Atomic Broadcast
CASD Protocols
Fan Zhang
Department of Computer Science
Outline

• Introduction

• CASD Protocols
  • Basic CASD protocol
  • Second Protocol, Tolerant of timing failures
  • Third Protocol, Tolerant of authentication-detectable Byzantine failures

• Discuss on Δ
Intro.

- It’s hard to perform a reliable broadcast with real-time and other guarantees (total order, atomicity) within a distributed system
  - random failure
  - communication delay
- **Goal**: ensure the **correct** processes participating in a broadcast to attain **consistent** information.
  - Atomic broadcast
  - **CASD (Cristian, Aghili, Strong, Dolev) Protocols**
The CASD protocol suite

- Also known as the “Δ-T” protocols

- Developed by Cristian and others at IBM, was intended for use in the (ultimately, failed) FAA project

- Goal is to implement a timed atomic broadcast tolerant of Byzantine failures

Flaviu Cristian
1951-1999
What’s atomic broadcast

• Broadcast: make all of them know

• Guarantees
  • Real-Time: all correct processes deliver at the same time and within a finite delay
  • Failure-Atomicity: all or none
  • Order: messages are delivered in same order among all correct processes

• Can be used to implement synchronous replicated storage
Caveats

- Imperfect clock should be acceptable

- A process may not be able to detect that its own clock is incorrect.

- When a process is faulty, the guarantees no longer apply to it.
Failure Classification

- Omission failures: Omit one or more response. E.g. crash, link down, link occasionally loses messages, etc.
- Timing failures: respond too early/late
- Byzantine failure: corrupted messages,
  - Authentication-detectable subset
- Nested

\[ \text{Omission} \subset \text{Timing} \subset \text{Byzantine} \]
System Model

• $G = (E, V)$

• network diameter: $d$

• Primitives:
  • $\text{broadcast}(\sigma)$: init a atomic broadcast
  • $\text{send}(m)$ on $l$: send msg. $m$ on link $l$
  • $\text{receive}(m)$ from $i$: receive a msg. $m$ on link $i$
Assumptions

- Share accurate clock \(|C_p(t) - C_q(t)| < \epsilon\)
- \(n\) processes, at most \(k\) of them may be faulty
- Failures won’t cause the network to be disconnected
- Transmission and processing delay \(< \delta\)
- Number of lost packets is finite in a single run
Basic CASD

Tolerant of Omission
Basic CASD Protocol

- message = \{msg, t, pid\}
  - \textit{msg}: body of message
  - \textit{t}: \texttt{timestamp} (local to the sender)
  - \textit{pid}: identification of the sender process

- receive and relay manner
Basic CASD Protocol

• A process $p$ initiate a broadcast at $t$ by creating message $m=\{msg, t, pid\}$.

• $p$ forwards $m$ to all reachable processors

• Upon receipt of $m$ at another processor $p'$
  • discard $m$ if duplicated or out of feasible time range
  • reply $m$ over all links except incoming one

• All process hold $m$ until $t+\Delta$ and then deliver in the order of timestamp (break tie with pid)
p₀, p₁ fail. Messages are lost when echoed by p₂, p₃

Source: Slides for CS5412, Ken

- get the msg.
- deliver the msg.
Ideas

• Assume known limits on number of processes that fail during protocol, number of messages lost

• Using these and the temporal assumptions, deduce worst-case scenario

• Now now that if we wait long enough, all (or no) correct process will have the message

• Then schedule delivery using original time plus a delay computed from the worst-case assumptions
Δ “deliver deadline”

- broadcast begins at $t$, all processes deliver at $t+\Delta$

- $\Delta$ is an estimated amount, based on configuration

- How big $\Delta$ should be?
  - Big enough for all *correct* processes to receive $m$ at $t+\Delta$
  - Small enough for whole system to be efficient
Reasoning $\Delta$

- Ensure $\Delta$ is large enough even in worst case
- Msg. is created by faulty process and go through all faulty processes before reach the first correct process
- Faulty processes are very faulty — they just forward the msg. to one neighbor (if zero, the broadcast would fail) — $k\delta$
- Msg. diffuses among correct processes for longest possible time — $d\delta$

$$\Delta = k\delta + d\delta + \epsilon$$

faulty  diffuse  clock skew
Second Protocol

Tolerant of Timing Failure
Idea

• In first protocols, the “acceptance window” is fixed
  • accept if \( t < T + \Delta \) & no duplicate
  • A msg. might be “too late” for (early) correct processes yet “in time” for other (late) correct processes.
  • Must ensure all correct neighbors behave coherently
if p accept m(@tp), p’s neighbor q should accept m if p receive m(@tq)

- $-\epsilon < t_p - t_q < \delta + \epsilon$
  - $-\epsilon$: p is $\epsilon$ behind q, delay is zero
  - $\delta + \epsilon$: q is $\epsilon$ earlier than q, delay is $\delta$

- msg = (msg m, timestamp $T$, #hop $h$)

- Timeliness Acceptance: $T - h\epsilon < t < T + h(\delta + \epsilon)$

- Deliver deadline: $\Delta = k(\delta + \epsilon) + d\delta + \epsilon$
Third Protocol

Tolerating Authentication-Detectable Byzantine
Idea

- Use authentication to determine if the msg. is corrupted
  - Sender *signs* the msg.
  - Relayers *authenticate* the msg. then *co-sign* & relay it
    - *deliver* only if the msg. can be authenticated
    - *discard* corrupted messages
- Termination time is same as the second protocol
  - But msg. processing delay increases (~10 times)
Delta

Over relaxed! Keep waiting unnecessarily

Aggressive?
Reduce $\Delta$

- $\Delta$ is essentially a minimum latency for the protocol
  - $\Delta=3s$, in LAN used by CS Cornell
- How to squeeze $\Delta = k\delta + d\delta + \epsilon$
  - Assume (almost) fully connected $d = 1$
  - Assume processes and communication is reliable ($k$)
  - Clocks are closely synchronized
- $\Delta$ can be reduced to 100-150ms
Problems

• Reduce $\Delta$ will cause more process to be considered “faulty”
  • Not really faulty, but only in protocol’s eye
  • Guarantees no longer hold for such processes

• Thus, CASD is weak because the processes using it has no way to know whether or not it’s one of the correct ones.

• Probabilistically reliable
all processes look “incorrect” (red) from time to time
Problem

- Incorrect processes can still operate even without any guarantee
  - divergence of states occurs
- Incorrect processes are not excluded from the system
  - They can still initiate messages
  - Their inconsistency can spread
- No way for inconsistent system to coverage back to a consistent state.
Repair

• “silent” failures

• static membership with subsets who are faulty but with them notified in some way (So that the faulty processes will know about their failure)
  • Byzantine problem?

• managed membership (in which you can only treat a process as faulty if you are prepared to first exclude that process from the system completely)
  • Another global state?
Summary

• Atomic broadcast: real-time, total ordered and atomicity.

• Could be quite slow if we use conservative parameter settings
  • But with aggressive settings, either process could be deemed “faulty” by the protocol
  • If so, it might become inconsistent

• Merit: In reliable environment, the CASD protocols are guaranteed to satisfy their real-time properties.
Thanks!