EECS 591
Distributed Systems

Manos Kapritsos
Fall 2020
Vector clocks
Network Time Protocol

- The oldest distributed protocol still running on the Internet
- Hierarchical architecture
- Latency-tolerant, jitter-tolerant, fault-tolerant.. very tolerant!
Very tolerant. How?

- Tolerance to jitter, latency, faults: redundancy
- Each machine sends NTP requests to many other servers on the same or the previous stratum
- The synchronization protocol between two machines is similar to Cristian’s algorithm
- Each response defines an interval \([T_1, T_2]\)
- How to combine those intervals?
Marzullo’s algorithm

Given $M$ source intervals, find the largest interval that is contained in the largest number of source intervals.
The intuition

- Visit the endpoints left-to-right
- Count how many source intervals are active at each time
- Increase count at starting points, decrease at ending points
Preprocessing

For each source interval \([T_1, T_2]\), create 2 tuples of the form \(<\text{time}, \text{type}>\):

- \(<T_1,+1>\) (start of interval)
- \(<T_2,-1>\) (end of interval)

Sort all tuples according to time

Example:
Source intervals: \([8,12]\), \([11,13]\), \([14,15]\)
Tuples: \(<8,+1>\ <12,-1>\ <11,+1>\ <13,-1>\ <14,+1>\ <15,-1>\)
Sorted: \(<8,+1>\ <11,+1>\ <12,-1>\ <13,-1>\ <14,+1>\ <15,-1>\)
The algorithm

```python
best=0, count=0
for all tuples<time[i],type[i]> {
    count = count + type[i]
    if(count>best) {
        best=count
        beststart=time[i]
        bestend=time[i+1]
    }
}
return [beststart, bestend]
```

Notes:
- `count`: numbers of “active” intervals
- `best`: best numbers of “active” intervals we have seen
- `count=count+type[i]`: if it's a startpoint (type=+1), increase count, else decrease it
- `if(count>best)`: if this is the highest number of active intervals we have seen, let the best interval be `[time[i], time[i+1]]`
  - If the next point is a startpoint, it will replace this best interval
  - If the next point is an endpoint, it will end this best interval
The algorithm at work

Sorted: <8,+1> <11,+1> <12,-1> <13,-1> <14, +1> <15, -1>

Init: best=0, count=0

<8,+1> : count = count + (+1) = 1
  Is count>best? Yes
  best=1, beststart=8, bestend=11

<11,+1> : count = count + (+1) = 2
  Is count>best? Yes
  best=2, beststart=11, bestend=12

<12,-1> : count = count + (-1) = 1
  Is count>best? No

<13,-1> : count = count + (-1) = 0
  Is count>best? No

<14, +1> : count = count + (+1) = 1
  Is count>best? No

<15, -1> : count = count + (-1) = 0
  Is count>best? No

return [11,12]
NTP timestamps

How to represent time?
“Monday September 14th 2020, 15:20:00”?
“20200914152000EDT”?

NTP: 64-bit UTC timestamp

offset = #seconds since January 1, 1900

Wraps around every $2^{32}$ seconds = 136 years
First wrap-around: 2036

Solution: 128-bit timestamp. “Enough to provide unambiguous time representation until the universe goes dim”
Start forming groups for research project (3 students per group)

- Take a look at future content in part 1
- I have uploaded a list of papers we will read in part 2
- Start thinking about what you want to do

Homework assignment #1 will be released soon
Atomic Commit

-Do you take each other?
- I do.
- I do.
- I now pronounce you atomically committed.
1. Evil Lorenzo Speaks French
2. And was born in Corsica
3. Went to Dartmouth instead of Cornell
4. Rides a Ducati instead of a Moto Guzzi
5. Still listens opera, but doesn’t care for Puccini
5. Evil Lorenzo thinks that 2f+1 is good enough
Properties

Property: a predicate evaluated over a run of the program (also called a trace)

Example: “every message that is received was previously sent”

Not everything you may want to say about a program is a property: “the program sends an average of 50 messages in a run”
SAFETY PROPERTIES

- “nothing bad happens”
- only one process can be in the critical section at any time
- messages that are delivered are delivered in causal order
- Windows never crashes
- A safety property is “prefix closed”:
  - if it holds in a run, it holds in every prefix
LIVENESS PROPERTIES

- “something good eventually happens”
  - a process that wishes to enter the critical section eventually does so
  - some message is eventually delivered
  - Windows eventually boots
- Every run can be extended to satisfy a liveness property
  - if it doesn’t hold in a run, that doesn’t mean it may not hold eventually
SAFETY OR LIVENESS?

Whenever process A wants to enter the critical section, then all other processes get to enter at most once before A gets to enter

This program terminates

If this program eventually sends a message, it will be a well-formed HTTP request
A really cool theorem

Every property is a combination of a safety property and a liveness property

(Alpern & Schneider)
Atomic commit: the objective

Preserve data consistency for distributed transactions in the presence of failures
Model

- For each distributed transaction T:
  - one coordinator
  - a set of participants
- Coordinator knows participants; participants don’t necessarily know each other
- Each process has access to a Distributed Transaction Log (DT Log) on stable storage
**The setup**

- Each process $p_i$ has an input value $vote_i$
  
  $$vote_i \in \{Yes, No\}$$

- Each process $p_i$ has an output value $decision_i$
  
  $$decision_i \in \{Commit, Abort\}$$
AC SPECIFICATION

AC-1: All processes that reach a decision reach the same one
AC-2: A process cannot reverse its decision after it has reached one
AC-3: The **Commit** decision can only be reached if all processes vote **Yes**
AC-4: If there are no failures and all processes vote **Yes**, then the decision must be **Commit**
AC-5: If all failures are repaired and there are no more failures, then all processes will eventually decide
COMMENTS

AC-1: All processes that reach a decision reach the same one.
AC-2: A process cannot reverse its decision after it has reached one.
AC-3: The **Commit** decision can only be reached if all processes vote Yes.
AC-4: If there are no failures and all processes vote Yes, then the decision will be **Commit**.
AC-5: If all failures are repaired and there are no more failures, then all processes will eventually decide.

**AC-1:**
- AC-1 does not require all processes to reach a decision.
- It does not even require all correct processes to reach a decision.

**AC-4:**
- Avoids triviality.
- Allows **Abort** even if all processes have voted Yes.

**Note:**
- A process that does not vote Yes can unilaterally **Abort**.
UNCERTAINTY

• A process in *uncertain* if it has voted Yes but does not have sufficient information to Commit.

• While uncertain, a process cannot decide unilaterally.

• Uncertainty + communication failures = blocking.
INDEPENDENT RECOVERY

- Suppose process $p$ fails while running Atomic Commit

- If, during recovery, $p$ can reach a decision without communicating with other processes, we say that $p$ can independently recover

- total failure (= all processes fail)
  - independent recovery

  blocking
A FEW CHARACTER-BUILDING FACTS

Proposition 1
If communication failures or total failures are possible, then every AC protocol may cause processes to become blocked.

Proposition 2
No AC protocol can guarantee independent recovery of failed processes.