Communication between Nodes

Epidemic Diffusion and Gossiping

Smruti R. Sarangi

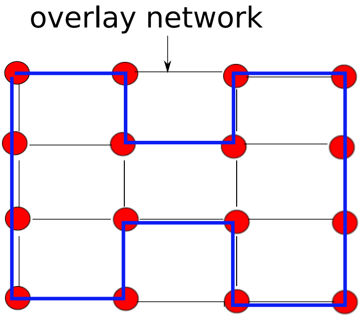
Department of Computer Science
Indian Institute of Technology
New Delhi, India

Outline

Overlay Network

Overlay Network

It is an application level network that is independent of the underlying network topology.



- Most common overlay networks are a ring and star.

Ring and Star

- A star is a centralized configuration.
  - The central node is typically a server.
  - The rest of the nodes are clients.
- A ring is the basis of a structure called a DHT (distributed hash table)
  - We will study about ring topologies in third generation peer to peer networks.
- Let us first focus on unstructured overlay networks.
  - There is no fixed global topology.
  - A node typically only knows a subset of other nodes.

Strategies for Unstructured Networks

The Problem

Multicast a message to a group of nodes.

- Send the message to all the neighbors.
- Ask the neighbors to further forward the message to their neighbors.
- Need a method to solve the exponential flooding of messages.
- Need mathematical techniques for analysis.

Epidemic Algorithms

Paper

Epidemic Algorithms for Replicated Database Maintenance by Alan Demers. Dan, Greene, Carl Hauser, Wes Irish, John. Larson. Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry, PODC 1987

- Problem: Propagate updates to a large set of databases in Xerox's corporate intranet.
- Updates are injected at one site and propagated to the rest of the sites.

General Mechanisms

Direct Mail

Each update is sent from one site to all sites.

Anti-Entropy

Choose a site at random and synchronize the contents of the databases by exchanging contents.

Rumor Mongering

A site distributes updates to other sites. When a site sees that most of its neighbors have the update, the rumor ceases to be hot . Gradually it dies away.

Tradeoffs

infective Already received the update, and willing to propagate.

susceptible Has not received the update.

removed Not participating in propagating updates.

- Anti-Entropy
  - Takes longer to propagate updates as compared to direct mail.
  - Does not have a built in termination mechanism.
  - Simple epidemic
- Rumor mongering
  - There is a chance that updates might not reach a node.
  - Has a built in termination mechanism.
  - Complex epidemic

Outline

- A network contains S sites.
- Database copy, K, at $s \in S$ is

$$s.valueOf : K \rightarrow (v : V \times t : T)$$

  - v is the value
  - t is the timestamp

Algorithm 1: Anti-entropy algorithm

```
ResolveDifference–push(s,s') {
if s.valueOf.t > s'.valueOf.t then
    s'.valueOf ← s.valueOf
end
}

ResolveDifference–pull(s,s') {
if s.valueOf.t < s'.valueOf.t then
    s.valueOf ← s'.valueOf
end
}

ResolveDifference–push,pull(s,s') {
ResolveDifference–pull(s,s')
ResolveDifference–push(s,s')
}
```

Analysis

- Anti-entropy distributes updates in $O(log(n))$ time (see results from epidemic theory).
- Pull-based algorithms
  - Let p_i be the probability of a site remaining susceptible after the i^{th} cycle.
  - $$p_{i+1} = p_i^2$$
- Push-based algorithms
  - Expected number of infective nodes: $n(1 - p_i)$
  - Probability of not contacting node X: $1 - 1/n$
  - $$p_{i+1} = p_i\,(1 - 1/n)^{n(1-p_i)}$$
  - Now, $(1 - 1/x)^x$ tends to $1/e$ as $x \to \infty$
  - Thus, for large n, and small p_i:

$$p_{i+1} = p_i e^{-1}$$

Analysis

- Anti-entropy distributes updates in $O(log(n))$ time (see results from epidemic theory).
- Pull-based algorithms
  - Let p_i be the probability of a site remaining susceptible after the i^{th} cycle.
  - $$p_{i+1} = p_i^2$$
- Push-based algorithms
  - Expected number of infective nodes: $n(1 - p_i)$
  - Probability of not contacting node X: $1 - 1/n$
  - $$p_{i+1} = p_i\,(1 - 1/n)^{n(1-p_i)}$$
  - Now, $(1 - 1/x)^x$ tends to $1/e$ as $x \to \infty$
  - Thus, for large n, and small p_i:

$$p_{i+1} = p_i e^{-1}$$

Pull based, and push-pull based methods are better at the end.

Discussion

- Push based methods are better at the beginning
- Towards the end pull based methods are better
- Instead of comparing entire database contents, we can do better:
  - First compare recent entries (Less than τ seconds old)
  - If they match, then nothing needs to be done.
  - If they do not match, then update recent entries, and compare check sums of the rest of the database.
  - If the checksums do not match, then synch. databases.

Outline

1. Communication between Nodes
2. Epidemic Protocols
   - Anti-Entropy
   - Rumor Mongering
   - Deleting Nodes and Spatial Distribution
3. Gossip Based Protocols
   - Protocol
   - Mathematical Analysis
   - Catastrophe Recovery

Terminology

- s $\rightarrow$ fraction of nodes that are susceptible
- i $\rightarrow$ fraction of nodes that are infective
- r $\rightarrow$ fraction of nodes that are removed

Governing Equations

- Sum of nodes is 1

$$s + i + r = 1$$

Equations

- Rate of decrease of susceptible nodes

$$\frac{ds}{dt} = -si$$

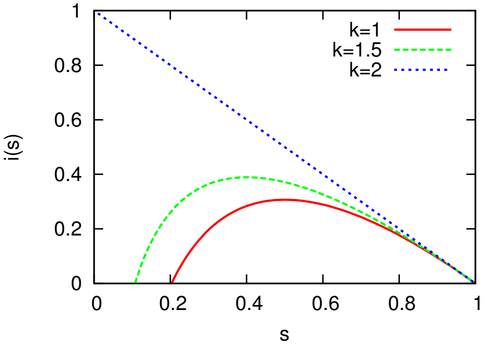
- Nodes lose interest in propagating rumors by a probabilistic factor of 1/k.

$$\frac{di}{dt} = si - \frac{1}{k}(1-s)i$$

Solution

$$i(s) = \frac{k+1}{k}(1-s) + \frac{1}{k}log(s)$$

i(s) vs s



Implication

There are still some susceptible nodes. They decrease exponentially with k.

Implications of the Solution

Let us now see what percentage of nodes are still susceptible, when no other node is infective $i(s) = 0$.

$$s = e^{-\frac{k+1}{1-s}}$$

- Exponentially decreases with k.
- Some nodes still remain susceptible.
- This value is called the residue

Fundamental Relationships

residue Sites that are still susceptible after the end of the epidemic.

traffic Average number of messages sent per site.

- m updates per site, n sites, total nm updates
- Chances that a site will miss all the updates:

$$s = (1 - 1/n)^{nm} = e^{-m}$$

Exponential relationship between traffic and residue

Backup with Anti-entropy

- Rumor mongering can miss sites.
- After a certain time, we can run an anti-entropy protocol.
- If two sites discover a missing update, then they could start a hot rumor.
- Xerox Clearinghouse did some redistribution through direct mail also.

Outline

1. Communication between Nodes
2. Epidemic Protocols
   - Anti-Entropy
   - Rumor Mongering
   - Deleting Nodes and Spatial Distribution
3. Gossip Based Protocols
   - Protocol
   - Mathematical Analysis
   - Catastrophe Recovery

Death Certificate

We can treat the deletion of an item as an update , and issue a death certificate. The death certificates can be propagated through rumors or anti-entropy. When a death certificate meets a later update, the update gets cancelled.

- When do we discard death certificates?
- Need to define a time threshold.
- If a death certificate is older than the time it takes to propagate the update to all sites, we can delete it.
- We can still maintain some copies at a selected number of retention sites.

Anti-entropy/Rumor with Dormant Death Certificates

Dormant Death Certificate

Keep a death certificate only at a few nodes. It it collides with an update, then activate the death certificate and propagate it.

- What if a dormant death certificate meets an obsolete update.
- Reactivate the dormant death certificate and distribute it.
- It is possible that a legitimate update can be cancelled if we don't set its time properly.
- This can be solved by using version numbers for updates. A death certificate will have two timestamps: original, and activation.
- The original stamp will be used to cancel updates, and the activation timestamp will be used to ultimately get rid of the death certificate.

Spatial Distributions

Now consider the fact that it takes time to send a message depending on the distance to the destination.

Known Results

- If a node can contact only neighbors, it takes $O(n)$ time to spread an update using anti-entropy.
- If a node can contact any other node, it takes $O(log(n))$ time.

General Result

Let the probability of connecting to a site at distance d be d^{-a}.

- For $a > 2$, it takes $O(n^k)$ time for convergence.
- For $a < 2$, it takes $O(log(n)^k)$ time for convergence.

Gossip Based Algorithms

Paper

A Gossip Style Failure Detection Service by Robert Renesse, Yaron Minsky, and Mark Hayden (Technical Report)

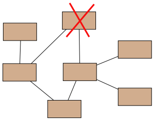
- Problem: A set of nodes fail. Design a failure detector that detects failures by gossiping.
- Model of failure: Fail-Stop $\Rightarrow$ If a node does not respond to a message for T seconds, then it has most likely failed.
- The algorithm needs to scale in terms of the number of nodes, n.

Outline

Aims of a Protocol

- Probability of a false positive is independent of n.
- Resilient to message loss and network partitions.
- Scalability in detection time: $O(nlog(n))$
- If clock drift across nodes is negligible, then the algorithm detects all failures with a known probability of a mistake.
- Bandwidth increase is linear in terms of processes

Basic Protocol



- Each node maintains a message list. (member_id, timestamp , heartbeat counter)
- Every T_{gossip} seconds, each node updates its heartbeat counter, and sends a gossip message to a randomly chosen node.
- The gossip message contains the member ids, and their heartbeats.
- The receiver merges the message lists (takes the larger heartbeat), and adopts the larger heartbeat counter for a node.
- The timestamp for a member indicates the last time that the receiver thinks that a node has updated its heartbeat counter.

Failure Detection

Failure Detection

- If the heart beat counter for a node hasn't increased in T_{fail} seconds, then a node presumes that it has failed.
- Let the probability of a false positive be bounded by P_{fail}.
- However, the entry is not removed because a node can continue to get gossips about the failed node, possibly from other nodes.
- It thus waits for more time and removes the node after $T_{cleanup}$ seconds.
- Let the probability of the node still being alive be bounded by $P_{cleanup}$
- $(P_{fail} = P_{cleanup})$ if $T_{cleanup} = 2 \times T_{fail}$

Outline

Analysis

- Assume that f out of n members have failed.
- k out of n members are infective.
- Only one node sends one message to another node in a round.
- Probability of incrementing the number of infective nodes:

$$P_{inc}(k) = \frac{k}{n} \times \frac{n-f-k}{n-1}$$

- Probability of having k infected members in round $i+1$ is $P(k_{i+1})$.
- Hence,

$$\begin{aligned} P(k_{i+1} = k) &= P_{inc}(k-1) \times P(k_i = k-1) \\ &+ (1 - P_{inc}(k)) \times P(k_i = k) \end{aligned}$$

- Probability that there is some process that does not get infected by p after r rounds is $P_{mistake}(p, r)$

$$P_{mistake}(p, r) = 1 - P(k_r = n - f)$$

-

$$\begin{aligned} P_{mistake}(r) &= \bigcup P_{mistake}(p, r) \\ &\leq (n - f)(1 - P(k_r = n - f)) \end{aligned}$$

Performance

- Number of members are increased from 1 to 200.
- Detection time (seconds) varies linearly in the log scale

$p = 10^{-9}$ Increases from roughly 0 to 250 seconds.
$p = 10^{-6}$ Increases from roughly 0 to 210 seconds.
$p = 10^{-3}$ Increases from roughly 0 to 150 seconds.

- 1 member has failed, 250 bytes per second (bandwidth restriction), $p = P_{mistake}$
- $time = O(nlog(n))$

Performance-II

- Let us vary $P_{mistake}$ from 10^{-10} to 10^{-1} in the log scale .
- With 150 members, the detection time reduces linearly (log scale) from 200s to 95s.
- With 100 members, the detection time reduces linearly from 130s to 60s.
- With 50 members, the detection time reduces linearly from 60s to 25s.

- Detection time vs $P_{mistake}$

Outline

Catastrophe Recovery

- Gossip algorithms do not work in the case of network partitions
- The failure detector needs to broadcast messages to re-establish connections.

Protocol

Broadcast Protocol

- Each second, a node probabilistically decides to send a broadcast.
- The probability depends on the last time a node received a broadcast.
- If a node received a broadcast 20 seconds ago, then it broadcasts with very high probability .
- A function of the form:

$$p(t) = \frac{t}{20}^{a}$$

fits well.

- A Gossip Style Failure Detection Service by Robert Renesse, Yaron Minsky, and Mark Hayden (Technical Report)
- Epidemic Algorithms for Replicated Database Maintenance by Alan Demers. Dan, Greene, Carl Hauser, Wes Irish, John. Larson. Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry, PODC 1987