Gossip Techniques

Makoto Bentz
mb434@cs.cornell.edu
Oct. 27, 2010
What is Gossip?

- Gossip is the periodic **pairwise** exchange of bounded size messages between **random nodes** in the system in which nodes states may affect each other.
- Has $O(\log n)$ completion time.
- **Benefits:** simplicity, limited resource usage, robustness to failures, and tunable system behavior.
How is Gossip Different?

- **Unicast:** One person tells one person

- **Broadcast:** One node tells everyone

- **Multicast:** One person tells all via intermediary nodes

- **Gossip:** Everyone tells someone else what they know
Eventual Consistency

- **Strong Consistency**: After the update completes, any subsequent access will return the updated value.

- **Weak consistency**: System doesn’t guarantee subsequent accesses will return the updated value. A number of conditions need to be met before the value will be returned.

- **Eventual consistency**: Subset of weak consistency; the system guarantees that **if no new updates** are made to the object, **eventually** all accesses will return the **last updated value**.
Gossip Techniques: Papers


• *Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining*, Van Renesse et al. ACM TOCS 2003.

• *Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead*, Indranil Gupta, Ken Birman, Prakash Linga, Al Demers and Robbert van Renesse. 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03); February 20-21, 2003. Claremont Hotel, Berkeley, CA, USA.
Epidemic Algorithms: Authors

Dan Greene is at Xerox parc
His research now focuses on vehicle networks

Alan Demers is a researcher at Cornell University

Carl Hauser is an Associate Professor at Washington State University
Epidemic Algorithms: Authors

Scott Shenker is an associate professor at U.C. Berkeley

Wes Irish now runs Coyote Hill Consulting LLC

Doug Terry is the Primary Researcher at Microsoft Research Silicon Valley
Epidemic Algorithms: Authors

• John Larson worked on Cedar DBMS and LDAP and at Sprint Advanced Technology Labs

• Howard Sturgis discovers 2-phase transaction commit and worked on Cedar DBMS and RPCs

• Dan Swinehart worked on Bayou
Epidemic Algorithms: Status Quo

Networks
Computers
Epidemic Algorithms: Problem Statement

- Clearinghouse Servers on Xerox Corporate Internet
- Several hundred Ethernets connected by gateways and phone lines
  - Several thousand computers
- Three-level hierarchy with top two levels being domains
- Need to keep databases on computers between domains (eventually) consistent
Epidemic Algorithms: First Attempt

- Originally using what was a rudimentary form of Direct Mail (Multicast) and Anti-Entropy (Gossip)

- Inefficient/Redundant
  - Anti-Entropy was being redundantly followed by Direct Mail, saturating the network (300 clients -> 90,000 mail messages)

- Not scalable
  - Network capacity saturated -> failure
Epidemic Techniques: What are they?

• “Epidemic algorithms follow the paradigm of nature by applying simple rules to spread information by just having a local view of the environment” Hollerung, Bleckmann

• Conway’s Game of Life is an epidemic algorithm

• Medical epidemics spread between individuals by contagion
## Epidemic Algorithms: Types of Spreading

<table>
<thead>
<tr>
<th>Unit Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Susceptible</td>
<td>Does not know info, but can get info</td>
</tr>
<tr>
<td>Infective</td>
<td>Knows the info and spreads it by the rule</td>
</tr>
<tr>
<td>Removed</td>
<td>Knows the info but does not spread it</td>
</tr>
</tbody>
</table>

Can be combinations of the above.
Epidemic Algorithms: Direct Mail

- **Direct Mail**: Send to everyone
- **Send**
  - FOR EACH s’ in S
    - DO PostMail[to: s’, msg : (“Update”, s.ValueOf)]
    - ENDLOOP
- **Receive**
  - IF s.ValueOf.t < t THEN
    - s.ValueOf - (7!,t)
- **Susceptible to failure**, $O(n)$ bottleneck, Original could have incomplete information
- Xerox system did not use broadcast mailing
Epidemic Algorithms: Anti-Entropy

• Anti-Entropy: Everyone picks a site at random, and resolves differences between it and its recipient

• FOR SOME s’ in S
  DO ResolveDifference[s, s’]
ENDLOOP

• Resolving can be done by push, pull, push-pull

• Slower than direct mail, and **expensive** to compare databases
Epidemic Algorithms: Anti-Entropy: Resolving

• Push
  ResolveDifference : PROC[.s, s’] = {
    IF s.Value0f.t > s’.Value0f.t THEN
      s’.ValueOf <- s.ValueOf
  }

• Pull
  ResolveDifference : PROCis, s’] = {
    IF s.Value0f.t < s’.Value0f.t THEN
      s.ValueOf + s’.ValueOf
  }

• Push-Pull
  ResolveDifference : PR.OC’[s. s’] = {
    SELECT TRUE FROM
      s.Value0f.l > s’.Value0f.t => s’.ValueOf - s.ValueOf;
      s.Value0f.t < s’.Value0f.t => s.ValueOf - s’.ValueOf;
    ENDCASE => NULL;
  }

Push converges much slower than pull or push-pull
Epidemic Algorithms: Rumor Spreading

1. There are initially no active people, each person with a rumor is active

2. Someone gets the rumor

3. Each active person then randomly phones other persons to tell them the rumor

4. If the recipient already knows the rumor, then the sender loses interest and becomes inactive
Epidemic Algorithms: Rumor Spreading

- **Blind vs. Feedback**
  Blind senders lose interest with probability $1/k$
  Feedback senders lose interest dependent on the recipient

- **Counter vs. Coin**
  Counter loses interest after $k$ unnecessary contacts
  Coin loses interest after a $1/k$ probability coin toss upon unnecessary contacts
Epidemic Algorithms: Theory

\[ s + i + r = 1 \]

\[
\begin{align*}
\frac{ds}{dt} &= -si \\
\frac{di}{dt} &= +si - \frac{1}{k}(1 - s)i \\
\frac{di}{ds} &= -\frac{k + 1}{k} + \frac{1}{ks} \\
i(s) &= -\frac{k + 1}{k} s + \frac{1}{k} \log s + c
\end{align*}
\]
Epidemic Algorithms: Backing up

- A complex epidemic may not converge
- Back up by adding anti-entropy as well as rumor mongering
  - Direct mail is $O(n^2)$ per cycle at worst case
  - Rumor mongering is always $O(n)$ or less
- Death certificates carry timestamps marking deletion
  - Dormant death certificates do not scale well (deletion time ~ $O(\log n)$)
  - Activation timestamp added to death certificate to prevent rollback of data changed after a death certificate first went out
Epidemic Algorithms: Testing

Table 5. Simulation results for anti-entropy, connection limit 1.

<table>
<thead>
<tr>
<th>Spatial Distribution</th>
<th>$t_{last}$</th>
<th>$t_{ave}$</th>
<th>Compare</th>
<th>Traffic</th>
<th>Update</th>
<th>Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Average</td>
<td>Bushey</td>
<td>Average</td>
<td>Bushey</td>
</tr>
<tr>
<td>uniform</td>
<td>11.00</td>
<td>6.97</td>
<td>3.71</td>
<td>47.54</td>
<td>5.83</td>
<td>75.17</td>
</tr>
<tr>
<td>$a = 1.2$</td>
<td>16.89</td>
<td>9.92</td>
<td>1.14</td>
<td>6.39</td>
<td>2.69</td>
<td>18.03</td>
</tr>
<tr>
<td>$a = 1.4$</td>
<td>17.34</td>
<td>10.15</td>
<td>1.08</td>
<td>4.68</td>
<td>2.55</td>
<td>13.68</td>
</tr>
<tr>
<td>$a = 1.6$</td>
<td>19.06</td>
<td>11.06</td>
<td>0.94</td>
<td>2.90</td>
<td>2.32</td>
<td>10.20</td>
</tr>
<tr>
<td>$a = 1.8$</td>
<td>21.46</td>
<td>12.37</td>
<td>0.82</td>
<td>1.68</td>
<td>2.12</td>
<td>7.03</td>
</tr>
<tr>
<td>$a = 2.0$</td>
<td>24.64</td>
<td>14.14</td>
<td>0.72</td>
<td>0.94</td>
<td>1.94</td>
<td>4.85</td>
</tr>
</tbody>
</table>
Epidemic Algorithms: Discussion

• I felt like this paper started to rush near the end
  • Great explanation of the theory, weak explanation of the testing and implementation

• This paper goes on to be the foundation of Gossip
  • Cited at least 249+18(PDOC+SIGOPS) times
Bayou: Authors

Alan Demers is a researcher at Cornell University

Carl Hauser is a Associate Professor at Washington State University

Doug Terry is the Primary Researcher at Microsoft Research Silicon Valley
Bayou: Authors

- **Marvin Theimer** is the Senior Principal Engineer at Amazon Web Services

  ![Marvin Theimer]

- **Michael Spreitzer** works in Services Management Middleware at Thomas J. Watson Research Center, Hawthorne, NY USA

  ![Michael Spreitzer]
• TOP 10 Reasons for the name "Bayou":

• 10. Why not?

• 9. It's better than "UbiData".

• 8. It's a lot better than "DocuData".

• 7. It's not an acronym.

• 6. It's not named after a soft drink (e.g. Tab, Sprite, Coda Cola, ...).

• 5. We're working on replication that's "fluid" like a bayou.

• 4. We're exploring a small part of the "UbiComp Swamp".

• 3. It's the name of a famous tapestry (spelled "Bayeux" however).

• 2. Our system will allow you to access data even when you're "bayou self".

• 1. It's pronounced "Bi-U", which makes it "Ubi" pronounced backwards.
Bayou: The Problem

- Wireless and mobile devices do not permit constant connectivity
- Weak connectivity
- Collaborative applications such as calendars
Bayou: The Design

- Data collections are replicated at Servers
- Clients run applications that access the servers via an API
  - Read and Write
- Each server stores an ordered log of Writes and the resulting data
- Performs Writes and Conflict Detection
- Anti-Entropy to propagate updates
Bayou: Design: Conflict Detection

• Dependency Checks
  • Application Specific Conflict Checks
  • Write is accompanied with query and expected result required to write (ex. to reserve 2, the set of reserved should not include 2)

• Merge Procedure
  • Conflict Detected -> Merge Procedure
  • High-level, interpreted language code to pick a result in merge
  • Does not lock conflicted data
Bayou: Design: Eventual Consistency

• Bayou replicas all follow Eventual Consistency

• This is ensured by the following two rules
  
  • Writes are performed in order

  • Conflict Detection and Merge procedure are deterministic, resulting in the same resolve at the server

• Writes are stable after they have been executed for the last time

• Commits will ensure stability
Bayou: Implementation

- Tuple Store, in-memory relational database
- Access Control by public-key cryptography, allows for grants, delegation and revocation
Bayou: Implementation

- Written in ILU (an RPC) and Tcl
- Per-database library mechanism for each write to prevent replicated code

Table 1: Size of Bayou Storage System for the Bibliographic Database with 1550 Entries
(sizes in Kilobytes)

<table>
<thead>
<tr>
<th>Number of Tentative Writes</th>
<th>0 (none)</th>
<th>50</th>
<th>100</th>
<th>500</th>
<th>1550 (all)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Log</td>
<td>9</td>
<td>129</td>
<td>259</td>
<td>1302</td>
<td>4028</td>
</tr>
<tr>
<td>Tuple Store Ckpt</td>
<td>396</td>
<td>384</td>
<td>371</td>
<td>269</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>405</td>
<td>513</td>
<td>630</td>
<td>1571</td>
<td>4029</td>
</tr>
<tr>
<td>Factor to 368K bibtex source</td>
<td>1.1</td>
<td>1.39</td>
<td>1.71</td>
<td>4.27</td>
<td>10.95</td>
</tr>
</tbody>
</table>
## Bayou: Implementation

### Table 2: Performance of the Bayou Storage System for Operations on Tentative Writes in the Write Log

(times in milliseconds with standard deviations in parentheses)

<table>
<thead>
<tr>
<th>Tentative Writes</th>
<th>0</th>
<th>50</th>
<th>100</th>
<th>500</th>
<th>1550</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Server running on a Sun SPARC/20 with Sunos</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undo all (avg. per Write)</td>
<td>0</td>
<td>31</td>
<td>(6)</td>
<td>70</td>
<td>(20)</td>
</tr>
<tr>
<td>Redo all (avg. per Write)</td>
<td>0</td>
<td>237</td>
<td>(85)</td>
<td>611</td>
<td>(302)</td>
</tr>
<tr>
<td></td>
<td>4.74</td>
<td>6.11</td>
<td></td>
<td></td>
<td>5.59</td>
</tr>
<tr>
<td><strong>Server running on a Gateway Liberty Laptop with Linux</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undo all (avg. per Write)</td>
<td>0</td>
<td>47</td>
<td>(3)</td>
<td>104</td>
<td>(7)</td>
</tr>
<tr>
<td>Redo all (avg. per Write)</td>
<td>0</td>
<td>302</td>
<td>(91)</td>
<td>705</td>
<td>(134)</td>
</tr>
<tr>
<td></td>
<td>6.04</td>
<td>7.05</td>
<td></td>
<td></td>
<td>7.01</td>
</tr>
</tbody>
</table>

### Table 3: Performance of the Bayou Client Operations

(times in milliseconds with standard deviations in parentheses)

<table>
<thead>
<tr>
<th>Server Client</th>
<th>Sun SPARC/20 same as server</th>
<th>Gateway Liberty same as server</th>
<th>Sun SPARC/20 Gateway Liberty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read: 1 tuple</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 tuples</td>
<td>27 (19)</td>
<td>38 (5)</td>
<td>23 (4)</td>
</tr>
<tr>
<td>Write: no conflict</td>
<td>159 (32)</td>
<td>212 (29)</td>
<td>177 (22)</td>
</tr>
<tr>
<td>with conflict</td>
<td>207 (37)</td>
<td>372 (17)</td>
<td>223 (40)</td>
</tr>
</tbody>
</table>
Bayou: Discussion

- Was a well-written paper
- Industry paper, testing not well explained
Resources