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!
To build a Failure Detector

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

• Fail-stop (crash) process failures
**GROUP MEMBERSHIP SERVICE**

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

Membership List  

Application Process $pi$

Membership Protocol  

Unreliable Communication
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,…

Dissemination
Failure Detector
Unreliable Communication

Focus of this series of lecture
LARGE GROUP: SCALABILITY

**Goal**

- 1000’s of processes
- Unreliable Communication Network

*this is us (pi)*

Process Group “Members”
GROUP MEMBERSHIP PROTOCOL

I pj crashed

II Failure Detector
  Some process finds out quickly

III Dissemination

Unreliable Communication Network

Fail-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! (but consensus is known to be unsolvable in asynchronous systems)
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
Ring Heartbeating

\( pi, \) Heartbeat Seq. \( l++ \)

\( pi \)

\( pj \)

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

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

Equal load per member
Single hb loss $\rightarrow$ false detection
Next

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

Array of Heartbeat Seq. $I$ for member subset

$p_i$

😊 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 a further $T_{\text{cleanup}}$ seconds, it will delete the member from the list.
- Why an additional timeout? Why not delete right away?
**Gossip-Style Failure Detection**

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10120</td>
<td>66</td>
</tr>
<tr>
<td>2</td>
<td>10103</td>
<td>62</td>
</tr>
<tr>
<td>3</td>
<td>10098</td>
<td>55</td>
</tr>
<tr>
<td>4</td>
<td>10111</td>
<td>65</td>
</tr>
</tbody>
</table>

Current time : 75 at node 2
• What happens if gossip period $T_{\text{gossip}}$ is decreased?
• Well-known result: a gossip takes $O(\log(N))$ time to propagate.
• So: Given sufficient bandwidth, 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
• 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 $PM(T)$

$T$ time units
...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 $PM(T)$

$T$ time units

$N*L$: Compare this across protocols
All-to-All Heartbeating

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

Every $T$ units

$L = N / T$
Gossip-style Heartbeating

Array of Heartbeat Seq. $l$ for member subset

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

$T = \log N \times tg$

$L = \frac{N}{tg} = N \times \log N / T$
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 $p_{ml}$

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

Can we reach this bound?

Key:
- Separate the two components
- Use a non heartbeat-based Failure Detection Component
Next

- Is there a better failure detector?
SWIM Failure Detector Protocol

Protocol period = T’ time units

\[ \text{Protocol period} = T' \text{ time units} \]
**Detection Time**

- Prob. of being pinged in $T' = 1 - \left(1 - \frac{1}{N}\right)^{N-1} = 1 - e^{-1}$
- $E[T] = T'. \frac{e}{e - 1}$
- Completeness: *Any* alive member detects failure
  - Eventually
  - By using a trick: within worst case $O(N)$ protocol periods
Accuracy, Load

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

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

for up to 15% loss rates
## SWIM Failure Detector

<table>
<thead>
<tr>
<th>Parameter</th>
<th>SWIM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First Detection Time</strong></td>
<td>- Expected $\left[ \frac{e}{e - 1} \right]$ periods</td>
</tr>
<tr>
<td></td>
<td>- Constant (independent of group size)</td>
</tr>
<tr>
<td><strong>Process Load</strong></td>
<td>- <strong>Constant</strong> per period</td>
</tr>
<tr>
<td></td>
<td>- $&lt; 8 L^*$ for 15% loss</td>
</tr>
<tr>
<td><strong>False Positive Rate</strong></td>
<td>- Tunable (via $K$)</td>
</tr>
<tr>
<td></td>
<td>- <strong>Falls exponentially</strong> as load is scaled</td>
</tr>
<tr>
<td><strong>Completeness</strong></td>
<td>- Deterministic time-bounded</td>
</tr>
<tr>
<td></td>
<td>- Within $O(\log(N))$ periods w.h.p.</td>
</tr>
</tbody>
</table>
**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
SWIM versus Heartbeating

First Detection Time
- Constant
- \( O(N) \)

For Fixed:
- False Positive Rate
- Message Loss Rate

SWIM
- Constant
- \( O(N) \)

Heartbeating
- \( O(N) \)
- Process Load

Heartbeating
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
   - Some process finds out quickly

II. Failure Detector
   - How?

III. Dissemination

Unreliable Communication Network

Fail-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**

FD :: $pi$ ping failed
Disssmn :: (Suspect $pj$)

FD :: $pi$ ping success
Disssmn :: (Alive $pj$)

Disssmn (Alive $pj$)

Disssmn (Suspect $pj$)

Time out

Disssmn (Failed $pj$)

Disssmn (Failed $pj$)
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 (routing protocol in ad-hoc nets)
• 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!)
Announcements

• MP1 – Demo signup sheet available on Piazza
  – Demo details up soon

• Check Piazza often! It’s where all the announcements are at!