A CHALLENGE

• You’ve been put in charge of a datacenter, and your manager has told you, “Oh no! We don’t have any failures in our datacenter!”

• Do you believe him/her?

• What would be your first responsibility?
  • Build a failure detector
  • What are some things that could go wrong if you didn’t do this?
Failures are the Norm

… not the exception, in datacenters.

Say, the rate of failure of one machine (OS/disk/motherboard/network, etc.) is once every 10 years (120 months) on average.

When you have 120 servers in the DC, the mean time to failure (MTTF) of the next machine is 1 month.

When you have 12,000 servers in the DC, the MTTF is about once every 7.2 hours!

Soft crashes and failures are even more frequent!
You have a few options

1. Hire 1000 people, each to monitor one machine in the datacenter and report to you when it fails.
2. Write a failure detector program (distributed) that automatically detects failures and reports to your workstation.
Target Settings

• Process ‘group’-based systems
  – Clouds/Datacenters
  – Replicated servers
  – Distributed databases

• Crash-stop/Fail-stop process failures
GROUP MEMBERSHIP SERVICE

Application Queries
e.g., gossip, overlays,
DHT’s, etc.

Membership List

Unreliable Communication

Membership Protocol

Application Process \( p_i \)
Two sub-protocols

Application Process \( pi \)

Group

Membership List

- **Complete list all the time** (Strongly consistent)
  - Virtual synchrony
- **Almost-Complete list** (Weakly consistent)
  - Gossip-style, SWIM, ...
- **Or Partial-random list** (other systems)
  - SCAMP, T-MAN, Cyclon, ...

Focus of this series of lecture
Large Group: Scalability A

Goal

this is us (pi)

1000’s of processes

Unreliable Communication Network

Process Group “Members”
**GROUP MEMBERSHIP PROTOCOL**

I. \( p_j \) crashed

II. Failure Detector
Some process finds out quickly

III. Dissemination

Unreliable Communication Network

Crash-stop Failures only
How do you design a group membership protocol?
I. $pj$ crashes

- Nothing we can do about it!
- A frequent occurrence
- Common case rather than exception
- Frequency goes up linearly with size of datacenter
II. DISTRIBUTED FAILURE DETECTORS: DESIRABLE PROPERTIES

- **Completeness** = each failure is detected
- **Accuracy** = there is no mistaken detection
- **Speed**
  - Time to first detection of a failure
- **Scale**
  - Equal Load on each member
  - Network Message Load
DISTRIBUTED FAILURE DETECTORS: PROPERTIES

- Completeness
- Accuracy
- Speed
  - Time to first detection of a failure
- Scale
  - Equal Load on each member
  - Network Message Load

Impossible together in lossy networks [Chandra and Toueg]
If possible, then can solve consensus!
What Real Failure Detectors Prefer

- Completeness
- Accuracy
- Speed
  - Time to first detection of a failure
- Scale
  - Equal Load on each member
  - Network Message Load

Guaranteed

Partial/Probabilistic guarantee
What Real Failure Detectors Prefer

• Completeness
• Accuracy
• Speed
  – Time to first detection of a failure
• Scale
  – Equal Load on each member
  – Network Message Load

Guaranteed
Partial/Probabilistic guarantee
Time until *some* process detects the failure
What Real Failure Detectors Prefer

• Completeness
• Accuracy
• Speed
  – Time to first detection of a failure
• Scale
  – Equal Load on each member
  – Network Message Load

Guaranteed
Partial/Probabilistic guarantee

Time until some process detects the failure
No bottlenecks/single failure point
**Failure Detector Properties**

- **Completeness**
- **Accuracy**
- **Speed**
  - Time to first detection of a failure
- **Scale**
  - Equal Load on each member
  - Network Message Load

In spite of arbitrary simultaneous process failures
Centralized Heartbeating

- Heartbeats sent periodically
- If heartbeat not received from $pi$ within timeout, mark $pi$ as failed

$pi$, Heartbeat Seq. $l++$

Hotspot
Ring Heartbeating

$pi$, Heartbeat Seq. $l++$

$pj$

☹️ Unpredictable on simultaneous multiple failures
All-to-All Heartbeating

$pi$, Heartbeat Seq. $l++$

$pi$  \hspace{1cm} \text{Equal load per member} \hspace{1cm} pj

\ldots
Next

• How do we increase the robustness of all-to-all heartbeating?
GOSSIP-STYLE HEARTBEATING

Array of Heartbeat Seq. $l$ for member subset

$pi$ ☺ Good accuracy properties
**GOSSIP-STYLE FAILURE DETECTION**

**Protocol:**
- Nodes periodically gossip their membership list: pick random nodes, send it list
- On receipt, it is *merged* with local membership list
- When an entry times out, member is marked as failed

Current time: 70 at node 2
(asynchronous clocks)
Gossip-Style Failure Detection

• If the heartbeat has not increased for more than $T_{\text{fail}}$ seconds, the member is considered failed
• And after $T_{\text{cleanup}}$ seconds, it will delete the member from the list
• Why two different timeouts?
Gossip-Style Failure Detection

What if an entry pointing to a failed node is deleted right after $T_{\text{fail}} (=24)$ seconds?

Current time : 75 at node 2
Multi-level Gossiping

- Network topology is hierarchical
- Random gossip target selection

=> core routers face $O(N)$ load

(Why?)

- Fix: In subnet $i$, which contains $n_i$ nodes, pick gossip target in your subnet with probability $(1-1/n_i)$
- Router load=$O(1)$
- Dissemination time=$O(\log(N))$
- What about latency for multi-level topologies?

[Gupta et al, TPDS 06]
Analysis/Discussion

• What happens if gossip period $T_{\text{gossip}}$ is decreased?

• A single heartbeat takes $O(\log(N))$ time to propagate. So: $N$ heartbeats take:
  – $O(\log(N))$ time to propagate, if bandwidth allowed per node is allowed to be $O(N)$
  – $O(N.\log(N))$ time to propagate, if bandwidth allowed per node is only $O(1)$
  – What about $O(k)$ bandwidth?

• What happens to $P_{\text{mistake}}$ (false positive rate) as $T_{\text{fail}}, T_{\text{cleanup}}$ is increased?

• Tradeoff: False positive rate vs. detection time vs. bandwidth
Next

• So, is this the best we can do? What is the best we can do?
Failure Detector Properties ...

• Completeness
• Accuracy
• Speed
  – Time to first detection of a failure
• Scale
  – Equal Load on each member
  – Network Message Load
...Are application-defined Requirements

- Completeness
- Accuracy
- Speed
  - Time to first detection of a failure
- Scale
  - Equal Load on each member
  - Network Message Load

Guarantee always

Probability $P_M(T)$

$T$ time units
Requirements

- Completeness
- Accuracy
- Speed
  - Time to first detection of a failure
- Scale
  - Equal Load on each member
  - Network Message Load

Guarantee always
Probability \( PM(T) \)
\( T \) time units

\( N*L: \) Compare this across protocols
All-to-All Heartbeating

\( pi, \text{Heartbeat Seq. } l++ \)

Every \( T \) units

\[ L = \frac{N}{T} \]
Gossip-style Heartbeating

Array of Heartbeat Seq. $l$ for member subset

Every $tg$ units = gossip period, send $O(N)$ gossip message

$T = \log N \times tg$

$L = \frac{N}{tg} = \frac{N \times \log N}{T}$
What's the Best/Optimal we can do?

- **Worst case** load $L^*$ per member in the group (messages per second)
  - as a function of $T$, $PM(T)$, $N$
  - Independent Message Loss probability $p_{ml}$

\[
L^* = \frac{\log(PM(T)) \cdot 1}{\log(p_{ml}) \cdot T}
\]
**Heartbeating**

- Optimal L is independent of N (!)
- All-to-all and gossip-based: sub-optimal
  - L=O(N/T)
  - try to achieve simultaneous detection at *all* processes
  - fail to distinguish *Failure Detection* and *Dissemination* components

**Key:**
- Separate the two components
- Use a non heartbeat-based Failure Detection Component
• Is there a better failure detector?
SWIM Failure Detector Protocol

Protocol period
= T’ time units

• random pj
  ping

• random K
  ping-req

K random processes

pi

pj

X

X

ack

ack

ping

ack
SWIM versus Heartbeating

First Detection Time
- Constant

Process Load
- Constant
- O(N)

For Fixed:
- False Positive Rate
- Message Loss Rate
## SWIM Failure Detector

<table>
<thead>
<tr>
<th>Parameter</th>
<th>SWIM</th>
</tr>
</thead>
</table>
| **First Detection Time**   | • Expected \( \left\lceil \frac{e}{e - 1} \right\rceil \) periods  
                           | • Constant (independent of group size)                                                                                       |
| **Process Load**           | • **Constant** per period  
                           | • \(< 8 \text{ L}^* \) for 15\% loss                                                                                      |
| **False Positive Rate**    | • Tunable (via K)  
                           | • **Falls exponentially** as load is scaled                                                                               |
| **Completeness**           | • Deterministic time-bounded  
                           | • Within \( O(\log(N)) \) periods w.h.p.                                                                              |
**Accuracy, Load**

- $PM(T)$ is exponential in $-K$. Also depends on $pml$ (and $pf$)
  - See paper

\[
\frac{L}{L^*} < 28 \quad \text{and} \quad \frac{E[L]}{L^*} < 8
\]

- for up to 15 % loss rates
Detection Time

- Prob. of being pinged in $T' = 1 - (1 - \frac{1}{N})^{N-1} = 1 - e^{-1}$

- $E[T] = T' \cdot \frac{e}{e - 1}$

- Completeness: Any alive member detects failure
  - Eventually
  - By using a trick: within worst case $O(N)$ protocol periods
TIME-BOUNDED COMPLETENESS

- Key: select each membership element once as a ping target in a traversal
  - Round-robin pinging
  - Random permutation of list after each traversal
- Each failure is detected in worst case $2N-1$ (local) protocol periods
- Preserves FD properties

This slide not covered (not in syllabus)
Next

• How do failure detectors fit into the big picture of a group membership protocol?
• What are the missing blocks?
GROUP MEMBERSHIP PROTOCOL

I pj crashed

II Failure Detector

Some process finds out quickly

III Dissemination

Unreliable Communication Network

Crash-stop Failures only
**DISSEMINATION OPTIONS**

• Multicast (Hardware / IP)
  – unreliable
  – multiple simultaneous multicasts

• Point-to-point (TCP / UDP)
  – expensive

• Zero extra messages: Piggyback on Failure Detector messages
  – Infection-style Dissemination
Infection-style Dissemination

Protocol period
= T time units

• random $pj$
  ping

• random $K$
  ping-req

Piggybacked membership information

K random processes
**Infection-style Dissemination**

- **Epidemic/Gossip style dissemination**
  - After $\lambda \log(N)$ protocol periods, $N^{-2\lambda-2}$ processes would not have heard about an update

- **Maintain a buffer of recently joined/evicted processes**
  - Piggyback from this buffer
  - Prefer recent updates

- **Buffer elements are garbage collected after a while**
  - After $\lambda \log(N)$ protocol periods, i.e., once they’ve propagated through the system; this defines weak consistency
Suspicion Mechanism

- False detections, due to
  - Perturbed processes
  - Packet losses, e.g., from congestion
- Indirect pinging may not solve the problem
- Key: suspect a process before declaring it as failed in the group
**Suspicion Mechanism**

- **Alive**
  - **FD::pi ping failed**
  - **Dissmn::(Suspect p_j)**

- **Suspected**
  - **Dissmn** (Suspect p_j)

- **Failed**
  - **Time out**
  - **FD::pi ping success**
  - **Dissmn::(Alive p_j)**

- **Dissmn** (Alive p_j)

- **Dissmn** (Failed p_j)
SUSPICION MECHANISM

• Distinguish multiple suspicions of a process
  – Per-process incarnation number
  – Inc # for pi can be incremented only by pi
    • e.g., when it receives a (Suspect, pi) message
  – Somewhat similar to DSDV
• Higher inc# notifications over-ride lower inc#’s
• Within an inc#: (Suspect inc #) > (Alive, inc #)
• (Failed, inc #) overrides everything else
Wrap Up

- Failures the norm, not the exception in datacenters
- Every distributed system uses a failure detector
- Many distributed systems use a membership service

- Ring failure detection underlies
  - IBM SP2 and many other similar clusters/machines

- Gossip-style failure detection underlies
  - Amazon EC2/S3 (rumored!)
Important Announcement

• Next week Tue and Thu: We’ll have a flipped classroom! (like Khan Academy)
• Homework before Next week
  • Please see video lectures for two topics
    • Timestamps and Ordering before Tue
    • Global Snapshots before Thu
• When you come to class on Sep 9\textsuperscript{th} (Tue) and Sep 11\textsuperscript{th} (Thu) the TAs will be helping you do exercises in class (not HW problems, but other exercise problems we will give you)
• We will not replay videos in class, i.e., there will be no lecturing.
• If you don’t see the videos before class, you will flounder in class. So make sure you see them before class.
• Exercises may count for grades.
• Please bring a pen/pencil and paper to both classes.