A Tool for Internet-Scale Cardinality Estimation of XPath Queries over Distributed Semistructured Data

Vasil Slavov, Anas Katib, Praveen Rao

University of Missouri-Kansas City, Kansas City, MO 64110.
vgslavov@mail.umkc.edu,anas.katib@mail.umkc.edu,raopr@umkc.edu

***Abstract*—We present a novel tool called XGossip for Internet-scale cardinality estimation of XPath queries over distributed XML data. XGossip relies on the principle of gossip: It is scalable, decentralized, and can cope with network churn and failures. It employs a novel divide-and-conquer strategy for load balancing and reducing the overall network bandwidth consumption. It has a strong theoretical underpinning and provides provable guarantees on the accuracy of cardinality estimates, the number of messages exchanged, and the total bandwidth usage. In this demonstration, users will experience three engaging scenarios: In the first scenario, they can set up, configure, and deploy XGossip on Amazon Elastic Compute Cloud (EC2). In the second scenario, they can execute XGossip, pose XPath queries, observe in real-time the convergence speed of XGossip, the accuracy of cardinality estimates, the bandwidth usage, and the number of messages exchanged. In the third scenario, they can introduce network churn and failures during the execution of XGossip and observe how these impact the behavior of XGossip.**

I. Introduction

Today, there is a growing need for large-scale, federated data sharing systems in the fields of biomedicine and healthcare. This is because *data sharing and collaboration* among institutions can accelerate discoveries and foster unprecedented innovations in areas such as cancer diagnosis and treatment. One striking example is the caBIG [1], an initiative of the National Cancer Institute for collaborative e-scene: The caBIG is a federated data sharing system with about 120 institutions.

Meanwhile, it is becoming evident that the overwhelming success of peer-to-peer (P2P) and XML technologies can advance the state-of-the-art due to two reasons: First, it has been suggested that a P2P model should be adopted for designing scalable, federated data sharing systems [2]. Second, the HL7 Version 3 standard, an XML-based standard for representation and interchange of healthcare data, is becoming increasingly important for complying with the *meaningful use* criteria of the HITECH Act (2009) [3] and achieving semantic interoperability across data sharing systems [4].

In this work, we focus on the problem of cardinality estimation that arises in Internet-scale, federated data sharing systems. Selectivity (or cardinality) estimation is a classical task dealt by query optimizers, for example, to decide the best join order for a query. While the problem of XML selectivity estimation in a local/centralized environment has been well-studied (*e.g.*, path trees and Markov tables [5], correlated subpath trees [6], StatiX [7], Bloom Histogram [8], XSKETCH [9], XSEED [10], lossy compression [11], sampling-based approach [12]), nobody has attempted to address this issue in an Internet-scale environment. For this purpose, we have developed a novel tool called XGossip, which operates as follows: Given an XPath expression (or query) q, XGossip estimates the total number of XML documents in the network that contain a match for q with a provable guarantee on the accuracy of the estimate. XGossip differs from prior work on XML selectivity estimation in the sense that it estimates the number of matching XML documents (in the network) rather than the size of the result set of an XPath expression.

XGossip's cardinality estimate is useful in many ways: It is useful for optimizing a distributed XQuery query by deciding the best join order based on the number of matching documents (in the network) for different XPath expressions in the query. It is also useful for developing IR-style relevance ranking schemes. Another use case is in the design of a clinical study, to quickly compute if sufficient samples are available for the study, without actually querying the network of distributed data sources. Today, there are tools such as HERON [13] that allow an investigator to issue cohort discovery queries to know the number of subjects available for a clinical trial.

In recent years, a few techniques have been proposed for computing Internet-scale statistics over structured data [14], [15], [16]. One may wonder if a technique like Distributed Hash Sketches [16] can be adapted to handle XML. This seems possible by first mapping each XPath pattern that appears in an XML document onto a one-dimensional space. Unfortunately, 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 axis such as '//' in the queries.

Gossip algorithms are popular in large-scale distributed systems due to their scalability and fault-tolerance [17], [18]. For cardinality estimation, designing a gossip algorithm that computes an aggregate like *average* (*e.g.*, Push-Sum [19]) seems to be a viable choice. In such an algorithm, pairs of peers exchange aggregates in a round, and after a provably finite number of rounds and a provably finite number of message exchanges, the aggregates converge to the true value. The XML data model, however, introduces new challenges: First, we should quickly compute the cardinality estimate of a query once it has been posed rather than wait for a finite number of gossip rounds after the query is known to peers like in Push-Sum. Then we must continuously gossip in the background, but we simply cannot gossip every XPath

pattern due to the potentially large number – we expect a heterogeneous collection of XML documents in a distributed environment. Second, our algorithm should scale with increasing number of XML documents and peers in the network and yield effective load balancing. Third, our algorithm should rely on exchanging a finite number of small messages to minimize the network bandwidth usage.

In the next section, we present the design of XGossip and highlight its novel features to overcome the aforementioned challenges. We refer the reader to a journal article [20] for a complete description of XGossip including our theoretical analysis and performance evaluation.

II. The Design of XGossip

A. System Model

Figure 1(a) illustrates the system model. Peers are assumed to be connected using a DHT overlay network (*e.g.*, Chord [21]). A peer owns a set of XML documents. It 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. Each peer executes an instance of XGossip and continuously gossips with other peers. Peers communicate with each other using the DHT's routing protocol. At any time, a peer can compute the cardinality estimate of an XPath query and will contact a few other peers during this process.

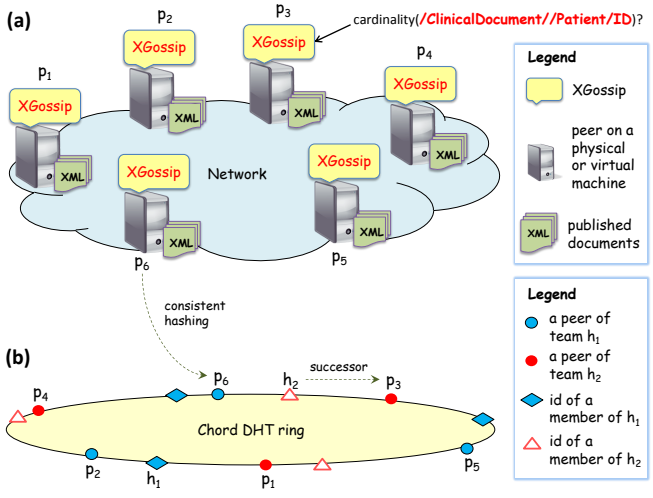


Fig. 1. System model

B. Novel Aspects of XGossip

XGossip is based on the principle of gossip. It is scalable, decentralized, and can cope with network churn and failures; these are properties desirable in a large-scale distributed system. It draws inspiration from Push-Sum [19] and has many novel aspects to efficiently compute cardinality estimates of XPath queries over distributed XML data sources.

In XGossip, peers exchange concise summaries (or signatures) of XML documents instead of XPath patterns. A document is represented by its signature [22], which is essentially a product of irreducible polynomials carefully assigned to a summarized representation of the document, and captures the document's structure and content. A query is also mapped into its signature. A useful necessary condition is that if a document contains a match for a query, the query signature divides the document signature [22]. Also the size of a document signature is much smaller that the document itself [22].

For effective load balancing and reducing the network bandwidth usage, XGossip employs: (i) a divide-and-conquer strategy by applying locality-sensitive hashing (LSH) [23], [24], and (ii) a compression scheme for compacting document signatures in gossip messages. Suppose there are n peers in the network. They are organized into teams of size Δ, where $\Delta << n$. A peer can be a member of multiple teams. The peers in a team gossip only a fraction of the distinct document signatures in the network. Given a signature, which can be viewed as a multiset, the LSH function in XGossip outputs k hash values, where each hash value is a 160-bit Chord id. Each hash value identifies a team and is the id of a team member. The ids of the remaining $\Delta - 1$ team members are computed by equally dividing the DHT address space. A peer that is a successor (as defined by Chord [21]) of an id of a team is a member of the team.

By virtue of LSH, given two signatures s_1 and s_2, the probability that at least one of their team ids is identical is $1-(1-p^l)^k$, where p is the Jaccard index of s_1 and s_2 and l is the number of hash functions used by the LSH function to generate each id. Thus, similar signatures are gossiped by the same team with high probability. This increases the chances of finding all the signatures that are required to estimate the cardinality of an XPath query. Furthermore, the compression scheme in XGossip is designed to take advantage of similar signatures in a gossip message. The above design choices lead to faster convergence of a cardinality estimate to its true value and smaller gossip messages.

Example 1: Suppose the peers in Figure 1(a) are mapped to the DHT address space as shown by the red and blue dots in Figure 1(b). Let $k = 2$ and $\Delta = 3$. For simplicity, suppose there is one signature s in the network. Let h_1 and h_2 denote the output of the LSH function on s. The blue diamonds denote the ids of members of team h_1, which is managed by peers p_2, p_5, and p_6 (blue dots). The red triangles denote the ids of members of team h_2, which is managed by peers p_1, p_3, and p_4 (red dots). Both teams will gossip s.

Finally, XGossip is based on a strong theoretical underpinning, builds on the convergence speed of Push-Sum and its property on mass conservation[1], and the properties of LSH, and provides provable guarantees on the accuracy of cardinality estimates, the number of messages exchanged, and the total bandwidth usage.

C. Key Components of XGossip

XGossip runs in two phases: (a) the initialization phase for creating teams and initializing the local state on each peer, and

[1]Given n peers, each peer p_i starts with a sum and weight pair $(x_i, 1)$, sends half the sum and weight to a randomly selected peer, keeps the remaining halves, and updates it in each round . The convergence of Push-Sum is based on the property of mass conservation, which is stated as follows: In any round, the average of the sums on all the peers is $\frac{1}{n}\sum_{i=1}^{n} x_i$ and the sum of the weights on all the peers is always n.

(b) the execution phase when peers in every team gossip.

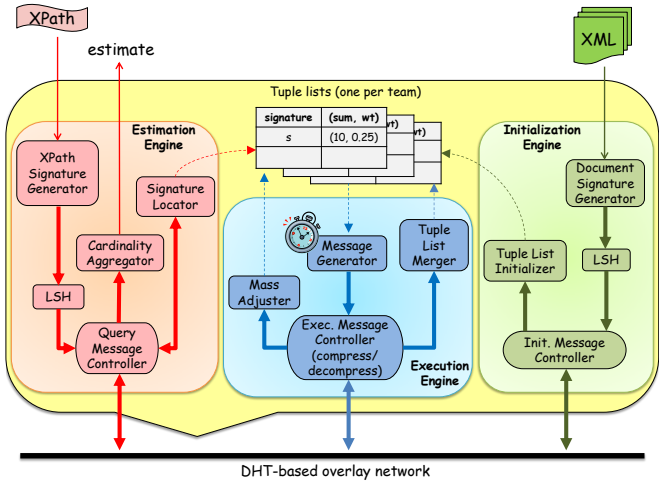


Fig. 2. Architecture of XGossip

Figure 2 illustrates the key components of XGossip. On a peer, XGossip maintains a collection of tuple lists, one for each team that the peer belongs to. A tuple list contains tuples of the form $(s, (f, w))$, where s is a document signature and (f, w) is a sum and weight pair like in Push-Sum [19]. The list is sorted using the first item of each tuple, which serves as a primary key. A tuple $(\perp, (0, w'))$ is also part of the list, where $\perp$ is a special multiset and serves as the largest key. (The sum is always 0 and the weight is initialized to 1.) This tuple is unique to XGossip and plays the role of a placeholder for signatures that the peer has not yet seen during gossip. As a result, the convergence speed of XGossip can be proven using the property of mass conservation [20].

The Initialization Engine operates during the initialization phase. It computes the signatures of the published documents. For each unique signature s, it creates a tuple $(s, (f_s, 1))$, where f_s is the number of documents having the same signature s. (Note that two documents can have the same signature [22].) Once the teams ids for s are computed using LSH, the tuple $(s, (f_s, 1))$ is sent to a randomly selected member of each team through the Init. Message Controller. The Init. Message Controller may receive messages from other peers. Based on the type of message, the Tuple List Initializer will either create a tuple list for a newly formed team and update it, or update the tuple list of an existing team. When a team is newly formed, the Init. Message Controller will send a special message to a team member to inform about it.

The Execution Engine operates during the execution phase and follows its own local clock. (The time interval between successive gossip rounds, however, is the same for all peers.) The Message Generator periodically creates a compressed message for each team using its tuple list and instructs the Exec. Message Controller to transmit the message to a randomly selected team member. The Controller may receive gossip messages from other peers during a round. These messages are decompressed and forwarded to the Tuple List Merger, which groups the received messages based on their teams and then merges the sums and weights in the messages for each team with the tuple list for that team.

Example 2: Suppose peer p_6 (in Figure 1(b)) holds the tuple list $T_1 = [(s_1, (f_1, w_1)), (\perp, (f_a, w_a))]$ for team h_1 during a gossip round. Suppose it receives the tuple list $T_2 = [(s_1, (f_2, w_2)), (s_2, (f_3, w_3)), (\perp, (f_b, w_b))]$ from a team member during the round. After merging, $T_1 = [(s_1, (\frac{f_1+f_2}{2}, \frac{w_1+w_2}{2})), (s_2, (\frac{f_3+f_a}{2}, \frac{w_3+w_a}{2})), (\perp, (\frac{f_a+f_b}{2}, \frac{w_a+w_b}{2}))]$. For a signature (or multiset) that appears in both lists, the corresponding sums and weights are merged. Otherwise, the sum and weight of $\perp$ from the list missing the signature is used. (Note that $f_a = 0$ and $f_b = 0$ always.) The merging step requires linear time as the tuple lists are sorted.

A few scenarios such as churn, network partitioning, and failures can disturb mass conservation in XGossip. The Mass Adjuster takes steps to preserve mass conservation whenever possible, by updating the sums and weights in the tuple lists appropriately when a message is not delivered to a peer, reaches a peer not belonging to the same team, or is delivered to a peer that just joined the network but did not participate in the initialization phase. (In the latter two cases, the receiving peer rejects the message.) If a gossiping peer fails abruptly, then mass conservation is not preserved (like in Push-Sum). This can impact the quality of cardinality estimates depending on how much disturbance was caused to mass conservation.

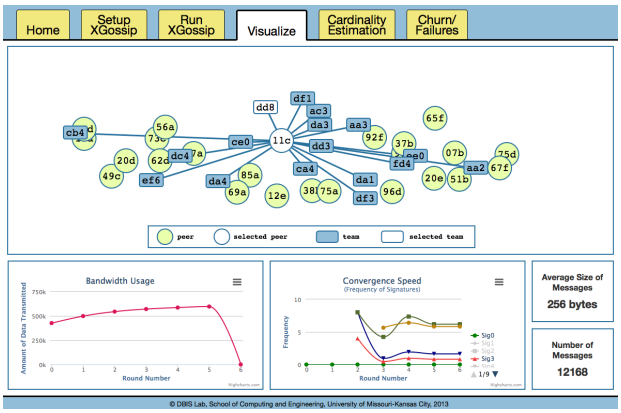
The Estimation Engine handles the task of cardinality estimation. When an XPath query is posed at a peer, the query initiator, LSH is applied on the query signature to identify the candidate teams that may gossip the document signatures that are divisible by (or supersets of) the query signature. For each candidate team, the Query Message Controller sends the query signature and team id to a randomly selected member of that team. (Instead of the query signature, a proxy signature generated from a DTD can be used to more accurately identify the candidate teams [20].) The Signature Locator in the Estimation Engine of a receiving peer locates the tuple list for the team id in the message. If present, it scans the tuple list and identifies every tuple that contains a document signature that is a superset of the query signature. The Query Message Controller of the receiving peer returns all the matching tuples to the Estimation Engine of the query initiator. Otherwise, no response is sent to the query initiator.

The Cardinality Aggregator of the query initiator collects all the received tuples. Two or more tuples, each returned by a different peer, may have identical signatures. When this happens, the Cardinality Aggregator retains one of the tuples (selected at random) and discards the rest. It then computes $\sum \frac{f}{w}$ on the remaining tuples. Similar to Push-Sum, given a tuple $(s, (f, w)))$, $\frac{f}{w}$ estimates the average of s's frequency over Δ peers. Therefore, $\Delta \times \sum \frac{f}{w}$ is output as the cardinality estimate of the query.

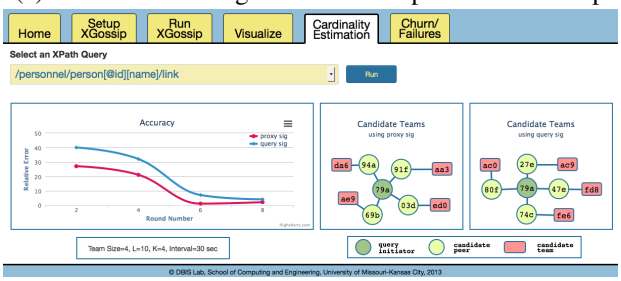
Next, we state our main result on the quality of cardinality estimates computed by XGossip [20]: Given an XPath query q, let R denote the set of distinct document signatures that are divisible by q's signature. Suppose q_{min} denotes the minimum similarity between q's signature and a signature in R. XGossip

can estimate the cardinality of q with a relative error of at most $|R| \cdot \epsilon$ and a probability of at least $(1-\delta)$ in $O(log(\Delta) + log(\frac{1}{\epsilon}) + log(\frac{\alpha}{\alpha+\delta-1}))$ rounds, where $\alpha = 1-(1-q_{min}^{l})^{k}$, and k and l are the parameters of the LSH function.

III. Demonstration Scenarios



(a) Screenshot during the execution phase of XGossip



(b) Screenshot during cardinality estimation

Fig. 3. Screenshots of the tool

XGossip is written in C++ and runs atop Chord [21]. The user interface is a web application written in Python and Javascript. (See Figure 3 for a few screenshots.) Users will experience three engaging scenarios during the demonstration.

(1) Setup, configuration, and deployment: First, users can setup and configure XGossip by choosing the number of peers in the network, team size, number of gossip rounds, time interval between successive gossip rounds, distribution of published XML documents, and tuning parameters of LSH. Then they can deploy XGossip on up to 400 EC2 instances in a desired EC2 availability zone.

(2) Execution and cardinality estimation: Next, users can execute XGossip and pose a variety of XPath queries including those containing '//' and '*'. They can observe the convergence speed of the frequency of signatures gossiped by different peers and their teams, the bandwidth usage in different rounds, the number of messages exchanged, the average message size, and the convergence of the cardinality estimates to their true values using both query and proxy signatures, in real-time. They can re-run XGossip with different settings and observe the effect on its convergence speed, the accuracy of cardinality estimation, and the bandwidth usage.

(3) Churn and failures: Finally, users can introduce churn and failures, including peer crashes, during the execution of XGossip. For churn, they can select the number of short-lived peers and the session lengths of these peers based on a log-normal distribution. For failures, they can choose the percentage of peers that crash abruptly during the execution of XGossip. They can observe the impact of churn and failures on the convergence speed of XGossip and the accuracy of cardinality estimates. They can also disable the Mass Adjuster causing disturbance to mass conservation and observe how this phenomenon impacts the behavior of XGossip.

References

[1] D. Fenstermacher, C. Street, T. McSherry, V. Nayak, C. Overby, and M. Feldman, "The cancer biomedical informatics grid (caBIG)," in *Proc. of IEEE Eng. in Medicine & Biology Soc.*, Shanghai, 2005, pp. 743–746.

[2] 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.

[3] "The HITECH Act," http://www.gpo.gov/fdsys/pkg/BILLS-111hr1enr/pdf/BILLS-111hr1enr.pdf.

[4] 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*, vol. 20, no. 1, pp. 71–78, 2006.

[5] 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 VLDB Conference*, San Francisco, 2001, pp. 591–600.

[6] 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 ICDE Conference*, Heidelberg, Germany, 2001, pp. 595–604.

[7] 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, Jun. 2002.

[8] 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*, Toronto, Canada, 2004, pp. 240–251.

[9] N. Polyzotis, M. Garofalakis, and Y. Ioannidis, "Selectivity estimation for XML twigs," in *Proc. of the 20th ICDE Conference*, Boston, 2004.

[10] 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 ICDE Conference*, Atlanta, 2006, p. 61.

[11] D. K. Fisher and S. Maneth, "Structural selectivity estimation for XML documents," in *Proc. of the 23th ICDE Conference*, Istanbul, Turkey, 2007, pp. 626–635.

[12] 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 EDBT Conference*, Saint Petersburg, Russia, 2009, pp. 335–344.

[13] "The healthcare enterprise repository for ontological narration (HERON)," http://informatics.kumc.edu/work/wiki/HERON.

[14] Y. Hu, J. G. Lou, H. Chen, and J. Li, "Distributed density estimation using non-parametric statistics," in *Proc. of 27th ICDCS Conference*, Jun. 2007, pp. 28–36.

[15] T. Pitoura and P. Triantafillou, "Self-join size estimation in large-scale distributed data systems," in *Proc. of the 24th ICDE Conference*, Cancun, Mexico, April 2008.

[16] N. Ntarmos, P. Triantafillou, and G. Weikum, "Statistical structures for internet-scale data management," *The VLDB Journal*, vol. 18, no. 6, pp. 1279–1312, 2009.

[17] "Amazon S3 availability event," http://status.aws.amazon.com/s3-20080720.html.

[18] 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 the 21st ACM SOSP Conference*, Stevenson, Washington, 2007, pp. 205–220.

[19] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proc. of the 44th IEEE Symp. on Foundations of Computer Science*, Cambridge, MA, Oct 2003.

[20] V. Slavov and P. Rao, "A gossip-based approach for internet-scale cardinality estimation of XPath queries over distributed semistructured data," *The VLDB Journal*, 2013, dx.doi.org/10.1007/s00778-013-0314-1.

[21] 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*, 2001, pp. 149–160.

[22] 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*, vol. 21, no. 12, pp. 1737–1752, December 2009.

[23] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. of the 13th ACM Symp. on Theory of Computing*, Dallas, Texas, 1998, pp. 604–613.

[24] T. H. Haveliwala, A. Gionis, D. Klein, and P. Indyk, "Evaluating strategies for similarity search on the web," in *Proc. of the 11th WWW Conference*, Honolulu, 2002, pp. 432–442.