Towards Internet-Scale Cardinality Estimation of XPath Queries over Distributed XML Data

Vasil G. Slavov
University of Missouri-Kansas City
Kansas City, MO 64110
vsfgd@umkc.edu

Praveen R. Rao
University of Missouri-Kansas City
Kansas City, MO 64110
raopr@umkc.edu

ABSTRACT

In the last decade, we have witnessed a huge success of the peer-to-peer (P2P) computing model. This has lead to the development of many Internet-scale applications and systems that are used commercially. Recently, the problem of computing statistics over data in Internet-scale systems has received attention. In this paper, we discuss the problem of cardinality estimation of XPath queries over distributed XML data stored in an Internet-scale environment such as a P2P network. Such cardinality estimates are useful for XQuery optimization and statistical hypothesis testing in domains such as health informatics. We present a novel gossip algorithm called XGossip, which given an XPath query, estimates the number of XML documents that contain a match for the query. XGossip is designed to be scalable, decentralized, and robust to failures – properties that are desirable in a large-scale distributed system. XGossip employs a novel divide-and-conquer strategy for load balancing and reducing bandwidth consumption. We conduct theoretical analyses on the quality of cardinality estimates, message complexity, and bandwidth consumption. We present a preliminary performance evaluation on PlanetLab and discuss our ongoing work.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems—*Query Processing*

General Terms

Design

Keywords

XML, peer-to-peer, gossip algorithms, cardinality estimation

1. INTRODUCTION

We have witnessed a huge success of the P2P model of computing in the last decade. This has culminated in the development of Internet-scale applications such as Kazaa, BitTorrent, and Skype. P2P computing has also become popular in ecommerce and ebusiness and has lead to the development of many Internet-scale systems. Innovations in P2P computing, most notably the concept of Distributed Hash Table (DHT) (*e.g.*, Chord [65], Pastry [60], CAN [58], Tapestry [74], Kademlia [45]), has been embraced by key-value stores of production quality such as Dynamo [20], Cassandra [41], and Voldemort [5].

NetDB'11, June 12, 2011, Athens, Greece.


```
FOR  $gene IN service
("http://cabio.osu.edu/GeneService.wsdl")/Gene,
$go IN service
("http://cabio.osu.edu/GeneOntologyService.wsdl")/GeneOntology,
$microarray IN service
("http://caarray.duke.edu/caArrayService.wsdl")/Microarray
LET $subject := $microarray/experiment/subject
WHERE
  $go/term='vacuole' AND $gene/goAcc=$go/acc AND
  $gene/gbAcc=$microarray/data/geneId AND
  count($microarray/data[geneId=$gene/$gbAcc]/condition)>50
RETURN
<subject>
  <subjectId>{ $subject/lsid }</subjectId>
  <species>{ $subject/species }</species>
  <microarrayData>
    { $microarray/data }
  </microarrayData>
</subject>
```

Figure 1: An example XQuery

Meanwhile, the overwhelming success of XML, coupled with the growing popularity of P2P systems, has lead to research in indexing and query processing over XML data in a P2P environment [39, 26, 9, 19, 56]. One compelling use case for employing XML and P2P technologies is for large-scale sharing of biomedical and healthcare data. This is because of two reasons: First, it is suggested that scalable clinical data sharing systems can be built using a P2P architecture [64]. Second, HL7 version 3, an XML based standard for representation and interchange of healthcare data (*e.g.*, discharge summaries, lab reports), is becoming a standard for enabling semantic interoperability across systems [46].

The Cancer Biomedical Informatics Grid (caBIG) [22, 8], an initiative of the National Cancer Institute, exemplifies a real world data sharing platform for collaborative e-science. It has about 120 participating institutions across the US. The underlying network infrastructure of caBIG, called caGrid [61, 6], is a model-driven, service-oriented architecture. The data services are exposed in a well-documented and interoperable form. A federated query processor allows a user to query across multiple data sources. In fact, caGrid has implemented the XML Data Service interface for querying and retrieving data from XML databases [7, 50]. Currently, a user query is mapped to an appropriate XPath query and executed over the XML database. As caGrid is based on a service-oriented architecture, we believe a P2P architecture can enhance its scalability for large-scale sharing of distributed XML repositories.

Consider a distributed XQuery query supported by caBIG (Figure 1). The query *finds all the expression data where there are at least 50 conditions for genes found in the vacuole* [2]. It performs joins across data exposed by three data services, namely, Gene, GeneOntology, and Microarray. A more powerful query can be constructed wherein the locations of the documents in the network are not explicitly specified. Such a query can be processed by first locating relevant XML documents based on XPath expressions in the

query using any of the prior techniques [39, 26, 9, 19, 56] and then applying existing distributed XQuery processing techniques [59, 23, 73, 3].

Knowing the cardinality estimates of XPath expressions will aid the process of query optimization. We formulate our problem of interest as follows: *Given an XPath expression p, estimate the total number of XML documents distributed across a P2P network that contain a match for p with provable guarantee on the quality of the estimate.* The above estimate is useful in the following ways, though it does not provide the size of the result set of an XPath query. A query optimizer can select appropriate query plans based on how the relevant documents are distributed in the network. For instance, consider the query in Figure 1. If we know the cardinality estimates of XPath expressions such as `/Gene/goAcc`, `/Gene[term='vacuole']`, `/Microarray/data[geneId]/condition`, etc., a particular join ordering can be chosen. Another use case is for designing clinical studies: Researchers and medical professionals can quickly determine (using cardinality estimates) whether sufficient samples are available for a study, without having to query the network of distributed data sources.

Computing statistics over structured data stored in an Internet-scale environment has received attention [49, 51]. However, none has focused on the XML data model. In an Internet-scale environment, XML documents are distributed across a large number of participating peers and thus, collecting all the XML documents at any one peer to apply existing techniques for XML selectivity estimation [54, 24, 43], is prohibitively expensive. Furthermore, the network is dynamic with peers joining, leaving, and failing at any time. Under these circumstances, there are important design requirements for an effective cardinality estimation algorithm. First, the algorithm should be scalable and operate on a large number of peers. Second, it should be decentralized and not rely on any central authority. Third, it should consume minimum network bandwidth and be robust to the dynamism of the network. Fourth, it should provide provable guarantees on the quality of the estimates.

In this paper, we present a novel gossip algorithm called XGossip for cardinality estimation of XPath queries in an Internet-scale environment. Gossip (or epidemic) algorithms are attractive for large-scale distributed systems due to their simplicity, scalability, decentralized nature, ability to tolerate failures and the dynamism of the network, and ability to provide probabilistic guarantees. (One real-life use case of gossip algorithms is in Amazon S3 data centers for spreading server state information [1].) However, designing a gossip algorithm for cardinality estimation over XML data is not straightforward and introduces new challenges due to the very nature of the XML data model. While Ntarmos *et al.* [49] argue that gossip algorithms may not be suitable due to high bandwidth requirement and hop-count, we aim to show that it is indeed possible to design efficient gossip algorithms for statistics generation.

The remainder of the paper is organized as follows. Section 2 provides background and motivations; Section 3 presents our gossip algorithms, their theoretical analyses, and optimizations; Section 4 presents our preliminary performance evaluation; and finally, we discuss our ongoing work in Section 5.

2. BACKGROUND AND MOTIVATIONS

Statistics Computation over XML Data.

An XML document is typically modeled as an ordered, labeled tree. XPath [14] is a query language for navigating and selecting nodes in an XML document. XPath queries can be represented by *twig patterns*. A twig pattern is essentially a tree-like pattern with nodes representing elements, attributes, and values, and edges representing parent-child or ancestor-descendant relationships. XQuery [16] is a functional query language that subsumes XPath, and allows the creation of new XML content.

Accurate statistics over XML data is necessary for efficient XML query processing. The topic of cardinality/selectivity estimation over XML data in a local environment has been well studied (*e.g.*, Path trees/Markov tables [10], Correlated subpath tree [18], pH-join [70], StatiX [25], XPathLearner [42], Bloom Histogram [69], XSKETCH [54], IMAX [55], XCLUSTER [53], XSEED [72], lossy compression based synopsis [24], sampling based technique [43]).

Information Exchange and Aggregate Computation Via Gossip Algorithms.

Gossip algorithms provide a means for communication, computation, and information spreading [63, 62]. Prior work on gossip algorithms have mainly focused on information exchange (or rumor spreading) [28, 36, 52, 21, 27, 13] and computing aggregates (and separable functions) [37, 38, 17, 35, 47]. The essence of these algorithms lies in the exchange of information or aggregates between a pair of peers (picked randomly if the peers form a complete graph or among neighbors using a probability matrix assuming the peers form an arbitrary graph). It has been shown that after a provably finite number of rounds and a provably finite number of message exchanges, the information has reached all the peers or the aggregates (and separable functions) have converged to the true value.

Statistics Computation in a Distributed Environment.

In the area of information retrieval, document frequency estimation in a P2P network has received some attention [12, 48]. Recently, techniques for statistics generation in large-scale distributed networks have been developed for relational data (*e.g.*, aggregates and histograms [49], self-join size estimation [51]). A few gossip algorithms have been developed for statistics computation in large-scale networks (*e.g.*, computing frequent elements [40], distribution estimation [33, 31]).

Motivations.

To the best of our knowledge, there is no published work on XPath cardinality estimation in an Internet-scale environment. Such cardinality estimates are useful for distributed XQuery optimization as well as statistical hypothesis testing. One may wonder if a technique such as Distributed Hash Sketches [49], designed for structured data, can be adapted for XML data. This would require us to first map each XPath pattern that appears in the XML document to one dimensional space. However, enumerating all possible XPath patterns is computationally expensive and can result in a very large number of patterns due to the hierarchical nature of XML, the presence of many different element and attribute names in a document, and the presence of axes such as '//' (ancestor-descendant) in the queries.

Although gossip algorithms seem simple[1], dealing with XML introduces several challenges. First, if gossip is begun when a query is posed, like in Push-Sum [38], then one has to wait for a finite number of rounds before the cardinality estimate of an XPath expression is available. If gossip is continuously run in the background, then it is infeasible to gossip every unique XPath pattern due to the large number of distinct patterns – we expect a heterogeneous collection of XML documents in a P2P environment. Second, our algorithm should scale with increasing number of XML documents and peers in the network and yield effective load balancing. Third, network bandwidth is a critical resource in an Internet-scale environment. Our gossip algorithm should rely on exchang-

[1]The proofs and analyses, however, are mathematically rigorous.

ing a finite number of small sized messages – an essential property of a gossip algorithm [15].

3. OUR PROPOSED APPROACH

In this section, we present the Push-Sum protocol introduced by Kempe *et al.* [38]. We draw inspiration from Push-Sum and present a gossip algorithm called VanillaXGossip for XPath cardinality estimation. Finally, we employ a novel divide-and-conquer approach to overcome the limitations of VanillaXGossip and present an improved algorithm called XGossip. We also present the theoretical analysis of VanillaXGossip and XGossip. In the interest of space, proofs of theorems are available in a technical report [57].

3.1 System Model

We assume that the peers are connected using the Chord DHT overlay network, although other DHTs may be employed. A *successor of a key* in Chord is a peer mapped to a Chord ID that is the closest to the key (greater than or equal to) in the clockwise direction [65]. As in a typical P2P network, a peer owns a set of XML documents. A peer is said to "publish" those documents that it wishes to share with others in the network. The original documents reside at the publishing peer's end. We assume that our cardinality estimation algorithm (VanillaXGossip or XGossip) runs continuously in the background. At any time, a peer may receive a query. The peer looks at its local state or contacts a few other peers to compute the cardinality estimate.

3.2 Push-Sum Protocol

Suppose a P2P network has n peers and each peer p_i has a non-negative value x_i. Suppose we want to estimate the "average" *i.e.*, $\frac{1}{n}\sum_{i=1}^{n} x_i$. In the Push-Sum protocol [38], each peer maintains a sum s_t and weight w_t in round t. In round 0, each peer p_i sends $(x_i, 1)$ to itself. In any round $t > 0$, a peer computes the new sum (or weight) by adding the sum (or weights) of the messages it receives. It sends half of the sum and half of the weight to a randomly selected peer and the remaining half of the sum and weight to itself. In a particular round, the ratio of the current sum and weight is the estimated average. Push-Sum employs uniform gossip where a peer can contact any other peer during a gossip round – in terms of connectivity, the peers form a complete graph.

THEOREM 1 (PUSH-SUM PROTOCOL [38]). *Suppose there are n peers $p_1, \ldots, p_n$ in a network. Each peer p_i has a value $x_i \geq 0$. With at least probability $1 - \delta$, there is a round $r_o = O(log(n) + log(\frac{1}{\epsilon}) + log(\frac{1}{\delta}))$, such that in all rounds $r \geq r_o$, at peer p_i, the relative error of the estimate of the average value $\frac{1}{n}\sum_{i=1}^{n} x_i$ is at most ϵ.*

The proof is based on an important property of *mass conservation* [38]. What this means is that in any round, the average of the sums on all the peers is the true average, and the sum of weights on all peers is always n. To compute the "sum", *i.e.*, $\sum_{i=1}^{n} x_i$, Push-Sum is run with only one peer starting with a weight of 1 and the rest of the peers starting with a weight of 0 [38].

3.3 VanillaXGossip

We draw inspiration from Push-Sum to develop our gossip algorithms VanillaXGossip and XGossip. We select Push-Sum as the basis due to several reasons. Push-Sum relies on uniform gossip where peers form a complete graph. Because we assume that peers are connected through a DHT-based structured overlay network, any peer can contact any other peer (in $O(log(n))$ hops). In addition, Push-Sum is synchronous, but peers can follow their local clocks and the convergence holds as long as mass conservation is preserved [38]. (The analysis of a synchronous model is simpler than that of an asynchronous model [38]). In both VanillaXGossip and XGossip, we also compute "average" instead of "sum" because these algorithms run in the background and to guarantee, that only one peer will set its weight to 1 and the rest to 0, would require sophisticated distributed synchronization.

Next, we describe VanillaXGossip. Rather than gossiping XPath patterns in each XML document, a peer gossips the signature of an XML document, which is computed based on the method proposed by Rao and Moon [56]. A signature of a document is a product of irreducible polynomials assigned to its edges such that the document's structural properties and content are captured by the signature. A useful necessary condition of this signature representation is that *if a document contains a match for a query, then the query signature divides the document signature* [56]. Another benefit of these document signatures is that they are much smaller in size than the original XML documents [56].

A document signature can also be viewed as a multiset of irreducible polynomials. So we use the terms "multiset" and "signature" interchangeably in subsequent discussions. Suppose f_s denotes the frequency of a multiset/signature s. VanillaXGossip has two phases. In the initialization phase, each peer creates a sorted list of tuples called T using only its local state. Each tuple is of the form $(s, (f_s, w))$, where s is a signature of a XML document published by the peer and f_s is the number of XML documents having the same signature s and w is its weight. (Two different XML documents can have the same signature [56].) The weight is initialized to 1 like in Push-Sum. A tuple $(\perp, (0, 1))$ is also added to T, where $\perp$ is a special multiset that is the largest among all the possible multisets. The list is sorted by the first item of the tuple (*i.e.*, s), which serves as a primary key.

We use the following notations: T.begin(), T.end(), and T.next() allow us to iterate over T; For a tuple $c \in T$, $c.s$, $c.f$ and $c.w$ refer to individual elements in the tuple; $T[s]$ refers to a tuple with signature s.

REMARK 1. *A tuple with multiset $\perp$ plays the role of a placeholder in VanillaXGossip for multisets that are not yet known to a peer during a gossip round. This preserves the important property of mass conservation like in Push-Sum.*

After initialization, the peers begin the gossip phase and execute $RunGossip()$ in Algorithm 1. During a round, a peer first collects the lists received during the gossip round including the one that it sent to itself. It then merges the lists to update (f_s, w) of each tuple. After merging, the peer sends the list with halved frequencies and weights to a randomly selected peer, and sends another copy of that list to itself. (We select a random peer by picking a random Chord id and routing the message to the successor of that id.)

The merging process is unique to VanillaXGossip and is described by procedure $MergeLists()$ in Algorithm 1. Because the lists are sorted by the primary key, the merging phase requires linear time to complete. The minimum key/multiset is selected and its updated sum and weight are computed across all the received lists. If a list does not contain the key, then the sum and weight of $\perp$ are used. (The sum value for $\perp$ is always 0.) In any round, for a tuple $(s, (f, w))$ in the merged list T_m, an estimate of the average of the frequency of s in the network is $\frac{f}{w}$.

EXAMPLE 1. *Figure 2 shows an example of how the merging of three lists T_1, T_2, and T_3 is done using $MergeLists()$.*

THEOREM 2 (VANILLAXGOSSIP). *Given n peers $p_1, \ldots, p_n$, let a signature s be published by some m peers with frequencies*

Algorithm 1: Execution phase of VanillaXGossip

proc $RunGossip(p)$
1: Let $T_1, T_2, ..., T_R$ denote the lists received in the current round by peer p
2: $T_m \leftarrow MergeLists(T_1, T_2, ..., T_R)$
3: Send T_m to a random peer p_r and the participating peer p
end
proc $MergeLists(T_1, T_2, ..., T_R)$
4: $T_m \leftarrow \emptyset$
5: **for** *r=1 to R* **do**
6: $c_r \leftarrow T_r.begin()$
7: **while** *end of every list is not reached* **do**
8: $s_{min} \leftarrow min\{c_1.s, ..., c_R.s\}$
9: $sum_f \leftarrow 0$; $sum_w \leftarrow 0$;
10: **for** *r=1 to R* **do**
11: **if** $c_r.s = s_{min}$ **then**
12: $sum_f \leftarrow sum_f + c_r.f$
13: $sum_w \leftarrow sum_w + c_r.w$
14: $c_r \leftarrow T_r.next()$
else
15: $sum_f \leftarrow sum_f + T_r[\perp].f$
16: $sum_w \leftarrow sum_w + T_r[\perp].w$
17: Insert $(s_{min}, (\frac{sum_f}{2}, \frac{sum_w}{2}))$ into T_m
18: **return** T_m
end

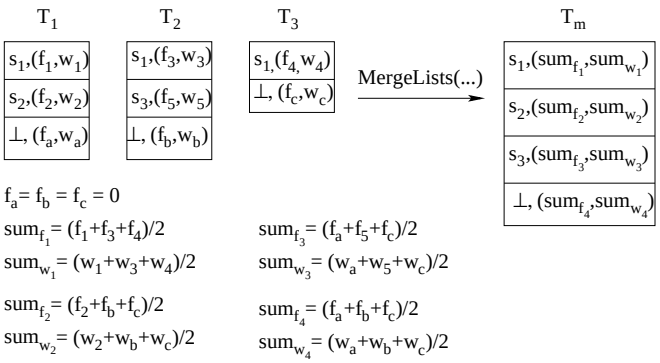


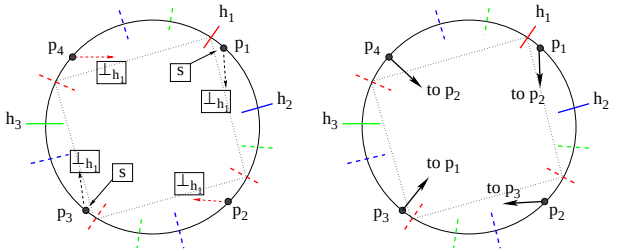
Figure 2: Merging of lists during VanillaXGossip

$f_1, \ldots, f_m$, where $m \leq n$. With at least probability $1-\delta$, there is a round $r_o = O(log(n) + log(\frac{1}{\epsilon}) + log(\frac{1}{\delta}))$, such that in all rounds $r \geq r_o$, at peer p_i, the relative error of the estimate of the average frequency of s, i.e., $\frac{1}{n}\sum_{i=1}^{m} f_i$, *is at most ϵ.*

Discussion. VanillaXGossip differs from Push-Sum in the following aspects. Push-Sum is initiated when a query is posed, and therefore, all peers are aware of the query and only gossip the aggregate value for that query. But VanillaXGossip runs continuously in the background and hence peers gossip the aggregate values of all the signatures that they are aware of. VanillaXGossip requires a placeholder to provide mass conservation and to guarantee convergence similar to Push-Sum.

3.4 XGossip: A Divide-and-Conquer Approach

One may notice that in VanillaXGossip, each peer eventually maintains a list with all distinct multisets in the network. This is inefficient in practice due to limited amount of main memory available at each peer. To overcome this limitation of VanillaXGossip, we employ a novel divide-and-conquer strategy using *locality sensitive hashing*. We call this improved algorithm XGossip. In XGossip, each peer will gossip only a provably finite fraction



(a) Initialization phase (b) Execution phase

Figure 3: Example for XGossip

of the distinct multisets in the network. The benefit of XGossip over VanillaXGossip is three-fold. Firstly, each peer will consume less memory. Secondly, each peer will consume less bandwidth. Thirdly, the convergence property of XGossip will be faster.

The concept of locality sensitive hashing (LSH), introduced by Indyk and Motwani [34], has been employed in many domains, including indexing high dimensional data and similarity searching [11, 44], similarity searching over web data [32] and in P2P networks [32, 30], ranges queries in P2P networks [29], and so forth. For similarity on sets based on Jaccard index, LSH on a set s can be performed as follows [32, 11]: Pick $k \times l$ random linear hash functions of the form $h(x) = (ax + b)\ mod\ p$, where p is a prime, and a and b are integers s.t. $0 < a < p$ and $0 \leq b < p$. Compute $g(s) = min(\{h(x)\})$ over all items in the set as the output hash value for s. It is established that given two sets s_1 and s_2, $Prob(g(s_1) = g(s_2)) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|}$. Each group of l hash values can be hashed again using another hash function $f(\cdot)$. Thus k hash values are output for a set.

In XGossip, we apply LSH on each document signature. We select $f(\cdot)$ to be the SHA-1 hash function. This way the hash values output by LSH for a signature are 160 bits and map onto the Chord DHT ring. We use the notation $\overline{h_s}$ to denote the vector of hash values output by LSH on s. We say that $\overline{h_s} = (h_{s1}, \ldots, h_{sk})$ defines k teams for s. Each hash value denotes the id of a team. Suppose Δ denotes the size of each team. Then for any team (with id) h_{si}, we calculate the Chord ids describing that team to be $\{h_{si}, h_{si} + 1 \times \frac{2^{160}}{\Delta}, h_{si} + 2 \times \frac{2^{160}}{\Delta}, \ldots, h_{si} + (\Delta - 1) \times \frac{2^{160}}{\Delta}\}$. (The addition operation will cause the result to wrap around the ring.)

The peers that are successors of the Chord ids defining a team, constitute the members of the team. These peers gossip only a subset of all the distinct signatures in the network. Also, they will only select a peer belonging to the team during a gossip round. Given two signatures with similarity p, the probability that there is at least one team that gossips both signatures is $1-(1-p^l)^k$. This is an important property of LSH that XGossip builds on. Thus similar signatures are gossiped by the same team with high probability. This increases the chances of finding all signatures that are required to estimate the cardinality of an XPath query. Furthermore, each peer gossips only a subset of the signatures in the network.

EXAMPLE 2. *Consider the DHT ring shown in Figure 3(a). Suppose $k = 3$ and a signature s produces hash values h_1 (red), h_2 (blue), and h_3 (green) after applying LSH. Each team is of size 4. The team h_1 is shown by a dotted square box. In this example, peers p_1, p_2, p_3, and p_4 are responsible for the team h_1.*

The divide-and-conquer approach in XGossip raises an interesting issue. Recall that in VanillaXGossip, a single special multiset $\perp$, required for mass conservation, is used by all peers in the network and is sent to a peer picked at random during a gossip round. But in XGossip, a peer cannot maintain one special multiset $\perp$.

Algorithm 2: Initialization phase of XGossip

global: T - tuple list
proc $InitGossipSend(p)$
1: Let T be initialized as in VanillaXGossip
2: **foreach** $c \in T$ *and* $c.s \neq \perp$ **do**
3: $\quad \overline{h_s} \leftarrow LSH(c.s)$
4: $\quad$ **foreach** $h_{si} \in \overline{h_s}$ **do**
5: $\quad\quad$ Create a team h_{si} and pick one id say q for the team at random and send $(c.s, (c.f, c.w))$ and h_{si} to the peer responsible for q according to the DHT protocol
end
proc $InitGossipReceive(p, (s, (f, w)), h)$
/* Keep one tuple list per team while receiving */
/* p is the peer that receives the message */
6: **if** T_h *does not exists* **then** create T_h
7: **if** s *is a regular multiset and* $T_h[s]$ *exists* **then**
8: $\quad$ Update the frequency in the tuple by adding f
9: **else if** s *is a regular multiset and* $T_h[s]$ *does not exist* **then**
10: $\quad$ Insert $(s, (f, w))$ into T_h
11: $\quad$ **if** $\perp_h$ *does not exist in* T_h **then**
12: $\quad\quad$ Insert $(\perp_h, (0, 1))$ into T_h; $InformTeam(p, \perp_h)$
13: **else if** $T_h[s]$ *does not exist* **then**
14: $\quad$ Insert $(s, (f, w))$ into T_h; $InformTeam(p, s)$
end
proc $InformTeam(p, \perp_{h_1})$
/* p is the peer executing $InitGossipReceive$ */
15: Suppose $h_2, \cdots, h_\Delta$ denote the other Chord ids for the team h_1
16: Let peer p be the successor of h_i
17: Send $(\perp_{h_1}, (0, 1))$ to the successor of $h_{(i \mod \Delta)+1}$
end

Rather a peer maintains one special multiset per team to which it belongs to. It sends that special multiset to peers that belong to that team during gossip. In fact, a peer may belong to more than one team. For a team h, its special multiset is denoted by $\perp_h$.

XGossip also has two phases. The first phase is the initialization phase and each peer runs $InitGossipSend()$ in Algorithm 2. Each peer creates the sorted list of tuples T based on the signatures of the XML documents it has published similar to the initialization phase of VanillaXGossip. For each tuple, the peer applies LSH on the tuple's signature and creates k teams. For each team, the peer randomly picks one of its Chord ids and sends the tuple to the successor of that id along with the team id.

When a peer receives a message during initialization via $InitGossipReceive()$ in Algorithm 2, it checks if the signature in the message is a regular multiset, *i.e.*it is a document signature. If so, it updates its list along with the special multiset for that team. But if a peer does not receive any message during initialization, then it does not know which team(s) it belongs to. *Then how can it initialize its special multiset?* We propose the following: When a peer receives a signature and the team id, it initializes its special multiset for that team. In addition, it contacts the next peer belonging to the team (in clock-wise direction along the DHT ring) and sends only the special multiset. A peer receiving a special multiset for a team, forwards it to its next peer of the team similarly. (The procedure $InformTeam()$ in Algorithm 2 does this task.) Note that the special multiset is only forwarded when a peer learns for the first time about a team it belongs to.

EXAMPLE 3. *Consider Figure 3(a). Suppose p_1 and p_3 receive a signature s during the initialization phase (solid black arrows). Each informs its next peer in the team with the special multiset (black dotted arrows). When p_2 and p_4 learn for the first time about team h_1, they forward the special multiset to their next peers in team h_1 (red dotted arrows).*

Algorithm 3: Execution phase of XGossip

proc $RunGossip(p)$
1: Let $T_1, T_2, ..., T_R$ denote the lists received in the current round by peer p
2: Group the lists based on their teams by checking their special multisets. Suppose each group is denoted by G_i.
3: **foreach** *group* G_i **do**
4: $\quad$ Merge the lists in G_i according to $MergeLists(\cdot)$
5: $\quad$ Let T_m denote the merged list
6: $\quad$ Compact T_m to save bandwidth
7: $\quad$ Let $h_1, \cdots, h_\Delta$ denote the Chord ids of the team
8: $\quad$ Pick an index $j \in [1, \Delta]$ at random such that p is not the successor of h_j
9: $\quad$ Send T_m to the peer that is the successor of h_j and to p
end

During the execution phase of XGossip, each peer groups the messages based on the teams from which they arrive. These are exactly the teams that the peer became aware of during initialization. For each group, $MergeLists()$ is invoked. Each merged list is then sent to a randomly picked peer belonging to the same team. The execution phase is described in Algorithm 3.

EXAMPLE 4. *Consider Figure 3(b) and the team h_1. Since peers p_1, p_2, p_3, and p_4 are the members of the team, they exchange messages belonging to that team during the gossip phase. A particular round may have peers exchange messages as shown with solid black arrows.*

THEOREM 3 (XGOSSIP). *Given n peers $p_1, \ldots, p_n$ in a network, let a signature s be published by some m peers with frequencies $f_1, \ldots, f_m$, where $m \leq n$. Suppose p_i belongs to a team that gossips s after applying locality sensitive hashing on s. Let Δ denote the size of the team. With at least probability $1 - \delta'$, there is a round $r_o = O(log(\Delta) + log(\frac{1}{\epsilon}) + log(\frac{1}{\delta'}))$, such that in all rounds $r \geq r_o$, at peer p_i, the relative error of the estimate of the average frequency of s, i.e., $\frac{1}{\Delta}\sum_{i=1}^{m} f_i$, is at most ϵ.*

3.5 Cardinality Estimation of XPath Queries

We describe how the cardinality of an XPath query, introduced in Section 1, is estimated. Suppose a query q is issued at peer p. In VanillaXGossip, the merged list T_m at p is searched to find every tuple $(s, (f_s, w))$, s.t. s is divisible by the query signature. (The divisibility test is a necessary condition for a query to have a match in the document [56].) Then $\sum \frac{f_s}{w}$ over all such tuples multiplied by the number of peers in the network is the desired cardinality estimate. [2] (We can assume that a good estimate of the number of peers in the network is known via Push-Sum.) By solely looking at the local state of p, the cardinality estimate can be computed.

Now let us consider XGossip. Let $\overline{h_q}$ denote the hash values after applying LSH on the signature of q. For each team h_{qi} ($1 \leq i \leq k$), p picks one member of that team at random and sends q's signature and h_{qi} to that team member. That team member scans its sorted tuple list for team h_{qi} and returns every $(s, (f_s, w))$ s.t. s is divisible by the signature of q. (In fact, a hash of the signature can be returned to save bandwidth.) Two or more tuples,

[2]We compute "average" instead of "sum" and therefore multiply by the number of peers.

Metric	VanillaXGossip	XGossip
Accuracy	$r\epsilon$	$r\epsilon$
Confidence	$(1-\delta)$	$(1-\delta)$
Convergence (# of rounds)	$O(log(n)+log(\frac{1}{\epsilon})$ $+log(\frac{1}{\delta}))$	$O(log(\Delta)+log(\frac{1}{\epsilon})$ $+log(\frac{\alpha}{\alpha+\delta-1}))$
Bandwidth	$O(nD)$	$O(log(n)kD\Delta)$
Messages	$O(n\ log(n))$	$O(\frac{log(n)}{n}kD\Delta log(\Delta))$

Table 1: Comparison of VanillaXGossip and XGossip

each returned by a different peer, may have identical signatures. In such a case, only one of the tuples should be retained and the rest can be discarded. Finally, $\sum \frac{f_s}{w}$ over the tuples received from k peers (after discarding the necessary tuples) multiplied by Δ is the desired cardinality estimate. Unlike VanillaXGossip, XGossip contacts k peers at query time and requires $O(k\ log(n))$ hops.

3.6 Analysis of VanillaXGossip and XGossip

Accuracy, Confidence, and Convergence.

To fairly compare VanillaXGossip and XGossip, we set the desired accuracy and confidence for cardinality estimation using both algorithms to $r\epsilon$ and $(1-\delta)$, assuming that there are r distinct document signatures that are divisible by a query signature. Let us denote these document signatures by R.

The number of rounds required by VanillaXGossip to achieve the desired accuracy and confidence is shown in Table 1. However, the confidence of XGossip is affected by an additional parameter due to the application of LSH. Suppose p_{min} denotes the minimum similarity between q's signature and a signature in R. Then the probability of finding all signatures in R by contacting k teams at query time is at least $\alpha = 1-(1-p_{min}^{l})^{k}$, where k and l are the parameters for LSH. Therefore, the net confidence of the estimate output by XGossip is $\alpha \cdot (1-\delta^{'})$, where $\delta^{'}$ is the value chosen in Theorem 3. To achieve the same confidence as VanillaXGossip, the following equation should hold: $(1-\delta) = \alpha \cdot (1-\delta^{'})$. Therefore, $\delta^{'} = \frac{\alpha+\delta-1}{\alpha}$. While in VanillaXGossip, there is one team of size n, in XGossip, each team is of size Δ. Using $\delta^{'}$ and Δ, the number of rounds required by XGossip is shown in Table 1.

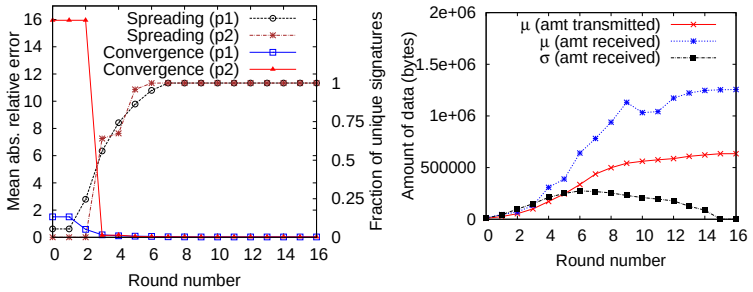
Message Complexity and Bandwidth Consumption.

In VanillaXGossip, eventually all peers gossip every unique signature in the network. Suppose D denotes the number of unique signatures. Therefore, the worst case bandwidth consumed by each peer is $O(D)$. The message complexity of VanillaXGossip is similar to Push-Sum and is $O(n\ log(n))$.

To analyze, XGossip, we will first discuss the property of consistent hashing in Chord [65]. Suppose there are n peers and K keys. Chord guarantees that with high probability each peer receives at most $\frac{(1+\rho)K}{n}$ keys, where ρ is bound by $O(log(n))$ [65]. In XGossip, we have at most kD team ids/Chord ids. So each peer becomes the successor for at most $O(\frac{log(n)}{n}kD)$ teams. Since there are Δ members per team, we have at most $O(\frac{log(n)}{n}kD\Delta)$ distinct signatures per team, which denotes the worst case bandwidth consumption. Given that each team in XGossip exchanges $O(\Delta log(\Delta))$ messages, the overall message complexity is shown in Table 1.

3.7 Churn and Failures

Kempe *et al.* have discussed a few failure scenarios in Push-Sum [38]. When a message is not delivered successfully to a peer during a gossip round, then the sending peer will consume the message as if the message were never sent and update its local sum and weight to conserve mass. If a peer decides to leave the network, it should send its sum and weight to another peer. Similar solution is adopted by VanillaXGossip and XGossip. Another issue arises in XGossip. Suppose the successor of any of the Chord ids defining a team changes, then a message sent by a peer of the team may be received by a peer who initially did not belong to the team. Then the receiving peer should reject the message so that the sending peer can consume the message to conserve mass.



(a) Convergence and Spreading (b) Bandwidth consumption

Figure 4: Results for VanillaXGossip

4. PRELIMINARY EVALUATION

We report a preliminary evaluation of VanillaXGossip. We have conducted the evaluation on PlanetLab [4] using 100 nodes. The algorithms were implemented in C++ using the Chord package [66].

Using 38 DTDs published on the Internet [67, 68, 71], we generated synthetic datasets using IBM's synthetic data generator. The average number of documents per DTD was 2,457 and a total of 90,936 documents was used. The average size of a document signature was 65 bytes. We randomly picked 4 peers for a DTD and distributed the documents for that DTD equally across those peers. We ran VanillaXGossip with 100 peers after the DHT routing tables had stabilized.

Figure 4(a) shows the convergence of VanillaXGossip for two randomly selected peers $p1$ and $p2$. Beyond round 6, the mean of the absolute relative error of the average frequency estimate on a subset of signatures stayed below 8%. We also plot the speed of information spreading by measuring the fraction of unique signatures learnt by a peer in the network. Figure 4(a) shows that by round 8, the two peers learnt about all the unique signatures in the network. Figure 4(b) shows the mean (μ) amount of data transmitted and received by a peer during a gossip round. It also shows the std. deviation (σ) of the amount of data received by a peer. Note that σ remained at 0 starting from round 15. This is because all the peers learnt about all the unique signatures in the network by round 15. We expect XGossip to converge faster and consume less bandwidth than VanillaXGossip.

5. ONGOING WORK

The application of locality sensitive hashing enables similar signatures to be gossiped by the same team with high probability. Thus it is likely that the signatures sent to a particular peer during a gossip round have high similarity. We are developing a scheme to compact similar signatures exchanged by a team and thereby reduce the bandwidth consumption. We are also investigating how XPath queries with value predicates be handled. We are currently evaluating XGossip on PlanetLab and a local area network and comparing it with VanillaXGossip.

Acknowledgements

Praveen Rao would like to acknowledge the support from University of Missouri Research Board and IBM Smarter Planet Faculty Innovation Award.

6. REFERENCES

[1] Amazon S3 Availability Event: July 20, 2008. http://status.aws.amazon.com/s3-20080720.html.

[2] caBIG Architecture Workspace: Common Query Language SIG, Summary and Initial Recommendations. https://cabig.nci.nih.gov/archive/SIGs/Common%20Query%20Language/ArchWSQuery%20SIG_Recomd_F2F_%20March05.ppt.

[3] DXQP - Distributed XQuery Processor. http://sig.biostr.washington.edu/projects/dxqp/.

[4] PlanetLab. http://www.planet-lab.org.

[5] Project Voldemort. http://project-voldemort.com/.

[6] The caGrid Portal. http://cagrid-portal.nci.nih.gov/web/guest.

[7] The caGrid xService. https://web.cci.emory.edu/confluence/display/xmlds/caGrid+xService.

[8] The Cancer Biomedical Informatics Grid. https://cabig.nci.nih.gov/.

[9] S. Abiteboul, I. Manolescu, N. Polyzotis, N. Preda, and C. Sun. XML Processing in DHT Networks. In *Proc. of the 24th IEEE Intl. Conference on Data Engineering*, Cancun, Mexico, Apr. 2008.

[10] A. Aboulnaga, A. R. Alameldeen, and J. F. Naughton. Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. In *Proc. of the 27th International Conference on Very Large Data Bases*, pages 591–600, San Francisco, CA, 2001.

[11] M. Bawa, T. Condie, and P. Ganesan. LSH Forest: Self-tuning Indexes for Similarity Search. In *Proceedings of the 14th International Conference on World Wide Web*, pages 651–660, 2005.

[12] M. Bender, S. Michel, P. Triantafillou, and G. Weikum. Global Document Frequency Estimation in Peer-to-Peer Web Search. In *Proceedings of WebDB*, 2006.

[13] N. Berger, C. Borgs, J. T. Chayes, and A. Saberi. On the Spread of Viruses on the Internet. In *Proc. of the 16th Annual ACM-SIAM Symposium on Discrete algorithms*, pages 301–310, 2005.

[14] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Simeon. XML path language (XPath) 2.0 W3C working draft 16. Technical Report WD-xpath20-20020816, World Wide Web Consortium, Aug. 2002.

[15] K. Birman. The Promise, and Limitations, of Gossip Protocols. *Operating Systems Review*, 41(5):8–13, 2007.

[16] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML Query Language W3C working draft 16. Technical Report WD-xquery-20020816, World Wide Web Consortium, Aug. 2002.

[17] S. P. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip Algorithms: Design, Analysis and Applications. In *INFOCOM 2005*, pages 1653–1664, 2005.

[18] Z. Chen, H. V. Jagadish, F. Korn, N. Koudas, S. Muthukrishnan, R. T. Ng, and D. Srivastava. Counting Twig Matches in a Tree. In *Proc. of the 17th International Conference on Data Engineering*, pages 595–604, Heidelberg, Germany, 2001.

[19] E. Curtmola, A. Deutsch, D. Logothetis, K. K. Ramakrishnan, D. Srivastava, and K. Yocum. XTreeNet: Democratic Community Search. In *Proc. of the 34th VLDB Conference*, pages 1448–1451, Auckland, New Zealand, 2008.

[20] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-Value Store. In *Proc. of 21st ACM SIGOPS Symposium on Operating Systems Principles*, pages 205–220, Stevenson, Washington, 2007.

[21] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. In *Proc. of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12, 1987.

[22] D. Fenstermacher, C. Street, T. McSherry, V. Nayak, C. Overby, and M. Feldman. The Cancer Biomedical Informatics Grid (caBIG). In *Proc. of IEEE Engineering in Medicine and Biology Society*, pages 743–746, Shanghai, China, 2005.

[23] M. Fernandez, T. Jim, K. Morton, N. Onose, and J. Simeon. DXQ: A Distributed XQuery Scripting Language. In *4th International Workshop on XQuery Implementation Experience and Perspectives*, 2007.

[24] D. K. Fisher and S. Maneth. Structural Selectivity Estimation for XML Documents. In *Proc. of the 23th IEEE Intl. Conference on Data Engineering*, pages 626–635, Istanbul, Turkey, 2007.

[25] J. Freire, J. R. Harista, M. Ramanath, P. Roy, and J. Simone. StatiX: Making XML Count. In *Proc. of the 2002 ACM-SIGMOD Conference*, Madison, Wisconsin, June 2002.

[26] L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt. Locating Data Sources in Large Distributed Systems. In *Proc. of the 29th VLDB Conference*, Berlin, 2003.

[27] A. Ganesh, L. Massoulie, and D. Towsley. The Effect of Network Topology on the Spread of Epidemics. In *INFOCOM 2005*, pages 1455–1466, 2005.

[28] C. Georgiou, S. Gilbert, R. Guerraoui, and D. Kowalski. On the Complexity of Asynchronous Gossip. In *Proc. of the 27th ACM Symposium on Principles of Distributed Computing*, pages 135–144, Toronto, Canada, 2008.

[29] A. Gupta, D. Agrawal, and A. E. Abbadi. Approximate Range Selection Queries in Peer-to-Peer Systems. In *Conference on Innovative Data Systems Research (CIDR)*, 2003.

[30] P. Haghani, S. Michel, and K. Aberer. Distributed Similarity Search in High Dimensions using Locality Sensitive Hashing. In *Proc. of the 12th International Conference on Extending Database Technology*, pages 744–755, 2009.

[31] M. Haridasan and R. van Renesse. Gossip-Based Distribution Estimation in Peer-to-Peer Networks. In *Proc. of the 7th International Conference on Peer-to-Peer Systems*, Tampa Bay, Florida, 2008.

[32] T. H. Haveliwala, A. Gionis, D. Klein, and P. Indyk. Evaluating Strategies for Similarity Search on the Web. In *Proc. of the 11th international conference on World Wide Web*, pages 432–442, Honolulu, Hawaii, 2002.

[33] Y. Hu, J. G. Lou, H. Chen, and J. Li. Distributed Density Estimation Using Non-parametric Statistics. In *Proc. of 27th International Conference on Distributed Computing Systems (ICDCS)*, pages 28–36, June 2007.

[34] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proc. of the 13th ACM Symposium on Theory of Computing*, pages 604–613, Dallas, Texas, 1998.

[35] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-Based Aggregation in Large Dynamic Networks. *ACM Transactions on Computer Systems*, 23:219–252, August 2005.

[36] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized Rumor Spreading. In *IEEE Symposium on Foundations of Computer Science*, pages 565–574, 2000.

[37] S. Kashyap, S. Deb, K. Naidu, R. Rastogi, and A. Srinivasan. Efficient Gossip-Based Aggregate Computation. In *Proc. of the 35th ACM Principles of Database Systems*, Chicago, IL, 2006.

[38] D. Kempe, A. Dobra, and J. Gehrke. Gossip-Based Computation of Aggregate Information. In *Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science*, Cambridge, MA, Oct 2003.

[39] G. Koloniari and E. Pitoura. Peer-to-Peer Management of XML Data: Issues and Research Challenges. *SIGMOD Record*, 34(2):6–17, June 2005.

[40] B. Lahiri and S. Tirthapura. Identifying Frequent Items in a Network using Gossip. *Journal of Parallel and Distributed Computing*, 70(12):1241–1253, 2010.

[41] A. Lakshman and P. Malik. Cassandra: A Structured Storage System on a P2P network. In *Proc. of the 2008 ACM-SIGMOD Conference*, Vancouver, Canada, 2008.

[42] L. Lim, M. Wang, S. Padmanabhan, J. S. Vitter, and R. Parr. XPathLearner: An On-line Self-Tuning Markov Histogram for XML Path Selectivity Estimation. In *Proc. of the 28th International Conference on Very Large Data Bases*, pages 442–453, Hong Kong, China, 2002.

[43] C. Luo, Z. Jiang, W.-C. Hou, F. Yu, and Q. Zhu. A Sampling Approach for XML Query Selectivity Estimation. In *Proc. of the 12th International Conference on Extending Database Technology*, pages 335–344, Saint Petersburg, Russia, 2009.

[44] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-Probe LSH: Efficient Indexing for High-dimensional Similarity Search. In *Proc. of the 33rd VLDB Conference*, pages 950–961, Vienna,

Austria, 2007.

[45] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proc. of 1st International Workshop on Peer-to-Peer Systems*, pages 53–65, London, 2002.

[46] C. N. Mead. Data Interchange Standards in Healthcare IT – Computable Semantic Interoperability: Now Possible but Still Difficult, Do We Really Need a Better Mousetrap? *Journal of Healthcare Information Management*, 20(1):71–78, 2006.

[47] D. Mosk-Aoyama and D. Shah. Fast Distributed Algorithms for Computing Separable Functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, July 2008.

[48] R. Neumayer, C. Doulkeridis, and K. Nørvåg. A Hybrid Approach for Estimating Document Frequencies in Unstructured P2P Networks. *Information Systems*, 36(3):579–595, 2011.

[49] N. Ntarmos, P. Triantafillou, and G. Weikum. Statistical Structures for Internet-Scale Data Management. *The VLDB Journal*, 18(6):1279–1312, 2009.

[50] T. C. Pan, J. D. Permar, A. Sharma, D. W. Ervin, T. M. Kurc, and J. H. Saltz. Virtualizing XML Resources As caGrid Data Services. In *Proc. of AMIA Translational Bioinformatics Summit*, San Francisco, CA, 2009.

[51] T. Pitoura and P. Triantafillou. Self-Join Size Estimation in Large-scale Distributed Data Systems. In *Proc. of the 24th IEEE Intl. Conference on Data Engineering*, Cancun, Mexico, April 2008.

[52] B. Pittel. On Spreading a Rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, 1987.

[53] N. Polyzotis and M. Garofalakis. XCluster Synopses for Structured XML Content. In *Proc. of the 22th IEEE Intl. Conference on Data Engineering*, page 63, Atlanta, GA, Apr. 2006.

[54] N. Polyzotis, M. Garofalakis, and Y. Ioannidis. Selectivity Estimation for XML Twigs. In *Proc. of the 20th IEEE Intl. Conference on Data Engineering*, Boston, MA, March 2004.

[55] M. Ramanath, L. Zhang, J. Freire, and J. R. Haritsa. IMAX: Incremental Maintenance of Schema-Based XML Statistics. In *Proc. of the 21st International Conference on Data Engineering*, pages 273–284, Tokyo, Japan, 2005.

[56] P. Rao and B. Moon. Locating XML Documents in a Peer-to-Peer Network using Distributed Hash Tables. *IEEE Transactions on Knowledge and Data Engineering*, 21(12):1737–1752, December 2009.

[57] P. Rao and V. Slavov. Towards Internet-Scale Cardinality Estimation of XPath Queries over Distributed XML Data. Technical Report TR-DB-2011-01, University of Missouri-Kansas City, June 2011. http://r.faculty.umkc.edu/raopr/TR-DB-2011-01.pdf.

[58] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *Proc. of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, San Diego, CA, 2001.

[59] C. Re, J. Brinkley, K. Hinshaw, and D. Suciu. Distributed XQuery. In *Proc. of the Workshop on Information Integration on the Web*, pages 116–121, 2004.

[60] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proc. of the IFIP/ACM Intl. Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, Nov. 2001.

[61] J. Saltz, S. Oster, S. Hastings, S. Langella, T. Kurc, W. Sanchez, M. Kher, A. Manisundaram, K. Shanbhag, and P. Covitz. caGrid: Design and Implementation of the Core Architecture of the Cancer Biomedical Informatics Grid . *Bioinformatics*, 22(15):1910–1916, 2006.

[62] D. Shah. Gossip Algorithms. *Foundations and Trends in Networking*, 3(1):1–125, 2009.

[63] D. Shah. Network Gossip Algorithms. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3673–3676, 2009.

[64] W. W. Stead and H. S. Lin. Computational Technology for Effective Health Care: Immediate Steps and Strategic Directions. *The National Academies Press, Washington D.C.*, 2009.

[65] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. of the 2001 ACM-SIGCOMM Conference*, pages 149–160, San Diego, CA, Aug. 2001.

[66] The Chord/DHash Project. Available from http://pdos.csail.mit.edu/chord/.

[67] The Niagara Project. http://www.cs.wisc.edu/niagara/.

[68] UW XML Repository. www.cs.washington.edu/research/xmldatasets.

[69] W. Wang, H. Jiang, H. Lu, and J. X. Yu. Bloom Histogram: Path Selectivity Estimation for XML Data with Updates. In *Proc. of the 30th VLDB Conference*, pages 240–251, Toronto, Canada, 2004.

[70] Y. Wu, J. M. Patel, and H. V. Jagadish. Estimating Answer Sizes for XML Queries. In *Proc. of the 8th International Conference on Extending Database Technology*, pages 590–608, Prague, 2002.

[71] XML.org. Available from http://www.xml.org/xml.

[72] N. Zhang, M. T. Ozsu, A. Aboulnaga, and I. F. Ilyas. XSEED: Accurate and Fast Cardinality Estimation for XPath Queries. In *Proc. of the 22th IEEE Intl. Conference on Data Engineering*, page 61, Atlanta, GA, 2006.

[73] Y. Zhang and P. A. Boncz. XRPC: Interoperable and Efficient Distributed XQuery. In *Proc. of the International Conference on Very Large Data Bases (VLDB)*, Vienna, Austria, September 2007.

[74] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, Jan. 2004.