Distributed Systems

CS425/ECE428

Instructor: Radhika Mittal
Two generals must agree on a time to attack the enemy base. They can communicate with each-other by sending messengers. But, a messenger may get killed by the enemy along the way. Thankfully, they have unlimited no. of messengers at their disposals. How can the two generals agree on a time to attack?
Logistics Related

- OHs information is up on website/course calendar.
- Sign-up forms for VM clusters is available on CampusWire.
  - Please fill it up by Thursday, Jan 26th, Thursday, 11:59pm.
- MP0 will be released on Wednesday.
- Lecture recordings on MediaSpace should be accessible to all registered students.
Today’s agenda

• System Model
  • Chapter 2.4 (except 2.4.3), parts of Chapter 2.3

• Failure Detection
  • Chapter 15.1
What is a distributed system?

Independent components that are connected by a network and communicate by passing messages to achieve a common goal, appearing as a single coherent system.
Relationship between processes

- Two main categories:
  - Client-server
  - Peer-to-peer
Key aspects of a distributed system

• Processes must communicate with one another to coordinate actions. Communication time is variable.

• Different processes (on different computers) have different clocks!

• Processes and communication channels may fail.
Key aspects of a *distributed* system

- Processes must communicate with one another to coordinate actions. Communication time is variable.

- Different processes (on different computers) have different clocks!

- Processes and communication channels may fail.
Two ways to model

• Synchronous distributed systems:
  • Known upper and lower bounds on time taken by each step in a process.
  • Known bounds on message passing delays.
  • Known bounds on clock drift rates.

• Asynchronous distributed systems:
  • No bounds on process execution speeds.
  • No bounds on message passing delays.
  • No bounds on clock drift rates.
Synchronous and Asynchronous

- Most real-world systems are asynchronous.
  - Bounds can be estimated, but hard to guarantee.
  - Assuming system is synchronous can still be useful.

- Possible to build a synchronous system.
Key aspects of a *distributed* system

• Processes must communicate with one another to coordinate actions. Communication time is variable.

• Different processes (on different computers) have different clocks!

• Processes and communication channels may fail.
Types of failure

- **Omission**: when a process or a channel fails to perform actions that it is supposed to do.
  - Process may crash.
How to detect a crashed process?

Periodic ping

Periodic heartbeats
How to detect a crashed process?

p sends pings to q every $T$ seconds.

$\Delta_1$ is the *timeout* value at p.

If $\Delta_1$ time elapsed after sending ping, and no ack, report q crashed.

If synchronous, $\Delta_1 = 2(\text{max network delay})$

If asynchronous, $\Delta_1 = k(\text{max observed round trip time})$
How to detect a crashed process?

q sends heartbeats to p every $T$ seconds. 

$(T + \Delta_2)$ is the timeout value at p.

If $(T + \Delta_2)$ time elapsed since last heartbeat, report q crashed.

If synchronous, $\Delta_2 = \text{max network delay} - \text{min network delay}$
How to detect a crashed process?

q sends heartbeats to p every $T$ seconds. 

$(T + \Delta_2)$ is the timeout value at p.

If $(T + \Delta_2)$ time elapsed since last heartbeat, report q crashed.
How to detect a crashed process?

q sends heartbeats to p every $T$ seconds.

$$(T + \Delta_2)$$ is the timeout value at p.

If $(T + \Delta_2)$ time elapsed since last heartbeat, report q crashed.

If synchronous, $\Delta_2 = \text{max network delay} - \text{min network delay}$

If asynchronous, $\Delta_2 = k(\text{observed delay})$
Correctness of failure detection

• Completeness
  • Every failed process is eventually detected.

• Accuracy
  • Every detected failure corresponds to a crashed process (no mistakes).
Correctness of failure detection

• Characterized by completeness and accuracy.

• Synchronous system
  • Failure detection via ping-ack and heartbeat is both complete and accurate.

• Asynchronous system
  • Our strategy for ping-ack and heartbeat is
  • Impossible to achieve both completeness and accuracy.
  • Can we have an accurate but incomplete algorithm?
    • Never report failure.
Metrics for failure detection

- Worst case failure detection time
  - Ping-ack: \( T + \Delta_1 - \Delta \) (where \( \Delta \) is time taken for last ping from p to reach q)
  - Heartbeat: \( \Delta + T + \Delta_2 \) (where \( \Delta \) is time taken for last message from q to reach p)

Try deriving these before next class!
Metrics for failure detection

• Worst case failure detection time
  • After a process crashes, how long does it take for the other process to detect the crash in the worst case?
Metrics for failure detection

• Worst case failure detection time
  • Ping-ack: $T + \Delta_1 - \Delta$ where $\Delta$ is time taken for the last ping from $p$ to reach $q$ before $q$ crashed. $T$ is the time period for pings, and $\Delta_1$ is timeout value.

Try deriving this!
Metrics for failure detection

- Worst case failure detection time
  - Ping-ack: $T + \Delta_1 - \Delta$ where $\Delta$ is time taken for the last ping from $p$ to reach $q$ before $q$ crashed. $T$ is the time period for pings, and $\Delta_1$ is timeout value.

Worst case failure detection time:
$$t + T + \Delta_1 - (t + \Delta) = T + \Delta_1 - \Delta$$

Q: What is worst case value of $\Delta$ for a synchronous system?
A: min network delay
Metrics for failure detection

• Worst case failure detection time
  • Heartbeat: $T + \Delta_2 + \Delta$ where $\Delta$ is time taken for last heartbeat from $q$ to reach $p$
    $T$ is the time period for heartbeats, and $T + \Delta_2$ is the timeout.

Try deriving this!
Metrics for failure detection

• Worst case failure detection time
  • Heartbeat: $T + \Delta_2 + \Delta$ where $\Delta$ is time taken for last heartbeat from q to reach p
    $T$ is the time period for heartbeats, and $T + \Delta_2$ is the timeout.

Worst case failure detection time:
$$(t + \Delta) + (T + \Delta_2) - t = T + \Delta_2 + \Delta$$

Q: What is worst case value of $\Delta$ in a synchronous system?
A: max network delay
Metrics for failure detection

• Worst case failure detection time
  • Ping-ack: $T + \Delta_1 - \Delta$ (where $\Delta$ is time taken for last ping from p to reach q before crash)
  • Heartbeat: $T + \Delta_2 + \Delta$ (where $\Delta$ is time taken for last heartbeat from q to reach p)
Metrics for failure detection

• Worst case failure detection time
  • Ping-ack: $T + \Delta_1 - \Delta$ (where $\Delta$ is time taken for previous ping from p to reach q)
  • Heartbeat: $T + \Delta_2 + \Delta$ (where $\Delta$ is time taken for last heartbeat from q to reach p)

• Bandwidth usage:
  • Ping-ack: 2 messages every $T$ units
  • Heartbeat: 1 message every $T$ units.
Metrics for failure detection

- Worst case failure detection time
  - Ping-ack: $T + \Delta_1 - \Delta$ (where $\Delta$ is time taken for previous ping from $p$ to reach $q$)
  - Heartbeat: $T + \Delta_2 + \Delta$ (where $\Delta$ is time taken for last heartbeat from $q$ to reach $p$)

- Bandwidth usage:
  - Ping-ack: 2 messages every $T$ units
  - Heartbeat: 1 message every $T$ units.

Effect of decreasing $T$?
Metrics for failure detection

• Worst case failure detection time
  • Ping-ack: $T + \Delta_1 - \Delta$ (where $\Delta$ is time taken for previous ping from p to reach q)
  • Heartbeat: $T + \Delta_2 + \Delta$ (where $\Delta$ is time taken for last heartbeat from q to reach p)

• Bandwidth usage:
  • Ping-ack: 2 messages every $T$ units
  • Heartbeat: 1 message every $T$ units.

Decreasing $T$ decreases failure detection time, but increases bandwidth usage.
Metrics for failure detection

• Worst case failure detection time
  • Ping-ack: $T + \Delta_1 - \Delta$ (where $\Delta$ is time taken for previous ping from p to reach q)
  • Heartbeat: $T + \Delta_2 + \Delta$ (where $\Delta$ is time taken for last heartbeat from q to reach p)

• Bandwidth usage:
  • Ping-ack: 2 messages every $T$ units
  • Heartbeat: 1 message every $T$ units.

Effect of increasing $\Delta_1$ or $\Delta_2$?
Metrics for failure detection

• Worst case failure detection time
  • Ping-ack: $T + \Delta_1 - \Delta$ (where $\Delta$ is time taken for previous ping from p to reach q)
  • Heartbeat: $T + \Delta_2 + \Delta$ (where $\Delta$ is time taken for last heartbeat from q to reach p)

• Bandwidth usage:
  • Ping-ack: 2 messages every $T$ units
  • Heartbeat: 1 message every $T$ units.

Increasing $\Delta_1$ or $\Delta_2$ increases accuracy but also increases failure detection time.
Types of failure

- **Omission**: when a process or a channel fails to perform actions that it is supposed to do.
  - Process may **crash**.
  - **Fail-stop**: if other processes can certainly detect the crash.
  - **Communication omission**: a message sent by process was not received by another.
Communication Omission

- Channel Omission: omitted by channel
- Send omission: process completes ‘send’ operation, but message does not reach its outgoing message buffer.
- Receive omission: message reaches the incoming message buffer, but not received by the process.
Two Generals Problem

When to attack?
Two Generals Problem

At dawn.

Has my message reached?
Two Generals Problem

Has my confirmation reached?

confirm
Two Generals Problem

Has my ack reached?

ack “confirm”.
Two Generals Problem

At dawn.

Has my message reached?

Keep sending the message until confirmation arrives.
Two Generals Problem

Assume confirmation has reached in the absence of a repeated message.

Still no guarantees! But may be good enough in practice.
Types of failure

- **Omission**: when a process or a channel fails to perform actions that it is supposed to do.
  - Process may **crash**.
  - **Fail-stop**: if other processes can detect that the process has crashed.
  - **Communication omission**: a message sent by process was not received by another.

Message drops (or omissions) can be mitigated by network protocols.
Types of failure

- **Omission:** when a process or a channel fails to perform actions that it is supposed to do, e.g. process crash and message drops.

- **Arbitrary (Byzantine) Failures:** any type of error, e.g. a process executing incorrectly, sending a wrong message, etc.

- **Timing Failures:** Timing guarantees are not met.
  - Applicable only in synchronous systems.
How to detect a crashed process?

Periodic ping

Periodic heartbeats
Extending heartbeats

• Looked at detecting failure between two processes.

• How do we extend to a system with multiple processes?
Centralized heartbeating

**Downside:**
What if $p_i$ fails?

$p_i, \text{Heartbeat Seq}++$
Ring heartbeating

$p_i$, Heartbeat Seq++

Downside:
What if multiple processes fail?
Ring repair overhead
All-to-all heartbeats

Everyone can keep track of everyone.

**Downside:**
Extending heartbeats

• Looked at detecting failure between two processes.

• How do we extend to a system with multiple processes?
  • Centralized heartbeating: not complete.
  • Ring heartbeating: not entirely complete, ring repair overhead.
  • All-to-all: complete, but more bandwidth usage.
Failures

• Three types
  • omission, arbitrary, timing.

• Failure detection (detecting a crashed process):
  • Send periodic ping-acks or heartbeats.
  • Report crash if no response until a timeout.
  • Timeout can be precisely computed for synchronous systems and estimated for asynchronous.
  • Metrics: completeness, accuracy, failure detection time, bandwidth.
• Failure detection for a system with multiple processes:
  • Centralized, ring, all-to-all
  • Trade-off between completeness and bandwidth usage.
Summary

• Sources of uncertainty
  • Communication time, clock drift rates

• Synchronous vs asynchronous models.

• Types of failures: omission, arbitrary, timing

• Detecting failed a process.