Gossip Protocols

CS 6410

Eugene Bagdasaryan
Gossip Protocols

CS 6410

Eugene Bagdasaryan
What was covered before?

- Paxos
- Byzantine Generals
- Impossibility consensus

What are these ideas aimed for?

What is the difference with the current paper?
What was covered before?

- Paxos
- Byzantine Generals
- Impossibility consensus

What are these ideas aimed for?
Data consistency, fault-tolerance

What is the difference with the current paper?
“Eventual” consistency, scalability, fault-tolerance
CAP theorem

CAP = Consistency, Availability, Partition tolerance

- Previous papers focused on Consistency and Partition Tolerance
  Paxos sometimes is unavailable for writes, but would remain consistent

- This paper wants to provide Availability, Partition Tolerance, and “relaxed” form of consistency
EPIDEMIC ALGORITHM FOR REPLICATED DATABASE MAINTENANCE

Xerox Palo Alto Research Center 1987
Authors

- Alan Demers
  Cornell Univ.
- Dan Greene
  Palo Alto Research Center
- Carl Hauser
  Washington State Univ.
- Wesley Irish
  Coyote Hill Consulting LLC
- Scott Shenker
  EECS Berkley
- Doug Terry
  Samsung Research America
- John Larson
  Howard Sturgis
  Dan Swinehart
Real applications

- Uber uses SWIM for real-time platform
- Apache Cassandra internode communication
- Docker’s multi-host networking
- Cloud providers multi node networking (Heroku)
- Serf by Hashicorp
Context

- Xerox wanted to run replicated database on hundred or thousand sites.
- Each update is injected at a single site and must be propagated to all other sites.
- A packet from a machine in Japan to one in Europe may traverse as many as 14 gateways and 7 phone lines.
Problem

- High network traffic to send update over the large set of nodes
- Time to propagate update to all nodes is significant
Problem

- High network traffic to send update over the large set of nodes
- Time to propagate update to all nodes is significant

"For a domain stored at 300 sites, 90,000 mail messages might be introduced each night".
Basic idea
Objective

- Design algorithms that scale gracefully
- Every replica receives every update *eventually*
Objective

- Design algorithms that scale gracefully
- Every replica receives every update eventually

“Replace complex deterministic algorithms for replicated database consistency with simple randomized algorithms that require few guarantees from the underlying communication system.“
Why epidemic? Why gossip?

- Highly available
- Fault-tolerant
- Overhead is tunable
- Fast
- Scalable
- Epidemic spreads eventually to everyone
Types of nodes

- **infective** – node that holds an update it is willing to share
- **susceptible** – node that has not yet received an update
- **removed** – node that has received an update but is no longer willing to share

\[ s + i + r = 1 \]
Types of communication

- Direct mail
- Anti-entropy
- Rumor mongering
DIRECT MAIL

- attempts to notify all other sites of an update soon after it occurs.

- **Social network case** – infected accounts send private message to his whole contact list with malicious link
DIRECT MAIL
DIRECT MAIL

May not know this node
DIRECT MAIL

Messages may be dropped
DIRECT MAIL

- Pros:
  - Fast
- Cons:
  - not reliable
  - heavy load on network
Every site regularly chooses another site at random and by exchanging database contents with it resolves any differences between the two.

Real life case – meet sometimes with old friends and tell all the fun stories about you and your friends.
ANTI-ENTROPY
ANTI-ENTROPY
ANTI-ENTROPY
ANTI-ENTROPY
ANTI-ENTROPY
ANTI-ENTROPY
ANTI-ENTROPY
Anti-entropy

Pros

- Complete sync of all info

Cons

- Very expensive to run

Optimizations:

- Checksums
- Recent Update Lists
- Inverted Index by timestamp
**Push vs Pull**

**Push**

A: Has X, Y, Z

B: Has Y, Z Missing X

**Pull**

A: Has X, Z Missing Y

B: Has X, Y, Z

Multiplications:
- A: +X +Y +Z
- B: +X -Y +Z

Actions:
- Packet (x)
- Retransmit (x)
Pull vs Pull

Pull or Push-pull

\[ p_{i+1} = (p_i)^2 \]

- \( p_i \) - probability of a site remaining susceptible after \( i \)-th round

To remain susceptible, node \( n_1 \) needs to contact another node \( n_2 \) on round \( i+1 \), which is also susceptible (with probability \( p_i \))
Push vs Pull

- Push

\[ p_{i+1} = p_i \left(1 - \frac{1}{n}\right)^n (1 - p_i) \]

- \( p_i \) - probability of a site remaining susceptible after \( i \)-th round
- \( \left(1 - \frac{1}{n}\right) \) – probability an infected node choose everything except the selected node \( n1 \)
- \( n(1 - p_i) \) – amount of infected nodes
Push vs Pull

- Pull or Push-pull

\[ p_{i+1} = (p_i)^2 \]

- Push

\[ p_{i+1} = p_i \left(1 - \frac{1}{n}\right)^n (1-p_i) \approx p_i e^{-1} \]

Pull converges to 0 much faster.
Rumor mongering

- Share an update, while it is hot. When everyone knows about it stop spreading.

- **News case** – newspapers write more articles on trending topics spreading information.
RUMOR MONGERING
RUMOR MONGERING
RUMOR MONGERING
RUMOR MONGERING

Pros

- Less traffic, than Direct mail
- Fast

Cons

- Some sites could miss the information

Can be improved by Complex Epidemics
Complex epidemics

- Hot rumors analogy
- Based on epidemiology literature
  \[ s + i + r = 1, \quad s - \text{susceptible}, \ i - \text{infective}, \ r - \text{removed} \]
- If node contacted already infected node, it loses interest and stops talking with probability \( \frac{1}{k} \)
- If \( k=1 \), 20% will miss the rumor for \( k=2 \) only 6%

\[ s = e^{-(k+1)(1-s)} \]
Complex epidemics

Criteria:

- Residue
  
  Amount of untouched nodes (s) after epidemics ended (i = 0) in \( s + i + r = 1 \)

- Traffic

  \[
  m = \frac{\text{Total update traffic}}{\text{Number of sites}}
  \]

- Delay

  Introduced \( t_{avg} \) and \( t_{last} \)
Variations

- Blind vs. Feedback
- Counter vs. Coin
- Push vs. Pull
- Minimization
- Connection Limit
- Hunting
Table 1. Push, Feedback & Counters

<table>
<thead>
<tr>
<th>Counter</th>
<th>Residue</th>
<th>Traffic</th>
<th>Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>s</td>
<td>m</td>
<td>(t_{avg})</td>
</tr>
<tr>
<td>1</td>
<td>0.176</td>
<td>1.74</td>
<td>11.0</td>
</tr>
<tr>
<td>2</td>
<td>0.037</td>
<td>3.30</td>
<td>12.1</td>
</tr>
<tr>
<td>3</td>
<td>0.011</td>
<td>4.53</td>
<td>12.5</td>
</tr>
<tr>
<td>4</td>
<td>0.0036</td>
<td>5.64</td>
<td>12.7</td>
</tr>
<tr>
<td>5</td>
<td>0.0012</td>
<td>6.68</td>
<td>12.8</td>
</tr>
</tbody>
</table>

Table 2. Push, Blind & Coin

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.960</td>
<td>0.04</td>
<td>19</td>
<td>38</td>
</tr>
<tr>
<td>2</td>
<td>0.205</td>
<td>1.59</td>
<td>17</td>
<td>33</td>
</tr>
<tr>
<td>3</td>
<td>0.060</td>
<td>2.82</td>
<td>15</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>0.021</td>
<td>3.91</td>
<td>14.1</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>0.008</td>
<td>4.95</td>
<td>13.8</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 3. Pull, Feedback & Counters

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.031</td>
<td>2.7</td>
<td>9.97</td>
<td>17.63</td>
</tr>
<tr>
<td>2</td>
<td>0.00058</td>
<td>4.49</td>
<td>10.07</td>
<td>15.39</td>
</tr>
<tr>
<td>3</td>
<td>0.000004</td>
<td>6.09</td>
<td>10.08</td>
<td>14.00</td>
</tr>
</tbody>
</table>
Deletion

- Death Certificates
  - Dormant DC
    - Too long to distribute
    - Can be lost
  - Anti-entropy with Dormant DC
    - Activate DC on sync with another node, if this node doesn’t have it
  - Rumor mongering with Dormant DC
    - Parallel to normal data distribution through rumor mongering
Spatial Distributions

- Different weights on connections between nodes
- Can reduce traffic on critical links
- Favor nearby neighbors
- Trade off between convergence time and average traffic per link
Perspective/Questions?

Perspective
- Fast, eventually consistent protocol
- Low traffic in the system

Potential problems:
- Weird topology can decrease performance
- Byzantine Failures
Astrolabe: a robust and scalable technology for distributed system monitoring, management, and data mining

Robbert van Renesse, Kenneth P. Birman, and Werner Vogels
Rise of Web Services and computer-to-computer systems in 2000-s
Availability and scalability of the system matters more than consistency
Applications require fast data mining
Objective

Build a system that
- Supports scaling
- Fault-tolerant
- Has an eventual consistency
- Guarantees security
- Can be controlled through SQL syntax
Overview

- Design principles
  - Scalability through hierarchy
  - Flexibility through mobile code
  - Robustness through a randomized p2p protocol
  - Security through certificates
- Now used in Amazon.com
ZONE AND MIB

- Zone name: path of zone identifiers from the root
- Management Information Base (MIB): attribute list containing the information associated with the zone
  - Each agent has a local copy of the root MIB, the MIBs of each child of the root
  - Each agent maintains MIB list of child zones
GOSSIP PROTOCOL

- Epidemic p2p protocol to propagate information
- Each zone elects a set of representative agents to gossip on behalf of that zone.
  - An agent may represent more than one zone
- Each agent periodically gossips
  - Picks one of the child zones at random,
  - Picks a random agent from child’s list
  - Sends attributes of all the child zones up to root level.
- Gossip spreads quickly, $O(\log n)$
- For scalability, robustness, and rapid dissemination of updates, eventual consistency is adopted.
EVENTUAL CONSISTENCY

- Probabilistic consistency
- Given an aggregate attribute $X$ that depends on some other attribute $Y$
  - When an update $u$ is made to $Y$, either $u$ itself, or an update to $Y$ made after $u$, is eventually reflected in $X$
  - With probability 1
ASTROLABE IS A FLEXIBLE MONITORING OVERLAY

<table>
<thead>
<tr>
<th>Name</th>
<th>Time</th>
<th>Load</th>
<th>Weblogic?</th>
<th>SMTP?</th>
<th>Word Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>swift</td>
<td>2271</td>
<td>1.8</td>
<td>0</td>
<td>1</td>
<td>6.2</td>
</tr>
<tr>
<td>falcon</td>
<td>1971</td>
<td>1.5</td>
<td>1</td>
<td>0</td>
<td>4.1</td>
</tr>
<tr>
<td>cardinal</td>
<td>2004</td>
<td>4.5</td>
<td>1</td>
<td>0</td>
<td>6.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Time</th>
<th>Load</th>
<th>Weblogic?</th>
<th>SMTP?</th>
<th>Word Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>swift</td>
<td>2003</td>
<td>.67</td>
<td>0</td>
<td>1</td>
<td>6.2</td>
</tr>
<tr>
<td>falcon</td>
<td>1976</td>
<td>2.7</td>
<td>1</td>
<td>0</td>
<td>4.1</td>
</tr>
<tr>
<td>cardinal</td>
<td>2231</td>
<td>1.7</td>
<td>1</td>
<td>1</td>
<td>6.0</td>
</tr>
</tbody>
</table>

Periodically, pull data from monitored systems

From Ken’s slide http://www.cs.cornell.edu/courses/cs614/2006fa/Slides/Epidemics.ppt
## State Merge: Core of Astrolabe Epidemic

### Table 1

<table>
<thead>
<tr>
<th>Name</th>
<th>Time</th>
<th>Load</th>
<th>Weblogic?</th>
<th>SMTP?</th>
<th>Word Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>swift</td>
<td>2003</td>
<td>.67</td>
<td>0</td>
<td>1</td>
<td>6.2</td>
</tr>
<tr>
<td>falcon</td>
<td>1976</td>
<td>2.7</td>
<td>1</td>
<td>0</td>
<td>4.1</td>
</tr>
<tr>
<td>cardinal</td>
<td>2201</td>
<td>3.5</td>
<td>1</td>
<td>1</td>
<td>6.0</td>
</tr>
</tbody>
</table>

From Ken’s slide http://www.cs.cornell.edu/courses/cs614/2006fa/Slides/Epidemics.ppt
### STATE MERGE: CORE OF ASTROLABE EPIDEMIC


#### Table 1: Event Log Details

<table>
<thead>
<tr>
<th>Name</th>
<th>Time</th>
<th>Load</th>
<th>Weblogic?</th>
<th>SMTP?</th>
<th>Word Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>swift</td>
<td>2011</td>
<td>2.0</td>
<td>0</td>
<td>1</td>
<td>6.2</td>
</tr>
<tr>
<td>falcon</td>
<td>1971</td>
<td>1.5</td>
<td>1</td>
<td>0</td>
<td>4.1</td>
</tr>
<tr>
<td>cardinal</td>
<td>2004</td>
<td>4.5</td>
<td>1</td>
<td>0</td>
<td>6.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Time</th>
<th>Load</th>
<th>Weblogic?</th>
<th>SMTP?</th>
<th>Word Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>swift</td>
<td>2011</td>
<td>2.0</td>
<td>0</td>
<td>1</td>
<td>6.2</td>
</tr>
<tr>
<td>falcon</td>
<td>1971</td>
<td>1.5</td>
<td>1</td>
<td>0</td>
<td>4.1</td>
</tr>
<tr>
<td>cardinal</td>
<td>2201</td>
<td>3.5</td>
<td>1</td>
<td>1</td>
<td>6.0</td>
</tr>
</tbody>
</table>
SCALING UP... AND UP...

- We don’t want every system to “see” all others (cost would be huge)
- Instead, structure into “zones”. You only see data from your neighbors...

From Ken’s slide http://www.cs.cornell.edu/courses/cs614/2006fa/Slides/Epidemics.ppt
A FORM OF DATA MINING CONTINUOUSLY SUMMARIZES REMOTE INFORMATION

Dynamically changing query output is visible system-wide

<table>
<thead>
<tr>
<th>Name</th>
<th>Avg Load</th>
<th>Weblogic?</th>
<th>SMTP?</th>
<th>Word Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>swift</td>
<td>2.0</td>
<td>0</td>
<td>1</td>
<td>6.2</td>
</tr>
<tr>
<td>falcon</td>
<td>1.5</td>
<td>1</td>
<td>0</td>
<td>4.1</td>
</tr>
<tr>
<td>cardinal</td>
<td>4.5</td>
<td>1</td>
<td>0</td>
<td>6.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Avg Load</th>
<th>Weblogic?</th>
<th>SMTP?</th>
<th>Word Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF</td>
<td>2.6</td>
<td>123.45.61.3</td>
<td>123.45.61.17</td>
<td></td>
</tr>
<tr>
<td>NJ</td>
<td>1.8</td>
<td>127.16.77.6</td>
<td>127.16.77.11</td>
<td></td>
</tr>
<tr>
<td>Paris</td>
<td>3.1</td>
<td>14.66.71.8</td>
<td>14.66.71.12</td>
<td></td>
</tr>
</tbody>
</table>

From Ken’s slide http://www.cs.cornell.edu/courses/cs614/2006fa/Slides/Epidemics.ppt
(1) QUERY GOES OUT… (2) COMPUTE LOCALLY… (3) RESULTS FLOW TO TOP LEVEL OF THE HIERARCHY

<table>
<thead>
<tr>
<th>Name</th>
<th>Avg Load</th>
<th>Weblogic?</th>
<th>SMTP?</th>
<th>Word Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>swift</td>
<td>2.0</td>
<td>0</td>
<td>1</td>
<td>6.2</td>
</tr>
<tr>
<td>falcon</td>
<td>1.5</td>
<td>1</td>
<td>0</td>
<td>4.1</td>
</tr>
<tr>
<td>cardinal</td>
<td>4.5</td>
<td>1</td>
<td>0</td>
<td>6.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Avg Load</th>
<th>Weblogic?</th>
<th>SMTP?</th>
<th>Word Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>gazelle</td>
<td>1.7</td>
<td>0</td>
<td>0</td>
<td>4.5</td>
</tr>
<tr>
<td>zebra</td>
<td>3.2</td>
<td>0</td>
<td>1</td>
<td>6.2</td>
</tr>
<tr>
<td>gnu</td>
<td>0.5</td>
<td>1</td>
<td>0</td>
<td>6.2</td>
</tr>
</tbody>
</table>

From Ken’s slide http://www.cs.cornell.edu/courses/cs614/2006fa/Slides/Epidemics.ppt
Yellow leaf node “sees” its neighbors and the domains on the path to the root.

From Ken’s slide http://www.cs.cornell.edu/courses/cs614/2006fa/Slides/Epidemics.ppt
Average # of rounds necessary to infect all nodes
CONCLUSION

- Tree-based gossip protocol
- Robust and scalable
- Eventual Consistency
Thank you