A STUDY OF GOSSIP ALGORITHMS FOR INTERNET-SCALE CARDINALITY ESTIMATION OF DISTRIBUTED XML DATA

Vasil G. Slavov
Computer Science & Electrical Engineering
University of Missouri-Kansas City

June 22nd 2012

Acknowledgements
National Science Foundation (IIS-1115871)
Thesis Committee

♦ Praveen R. Rao, Ph.D., Committee Chair
♦ Yugyung Lee, Ph.D.
♦ Deep Medhi, Ph.D.
Overview

♦ Introduction
♦ Background
♦ Thesis objectives
♦ Design
  ♦ VanillaXGossip
  ♦ XGossip
♦ Implementation
♦ Performance evaluation
  ♦ Amazon Elastic Compute Cloud (EC2)
♦ Conclusions
Introduction

♦ XML and XPath – W3C standards

```xml
<ClinicalDocument>
  <typeId extension="POCD_HD000040" root="2.16.840.1.113883.1.3"/>
  <id extension="CSE001" root="2.16.840.1.113883.19.4"/>
  <code code="8647-0" codeSystem="2.16.840.1.113883.6.1" codeSystemName="LOINC" displayName="Hospital Consultations"/>
  <confidentialityCode code="V" codeSystem="2.16.840.1.113883.5.25"/>
  <RecordTarget>
    <Patient>
      <ID>711</ID>
    </Patient>
    <ProviderOrganization id root="2.16.840.1.113883.19.5"/>
  </RecordTarget>
  <representedOrganization>UMKC School of Computing & Engineering (Research)</representedOrganization>
</ClinicalDocument>
```

XPath

```
/ClinicalDocument[RecordTarget/PatientRole/Patient = "M"] [RecordTarget/PatientRole/ID]
```

```
<section>
  <code code="" codeSystem="2.16.840.1.113883.6.1" codeSystemName="LOINC"/>
  <title>History of Present Illness</title>
  <text>
    The patient was a 108-year-old nursing home resident, who was admitted with a two-day history of increased respiratory secretions and a 24-hour history of elevated fever. Despite Augmentin, the patient's delirium worsened in the 24 hours prior to admission, and her temperature was up to 102. She was refusing to take p.o.'s
  </text>
</section>
```

```
<section>
  <code code="11348-0" codeSystem="2.16.840.1.113883.6.1" codeSystemName="LOINC"/>
  <title>Past Medical History</title>
```
The Story So Far...

Peer-to-peer Computing
(Distributed Hash Tables)

XML $\cap$ P2P $\cap$ Gossip

Gossip (or epidemic) algorithms

Galanis et.al. [VLDB ‘03],
XPeer [P2P&DB ‘04],
XP2P [WIDM ‘04],
Garces et al. [ICDCS ‘04],
Skobeltsyn et.al. [ODBASE’05],
KadoP [ICDE ‘08],
XTreeNet [VLDB ‘08],
psiX [TKDE ‘09, ICDE ‘09],
...

Chord [SIGCOMM ‘01],
CAN [SIGCOMM ‘01],
Pastry [Middleware ‘01],
Tapestry [JSAC ‘04],
Kademlia [IPTPS ‘02]
Dynamo [SOSP ‘07],
Cassandra [SIGMOD ‘08],
Voldemort [ICDE ‘11],
...

Demers et. al. [PODC ‘87],
Karp et. al. [FOCS ‘00],
Kempe et.al. [FOCS ‘03],
Berger et.al. [SODA ’05],
Ganesh et. al. [INFOCOMM ‘05],
Boyd et. al. [INFOCOMM ‘05],
Jelasity et. al. [TOCS ‘05],
Kashyap et. al. [PODS ‘06],
Georgiou et. al. [PODC ‘08],
Mosk-Aoyama et. al. [TOIT ’08],
...
XML ∩ P2P = ?

Large-scale sharing of biomedical and clinical data

Scalable clinical data sharing systems via a P2P architecture [Stead and Lin, 2009]

HL7 version 3 standard; XML based; semantic interoperability; can model discharge summaries, lab reports, …

The Cancer Biomedical Informatics Grid (caBIG): a real world data sharing platform
The Problem: XPage Cardinarity Estimation

♦ Compute the number of documents in the network that contain a match for the expression /Gene//goAcc

♦ Useful for query optimization

♦ Desired properties
  ♦ Scalability
  ♦ Decentralization
  ♦ Fault-tolerance
  ♦ Efficient usage of bandwidth
  ♦ Provable guarantee on the quality of the estimate
Gossip Algorithms

♦ Communication, computation, and information spreading
♦ Attractive in large-scale, distributed systems
♦ Real world examples: Amazon S3, Dynamo, Cassandra

Gossip algorithms

- Information exchange
  - Synchronous
  - Asynchronous

- Aggregate computation
  - Synchronous
  - Asynchronous

Can further classify based on the topology of the network
Push-Sum Protocol

- By Kempe, Dobra, and Gehrke [FOCS ‘03]
- Each peer wishes to know the average
- Each peer maintains a (sum, weight) pair during gossip
  - In round $@ t = 0$: send $(f_1, 1)$ to itself
  - In any round $@ t > 0$: add up sums & weights, send $(s_1, s_2)/2, (w_2, w_2)/2$ to itself and to random peer
- Convergence ($n$ peers)
  - Rounds: $O(\log(n) + \log(1/\varepsilon) + \log(1/\delta))$
  - Messages: $n$ messages per round
- Proof is based on the property of “mass conservation”
Thesis Objectives

1. Implementing gossip in an Internet-scale environment
2. Conducting a comprehensive evaluation
3. Analyzing the experimental results
Design of VanillaXGossip

- Builds on Push-Sum
- XML documents are mapped to their signatures
  - $\psi X$ [Rao et al. TKDE ‘09, ICDE ‘09]
  - XML doc. $\rightarrow$ data signature; XPath query $\rightarrow$ query signature

Each peer creates a sorted tuple list
Merging Phase

♦ Suppose a peer receives 3 tuple lists during a gossip round

\[ s_1, (f_1, w_1) \]
\[ s_2, (f_2, w_2) \]
\[ \bot, (f_a, w_a) \]

\[ s_1, (f_3, w_3) \]
\[ s_3, (f_5, w_5) \]
\[ \bot, (f_b, w_b) \]

\[ s_1, (f_4, w_4) \]
\[ \bot, (f_c, w_c) \]

\[ T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_m \]

- \[ f_a = f_b = f_c = 0 \]
- \[ \text{sum}_{f_1} = \frac{(f_1 + f_3 + f_4)}{2} \]
- \[ \text{sum}_{w_1} = \frac{(w_1 + w_3 + w_4)}{2} \]
- \[ \text{sum}_{f_2} = \frac{(f_2 + f_b + f_c)}{2} \]
- \[ \text{sum}_{w_2} = \frac{(w_2 + w_b + w_c)}{2} \]
- \[ \text{sum}_{f_3} = \frac{(f_a + f_5 + f_c)}{2} \]
- \[ \text{sum}_{w_3} = \frac{(w_a + w_5 + w_c)}{2} \]
- \[ \text{sum}_{f_4} = \frac{(f_a + f_b + f_c)}{2} \]
- \[ \text{sum}_{w_4} = \frac{(w_a + w_b + w_c)}{2} \]

\[ T_m \rightarrow \text{randomly selected peer} \]
VanillaXGossip

♦ Special multiset ⊥
  ♦ Placeholder for signatures not yet known to a peer during a gossip round
  ♦ Preserves the property of “mass conservation”

♦ Convergence
  ♦ Rounds: $O(\log(n) + \log(1/\varepsilon) + \log(1/\delta))$

Problem 😞
A peer will end up with all the distinct signatures
More memory, more bandwidth
XGossip

♦ Idea

♦ Divide-and-conquer approach using Locality Sensitive Hashing (LSH)

♦ A subset of peers are responsible for gossiping a subset of distinct signatures

Benefits 😊

Less memory, less bandwidth, faster convergence
Locality Sensitive Hashing (LSH)

- Introduced by Indyk and Motwani [STOC ’98]
- Applications
  - Web clustering, computer vision, computational biology, etc.
- Idea
  - Use many hash functions
  - Probability of collision is higher for inputs that are more similar
- LSH on sets using Jaccard index [WWW ‘02, WWW ‘05]
  - \( P[h(s_1) = h(s_2)] = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|} \)
  - \( h() \rightarrow \) min-hashing
LSH on Sets

♦ Suppose $p = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|}$

♦ Pick $k \times \ell$ random linear hash functions

Output of $\ell$ hash functions

\[ P[\text{at least one pair of yellow and green is identical}] = 1 - (1 - p^\ell)^k \]

Can pick $k$ and $\ell$ s.t.
- High probability if similarity $\geq p$
- Low probability if similarity $< p$
XGossip (1/2)

♦ Define $k$ teams for a signature $s$
  ♦ $\text{LSH}(s) \rightarrow \{h_1, \ldots, h_k\}$
  ♦ Each team has id $h_i$, $1 \leq i \leq k$; $\Delta$ denotes team size
  ♦ $\alpha = 1 - (1 - p^\ell)^k$

$\Delta = 4$

- denotes a peer

Cardinality estimation: more likely to find all the required signatures in the same team
XGossip (2/2)

♦ Initialization and execution phases
♦ Convergence
  ♦ Rounds: $O(\log(\Delta) + \log(1/\epsilon) + \log(1/\delta))$
♦ Bandwidth reduction
  ♦ Compress signatures when sending a gossip message
VanillaXGossip: Cardinality Estimation

Suppose a peer wants to compute \( \text{card}(\text{/Gene}//\text{goAcc}) \)

Tuple list \( T \) @ the peer

<table>
<thead>
<tr>
<th>Signature</th>
<th>(sum,weight)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_i )</td>
<td>( (f_i, w_i) )</td>
</tr>
</tbody>
</table>

Sum the frequency estimates of signatures that are supersets of the query signature; multiply by \( n \)
XGossip: Cardinality Estimation

Suppose a peer wants to compute card(/Gene//goAcc)

Apply LSH on the query signature $\rightarrow$ $k$ teams

Merge the frequency estimates of signatures that are supersets of the query signature; multiply by $\Delta$
Implementation

♦ Built on top of the Chord DHT [SIGCOMM ‘01]
  ♦ Use Chord for routing (key-value pairs)
  ♦ 4 processes per peer: Chord (lsd, synsd, adbd), Gossip (gpsi)
  ♦ Communicate over UNIX sockets
  ♦ Read signatures from files, store in main memory

♦ Implemented in C++
  ♦ Data structures from STL
    ♦ typedef std::map<std::vector<POLY>, std::vector<double>, CompareSig> mapType;
    ♦ typedef std::map<chordID, std::vector<int>> teamid2totalT;

♦ SFSlite asynchronous library

♦ Challenges
System Architecture
Performance Evaluation

♦ Datasets

- Generated by the IBM Synthetic XML generator using well-known DTDs (Treebank, DBLP, etc.)
- Uniformly distributed among all peers

<table>
<thead>
<tr>
<th>Dataset</th>
<th># of DTDs</th>
<th>Avg # of docs per DTD</th>
<th>Total # of docs</th>
<th>Avg. document signature size</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>11</td>
<td>190,809</td>
<td>2,098,900</td>
<td>114 bytes</td>
</tr>
<tr>
<td>D2</td>
<td>13</td>
<td>192,223</td>
<td>2,498,900</td>
<td>127 bytes</td>
</tr>
</tbody>
</table>
Performance Evaluation

Query sets
- XPath queries generated by YFilter [TODS ‘03]
- A total of 753 queries
- 2 query approaches
  - LSH(XPath sig)
  - LSH(proxy sig)
- Query subsets based on $p_{\text{min}}$ value

<table>
<thead>
<tr>
<th>Query Set</th>
<th>Value of $p_{\text{min}}$</th>
<th># of queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_0$</td>
<td>[0, 0.5)</td>
<td>101</td>
</tr>
<tr>
<td>$Q_1$</td>
<td>[0.5, 1]</td>
<td>652</td>
</tr>
<tr>
<td>$Q_2$</td>
<td>[0.6, 1]</td>
<td>356</td>
</tr>
<tr>
<td>$Q_3$</td>
<td>[0.7, 1]</td>
<td>300</td>
</tr>
<tr>
<td>$Q_4$</td>
<td>[0.8, 1]</td>
<td>277</td>
</tr>
<tr>
<td>$Q_5$</td>
<td>[0.9, 1]</td>
<td>26</td>
</tr>
</tbody>
</table>
Network Setup and Distribution of Documents

♦ Amazon EC2
  ♦ 20 medium instances (2 cores, 1.7GB memory)
  ♦ US East availability zone

♦ Peers use local clock
♦ 120s long rounds

♦ Variables
  ♦ team size (Δ): 8, 16
  ♦ LSH parameter k: 4, 8
  ♦ LSH parameter l: 10 (fixed): α = 0, when p < 0.5
  ♦ compression
  ♦ # of peers: 500, 1000, 2000

<table>
<thead>
<tr>
<th>Total # of peers in the network (n)</th>
<th># of peers per EC2 instance</th>
<th># of peers picked per DTD (z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>25</td>
<td>250</td>
</tr>
<tr>
<td>1000</td>
<td>50</td>
<td>500</td>
</tr>
<tr>
<td>2000</td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>
Evaluation Metrics

♦ Convergence speed of the frequency of signatures
  ♦ Mean absolute relative error (MARE) of the frequency estimate of the document signatures

\[
\frac{1}{M} \times \sum_{i=1}^{M} \frac{|e f_i - f_i|}{f_i}
\]

♦ Where
  ♦ M: tuple list size
  ♦ \(f_i\): true signature frequency
  ♦ \(ef_i\): estimated signature frequency
  ♦ VanillaXGossip: freq / weight * n
  ♦ XGossip: freq / weight * \(\Delta\)

♦ Accuracy of cardinality estimation
  ♦ MARE of the cardinality estimate of the queries

♦ Bandwidth consumption during gossip
  ♦ Amount of data transmitted per round by all peers
XGossip in the Cloud

Amazon EC2

Card(/Gene//goAcc)
Diffusion Speed of Signatures

VanillaXGossip
Convergence of Frequencies of Signatures

**VanillaXGossip**

![Graph showing mean abs. relative error over rounds for VanillaXGossip with dataset D_1, n = 1000, Δ = 8, k = 8, l = 10.]

**Dataset D_1, n = 1000**

**XGossip**

![Graph showing mean abs. relative error over rounds for XGossip with dataset D_1, n = 1000, Δ = 8, k = 8, l = 10.]

Dataset D_1, n = 1000
Δ = 8, k = 8, l = 10
Accuracy of Cardinality Estimation (1/6)

VanillaXGossip vs. XGossip

At different rounds

After 20 rounds

Relative error below 20%

n = 1000, Δ = 8, k = 8, l = 10
Accuracy of Cardinality Estimation (2/6)

XGossip: LSH and team size

After 30 rounds, n = 1000, l = 10
Accuracy of Cardinality Estimation (3/6)

**XGossip**

\[ \Delta = 8, \, k = 4 \]

\[ \Delta = 8, \, k = 8 \]

- \( n = 1000 \), after 30 rounds
- \( \Delta = 16, \, k = 4 \) is almost identical
- \( \Delta = 16, \, k = 8 \) is almost identical
Accuracy of Cardinality Estimation (4/6)

**XGossip: increasing # of rounds**

![Bar chart showing error range with different rounds](chart.png)

\[ n = 1000, \Delta = 8, k = 8, l = 10 \]
Accuracy of Cardinality Estimation (5/6)

Varying # of peers

Relative error below 20%
\[ \Delta = 8, k = 8, l = 10 \]

Varying # of peers (query set \( Q_3 : [0.7, 1] \))

Relative error below 10%
\[ n = 1000, \Delta = 8, k = 8, l = 10 \]
Accuracy of Cardinality Estimation (6/6)

XGossip

500 peers

After 5 rounds
$\Delta = 8$, $k = 8$, $l = 10$

1000 peers
Bandwidth Consumption (1/2)

**VanillaXGossip vs. XGossip**

*(Dataset D₁)*

- VanillaXGossip: 10,309 MB
- XGossip: 484 MB

**XGossip compression**

*(Dataset D₂)*

- XGossip (no compression): 9,874.2 MB
- XGossip: 1,805.9 MB
Bandwidth Consumption (2/2)

**XGossip**

Different values of \( k \) and \( \Delta \)

Varying the # of peers

\[ n = 1000 \]

\[ l = 10 \]
## Performance Analysis: Results Summary

- ♦ LSH: tune k and l
- ♦ Compression works
- ♦ XGossip scales, VanillaXGossip does not
- ♦ XGossip converges faster than VanillaXGossip
- ♦ XGossip transmits less data than VanillaXGossip

<table>
<thead>
<tr>
<th># of peers</th>
<th>Avg. # of teams/peer</th>
<th>Avg. # of sigs/peer</th>
<th>Avg. # of sigs/team</th>
<th>Avg. msg size/peer (bytes)</th>
<th>Total # of msgs</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>88.40</td>
<td>4,024.25</td>
<td>45.52</td>
<td>1,160.18</td>
<td>440,240</td>
</tr>
<tr>
<td>1000</td>
<td>44.82</td>
<td>2,040.81</td>
<td>45.52</td>
<td>1,265.64</td>
<td>440,240</td>
</tr>
<tr>
<td>2000</td>
<td>23.09</td>
<td>1,051.20</td>
<td>45.52</td>
<td>1,244.91</td>
<td>441,750</td>
</tr>
</tbody>
</table>
Conclusion

♦ Thesis objectives
  1. Implementing gossip in an Internet-scale environment
  2. Conducting a comprehensive evaluation
  3. Analyzing the experimental results

♦ The results we obtained were consistent with the theoretical analysis of VanillaXGossip and XGossip.
Questions?

♦ References


♦ Acknowledgements

♦ National Science Foundation (IIS-1115871), 2011-2014