments

- All three are closely related
- "Byzantine agreement" solution can be basis for other two solutions
  - not necessarily most efficient...

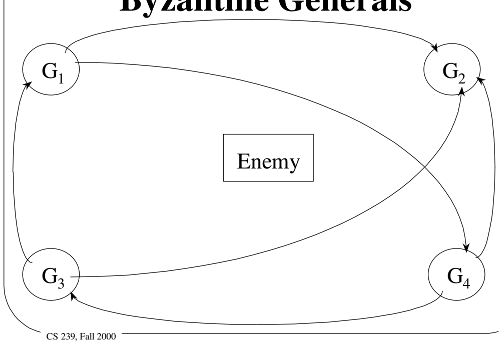
Byzantine Agreement

- Byzantine Generals
- Upper bound on |Faulty processors|
- Impossibility result
- Lamport-Shostak-Pease Algorithm

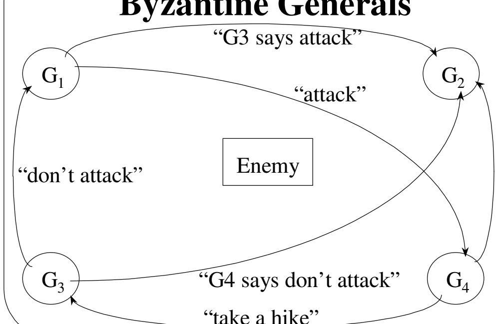
Byzantine Generals



Byzantine Generals



Byzantine Generals



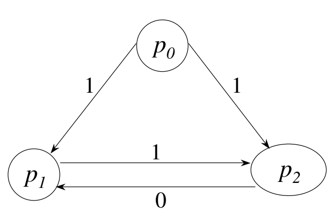
Upper bound on |Faulty processors|

- Too many faulty processors can prevent nonfaulty processors from agreement
- How many is too many?
- If fully connected, non-authenticated
  $m \leq \lfloor (n-1)/3 \rfloor$
- If authenticated, $m < n$ for agreement

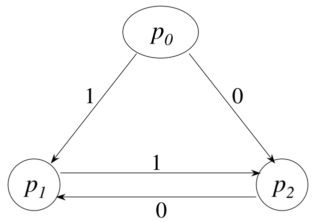
Impossibility result

- Agreement impossible if n=3, m=1
- Consider n=3, m=1, values are 0 and 1, processor p_0 initiates value
- two cases (indistinguishable!)
  - p_0 is not faulty
  - p_0 is faulty

p_0 is not faulty



p_0 is faulty



Lamport-Shostak-Pease Algorithm

- “Oral message” (non-authenticated) approach
- recursively defined(!)

Algorithm OM(0)

- source processor sends value to every processor
- each processor uses received value
  – default value is 0, if faulty source doesn’t send

Algorithm OM(m), m>0

- source processor sends value to every processor
- foreach i, let v_i := value process i gets from source

  Processor i initiates **Algorithm OM(m-1)** using v_i
  - processor i acts as new source
  - sends to n-2 (or n-3, etc.) other processors

Algorithm OM(m), *continued*

- foreach i and each $j(\neq i)$, let v_j := value process i received from processor j in Step 2 using Algorithm OM(m-1).

  Processor i uses the value *majority*(v_1, v_2, ..., v_{n-1})

Algorithm OM( ) comments

- message complexity is O(n^m)
- requires m +1 rounds
- Dolev solution is polynomial, but 2m + 3 rounds

Applications of Agreement

- clock synchronization
  - multiple initial values
- atomic commit

CS 239, Fall 2000 Lecture 7 Page 46

Summary

- Agreement protocols are needed to…
- Important parameters are…
- It is impossible to reach agreement when …
- Byz. agreement solutions are valuable …
- Typical applications of agreement are ...

CS 239, Fall 2000 Lecture 7 Page 47