Replication Routing Indexes for XML

Panos Skyvalidas, Evaggelia Pitoura, Vassilios V. Dimakopoulos, and Georgia Koloniari

Department of Computer Science, University of Ioannina, Greece
{skyval,pitoura,dimako,kgeorgia}@cs.uoi.gr

Abstract. Peer-to-peer systems have attracted considerable attention as a means of sharing content among large and dynamic communities of peers. In this paper, we consider sharing of XML documents. We propose a simple yet efficient replication method based on replication routing indexes. A *replication routing index* for a peer is a path-based XML index that summarizes access information for each of the links of the peer. These indexes are used for determining both the granularity of the XML fragments to be replicated as well as appropriate peers for placing the fragments. We also present experimental results of the deployment of our indexes in a dynamic unstructured peer-to-peer system.

1 Introduction

Peer-to-peer (p2p) systems have attracted a lot of attention as a means of data sharing among large and dynamic populations of peers. In a p2p system, each peer connects with (knows about) a small number of other peers, thus forming an *overlay network*. A central issue is locating peers that hold data of interest. Proposals for constructing overlays vary from forming rigid topologies and placing data on specific peers to unstructured networks where there is no control over the network topology or the content placement. In all overlay types, content replication reduces the latency of lookups. Various replication techniques have been proposed that can be categorized roughly as passive or proactive.

With *passive replication*, items are replicated after being successfully located. A commonly used passive replication scheme is *path replication* that caches data items along the lookup path. It has been shown that the number of copies produced by path replication is close to optimal with respect to the expected number of steps to reach a locatable item [6, 9]. With *proactive replication*, the creation of replicas is initiated by holders of data items not necessarily after a successful lookup of the item. Path replication is simple and requires minimum bookkeeping. However, it tends to cluster copies along lookup paths. Further, passive replication considers the average search cost over all items and thus it does not allow for fine tuning the search efficiency for popular and unpopular items.

To support proactive replication in unstructured p2p systems, we introduce a data structure called replication routing index. A *Replication Routing Index (RepRI)* for a peer has one entry for each of its links with statistics regarding the requests it has received through this link. Our replication strategy uses these indexes for fine tuning search by deciding whether to maintain a copy locally or forward it as well as for selecting a direction for doing so. RepRI entries are also

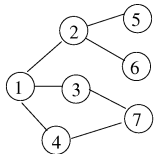
used as hints during lookup to direct searches towards paths that most probably hold replicas of the requested items.

Motivated by the fact that XML is increasingly being used in data intensive applications, we also study both passive and proactive replication in the case of sharing XML content. XML documents have a hierarchical structure and thus, different document fragments may have different access frequencies. Our *Replication Routing Index for XML (RepRIX)* maintains statistics for fragments instead of whole documents, thus allowing for fine tuning the unit of replication, so that fragments of the same document may have different numbers of replicas.

We have compared experimentally both proactive and passive variations of fragment replication. Both types outperform whole document replication by increasing the percentage of items located and reducing the required steps for doing so. Proactive replication with hints is shown to work better than passive replication. Furthermore, proactive replication allows for fine tuning the search efficiency for popular and unpopular items at the additional overhead of maintaining replication indexes.

2 Proactive Replication in Unstructured p2p Systems

First, we consider unstructured overlays where peers share simple data files. Search is based on keyword queries that refer to file names. In unstructured overlay networks, a search request for an item needs to be propagated through the overlay using some variation of flooding or random walks. With *flooding*, each peer contacts its neighbors in the overlay, which in turn contact their own neighbors, until the requested item is located. With *random walks*, the peer initiates one or more depth-first traversals in the overlay. In both cases, to avoid overwhelming the network, the search query is propagated for up to a maximum of TTL (Time-To-Live) hops.



Filename	LReq	Req1	Hop1	Req2	Hop2	Req3	Hop3	Ownership
Talk_mp3	0	8	4	2	6	4	4	1
Sun_jpg	4	3	2	0	7	1	3	0
Precious_mp3	0	5	1	2	6	4	5	1

Fig. 1. The replication routing index of peer 1

To support proactive replication, we introduce a data structure that we call *Replication Routing Index (RepRI)* which maintains statistics regarding file requests. The $RepRI(p)$ of a peer p maintains entries corresponding to each of its links (neighbors) summarizing the requests received through this link. For simplicity, entries are for individual files. Appropriate techniques, such as maintaining information for groups of file requests, can be used to reduce the size of the structure. The entry $RepRI(p)(f)$, for a peer p with k neighbors and file f is of the form $(LReq,\ Req_1, Hop_1,\ \ldots Req_k,\ Hop_k, Ownership)$, where $LReq$ is the number of queries for file f initiated by peer p itself, Req_i and Hop_i with i

$= 1, \ldots, k$, are respectively the number of queries that have been forwarded to p by its neighbor i and the average number of hops required for the corresponding queries to reach p. Ownership is set to 1 and 0 depending on whether file f is stored locally at p or not. Figure 1 shows the RepRI maintained by peer 1, which has three neighbors, 2, 3 and 4. Peer 1 has processed queries for two local files, *Talk.mp3* and *Precious.mp3* and for one file *Sun.jpg* stored at some other peer.

Our goal is to create replicas towards the direction they are actually needed. The decision which file to replicate and towards which direction is based on both the popularity of the file and the cost for locating it. To this end, each peer p with k neighbors calculates the *replication utility*, $ru(f)$, of a file f as follows:

$$ru(f) = \alpha \; pop\_factor(f) + (1 - \alpha) \; hop\_factor(f)$$

where $pop\_factor(f) = \sum_{i=1}^{k} Req_i(f) \; / \; Max_{f'}(\sum_{i=1}^{k} Req_i(f'))$ and $hop\_factor(f) = (\sum_{i=1}^{k} Hop_i(f)/k)/TTL$. $Req_i(f)$ and $Hop_i(f)$ denote respectively the number of queries for file f and the corresponding number of hops for these queries that p has received from its neighbor i. The weight α is a tuning parameter that determines how much each of these two factors, namely popularity and search size, affects replication. The larger the value of α, the more efficient the search for popular items. Each peer p also maintains the *average replication utility (aru)* for all files in its index.

Periodically, each peer p creates replicas of all locally stored files that have replication utility greater than or equal to its average replication utility. For each such file f, peer p sends a replication message to its neighbor i that has the maximum corresponding $Req_i(f)$. A peer i that receives a replication message for a file f consults its own *RepRI* index and decides to store a copy of f if $LReq(f)$ is greater than 0 or $ru(f)$ is greater than or equal to aru. Peer i also checks whether it should forward the replication message to its neighbors based on the corresponding $Req(f)$ fields. After each replication phase, the entries for all files in the RepRI of the initiating peer p are reset, thus providing a simple aging scheme for the entries. The period of the replication procedure depends on the query workload. A large value leads to making more informed decisions based on sufficiently large samples of requests and ignores popularity fluctuations that may be caused by random variations in the query workload. Furthermore, it reduces the associated network overhead. On the other hand, the system adapts to workload changes less promptly.

When a peer reaches its maximum storage capacity for replicas, it replaces the file with the smallest replication utility. Finally, when a peer forwards a replication message to one of its neighbors without having stored it locally, it sets a corresponding counter to a *hint value* indicating that the file has been copied along this direction. Thus, the next time the peer receives a query for that file, it knows towards which of its neighbors to forward the request.

3 Replication Routing Indexes for XML

We consider replication for XML. An XML document is represented by a rooted labeled tree, where nodes correspond to XML elements labeled with the corresponding tag and edges represent direct element-subelement relationships. Fig-

ure 2 depicts an XML description of a song collection provided by a node and the corresponding XML tree. To support querying the structure of documents, in addition to their content, query languages for XML employ path patterns. We focus on simple path queries of the form $p_1\ e_1\ p_2\ e_2\ \ldots\ p_n\ e_n$ where each e_i is an element name or the wildcard operator * and each p_i is either / or // denoting respectively parent-child and ancestor-descendant traversal.

A path query q is evaluated at a node u of an XML tree T and its result is the set of nodes of T reachable from u via q. For a query q and a document d, we say that d matches q, if the path expression forming the query exists in the document. A peer that stores documents that match the query is called a *matching peer*. Using path queries, peers receive as results fragments of the information included in an XML document. An XML *fragment* F_i is a subtree of T rooted at some node n of T. Each fragment is represented by a simple path expression starting from the root node r of T and leading to n. In Figure 2 (shaded part), we see an example of an XML fragment represented by the path */song_collection/rock/**. Different fragments of a document may have different access frequencies. We exploit this fact by replicating at the fragment level.

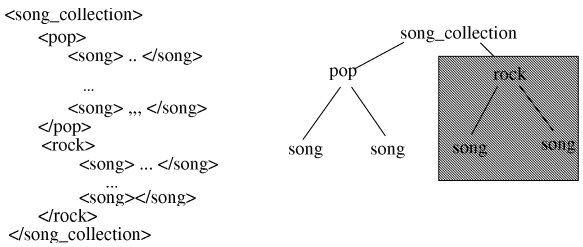


Fig. 2. (left) Example of an XML document (right) The corresponding tree

We assume that the documents stored initially at peers are not fragmented. Document fragmentation is the result of replication. We also assume that a unique identifier is associated with each document. This is useful for handling possible updates. We propose two techniques for replica creation, namely skeleton and subtree replication.

With *skeleton replication*, for creating a replica for a path expression $/a_1\ /a_2\ \ldots\ /a_n$, we copy the skeleton (element hierarchy) of the original document from the root node a_1 to the fragment-root node a_n. We also copy the data contained in the subtree defined by the path query. Data that corresponds to elements that are not included in the related path is not replicated. Instead, it is replaced by an external link. External links are special elements identified by the tag name *externalLink* used to indicate where the data of the element, to which they belong, is located. External links can be viewed as an intentional description of this missing data and provide the means to obtain it, if needed.

With *subtree replication*, the replicas that are created contain only the fragment defined by the path query and not the whole skeleton, that is, the replica

just consists of the subtree (of the original document) rooted at the fragment-root node. In addition, for evaluating a path query over a replica, three attributes are added to its root element: the *path*, the *parent* node and the *hasSiblings* attributes. The *path* attribute contains the sequence of element names from the root node of the original document to the root node of the replica fragment. Since all documents have a unique id, the *parent* node attribute contains the parent node's id for achieving cohesion between the original document and the replica. Finally, the *hasSiblings* attribute takes the values *yes* or *no* indicating whether the replica's root node has any sibling nodes which have not been replicated.

Instead of maintaining statistics for whole documents, a *Replication Routing Index for XML (RepRIX)* maintains entries that correspond to paths, representing fragments that the peer has processed queries for. Peers make decisions regarding replicating fragments of documents based on the replication utilities of the corresponding paths. An important point is that the replication utility of a path is calculated by taking into account the replication utility of all paths that are subsumed by it. A path p_1 is subsumed by a path p_2, if whenever a document d matches p_2, it also matches p_1. Checking XML query subsumption is a well-studied problem (for instance, see [12] for a survey). In this paper, we use a simple test: we check whether p_2 is a prefix of p_1.

Table 1. Simulation Parameters

Parameters	**Default value**	**Range**
Network size	5000	1000-10000
Random walkers	16	
File distribution	Random	
Query distribution	Zipf (alpha = 1.0)	alpha = 0.0 - 2.0
TTL	7	
Avg file size	4KB	3.5KB-4.5KB
Avg fragment size	1.3KB	0.5KB-3KB
Storage limit	16KB	8KB-64KB

4 Experimental Evaluation

All replication techniques are implemented as components of the Peersim simulator [10]. Initially, all documents are distributed randomly among the peers. The documents used for the experiments are generated using the ToXgene [14] XML generator. The queries are simple path expressions extracted by the data set. The search mechanism is random walks with TTL. The network follows power-law topologies, with sizes ranging from 1000 to 10000 peers. In all experiments, we consider that the system poses a constraint on the number of replicas that can be stored at each peer. Each peer is the owner of approximately one document and maintains replicas of 2-16 others resulting in about 2-16 replicas per document on average. Table 1 summarizes our experimental parameters.

We present two sets of experiments. In the first set, we consider proactive and path fragment replication, while in the second set, we compare whole doc-

ument versus fragment replication. All reported values were averaged over multiple runs. The search sizes presented are those of successful queries. In these experiments, we use skeleton replication for both proactive and path replication. Skeleton replication achieves better results than subtree replication, since external links speed up search. Skeleton replication answers about 6% more queries than subtree replication and decreases the search size by around 15%. However, subtree replication creates messages having smaller sizes by as much as 25%.

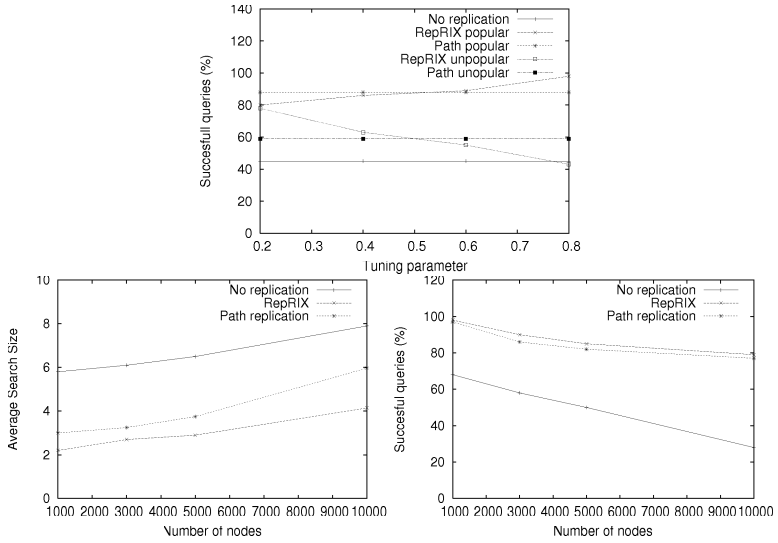


Fig. 3. (top) Query hits for popular and unpopular fragments for various α. (bottom-left) Average search size and (bottom-right) query hits for various network sizes

Proactive Replication vs Path Replication. In all experiments, both techniques use all the space available for replica storage, creating approximately the same number of replicas. We first consider how the tuning parameter α of the replication utility affects search. In Figure 3(top), we report the percentage of successful searches for the 20% most popular and the 20% most unpopular fragments. As α increases, the number of replicas of popular fragments increases and thus the percentage of successful queries for popular fragments increases as well, while for unpopular fragments, it drops. Similarly, our results have shown that increasing the value of α reduces the search size for popular fragments (from 4 to around 2 hops) and increases the search size of unpopular ones (from 4 to around 6.5). Thus, α can be set appropriately to fine tune the search efficiency of popular and unpopular items. For example, a small value of α (for instance $\alpha = 0.3$) increases the number of unpopular items that can be located and reduces their search size. In the following, we assume $\alpha = 0.6$ and report average values.

Regarding scalability, we tune the period of replication for proactive replication so that the number of replication messages is about the same with that in the case of passive (path) replication. Figures 3(bottom-left) and 3(bottom-

right) show that RepRIX replication outperforms path replication with respect to both the average search size and the percentage of successful queries.
Fragment vs Whole Document Replication. Figures 4(top-left) and 4(top-right) present a comparison of fragment and whole document XML replication for both path and RepRIX replication. As shown, in both cases, fragment replication manages the available storage much more efficiently storing only the most popular fragments resulting in both smaller search sizes and more successful queries. Figure 4(bottom) shows how the size of the popular fragments affects the performance of fragment replication. When most queries refer to relatively small fragments, both RepRIX and path fragment replication methods show better performance, since more fragments are replicated.

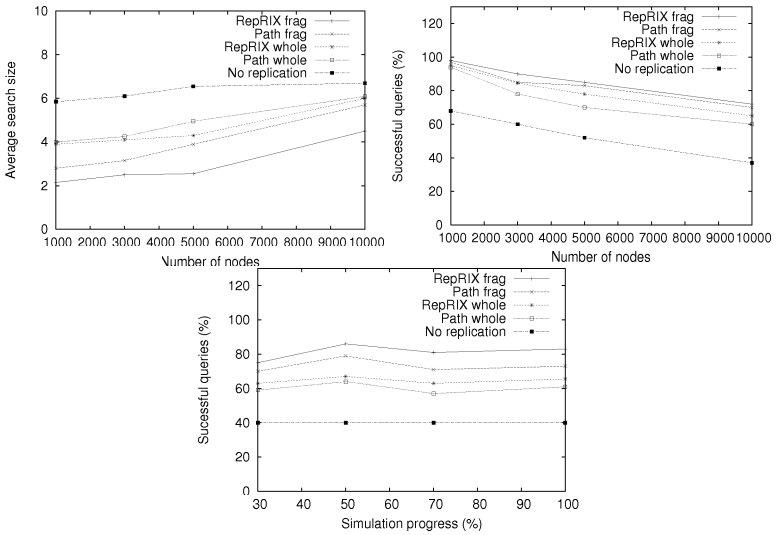


Fig. 4. (top-left) Average search size and (top-right) query hits for fragment and whole document replication. (bottom) Average search size with the size of popular fragments

5 Related Work

Replication in unstructured p2p systems has received considerable attention. The authors of [9] proved that square-root replication that allocates replicas to items proportionally to the square root of their query rate produces the optimal number of copies in terms of optimizing the average search time. Path replication was shown to achieve almost square-root replication under certain conditions [6, 9]. In this paper, we compare path replication with a proactive scheme for replica allocation that allows for tuning the search cost of popular and unpopular items. Beehive [11] describes a proactive replication protocol for structured overlays that achieves O(1) lookup. The basic difference with our approach is that in structured overlays, items are placed at specific peers and thus the search cost can be reduced by carefully placing replicas of an item along its search path. However, in unstructured overlays, the search path for an item is not known.

There is some work on distribution and XML. In [1], the authors consider dynamic XML documents that contain both materialized and intentional XML data that can be produced by web service calls. The focus is on whether to materialize the result of a call or not. The authors of [5] deal with the problem of supporting parallel query processing when parts of an XML document are located at different peers. Research in [3] and [4] addresses the XML allocation problem, that is given a collection of XML documents how to fragment them and allocate the resulting fragments among the nodes of a distributed system to improve performance. There has also been work on XML processing XML in both structured (e.g., [2, 7, 13]) and unstructured p2p systems (e.g., [8]). Such research is complementary to ours, since in this paper, we address replication.

6 Summary

Replication is commonly used in peer-to-peer systems to improve response time of queries. In this paper, we focus on replicating XML documents. A distinctive feature of XML replication is determining an appropriate granularity that would allow different levels of replication for fragments of the same XML document. We have presented both a proactive and a passive replication protocol to achieve such replication at the fragment level. Our proactive replication protocol is based on replication routing indexes that are path-based indexes used to determine the appropriate peers towards which to create copies. Our experimental results show that our fragment-based replication model outperforms whole document replication, while proactive replication works better than passive replication at the cost of maintaining access statistics.

References

1. Abiteboul S., Bonifati A.,. Cobena G, Manolescu I., and Milo T.: Dynamic XML Documents with Distribution and Replication. In SIGMOD (2003)
2. Bonifati A., Matrangolo U., Cuzzocrea A., Jain M.: XPath lookup queries in P2P networks. In WIDM (2004)
3. Bremer, J. M., Gertz M.: On Distributing XML Repositories. In WebDB (2003)
4. Buneman P., Choi B., Fan W., Hutchison R., Mann R., Viglas S. D.: Vectorizing and Querying Large XML Repositories. In ICDE (2005)
5. Buneman P., Cong G., Fan W., Kementsietsidis A.: Using Partial Evaluation in Distributed Query Evaluation. In VLDB (2006)
6. Cohen E., Shenker S.: Replication strategies in unstructured peer-to-peer networks. In SIGCOMM (2002)
7. Galanis L., Wang Y., Jeffery S., and DeWitt D.: Locating Data Sources in Large Distributed Systems. In VLDB (2003)
8. Koloniari G., Pitoura E.: Content-Based Routing of Path Queries in Peer-to-Peer Systems. In EDBT (2004)
9. Lv Q., Cao P., Cohen E., Li K., Shenker S.: Search and Replication in Unstructured Peer-to-Peer Networks. In ICS (2002)
10. Peersim website: http://peersim.sourceforge.net
11. Ramasubramanian V., Sirer E. G.: Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays. In NSDI (2004)
12. Schwentick T.: XPath Query Containment. In Sigmod Record, 331(1). (2004)
13. Skobeltsyn G., Hauswirth M., Aberer K.: Efficient Processing of XPath Queries with Structured Overlay networks. In ODBASE (2005)
14. ToXgene website: http://www.cs.toronto.edu/tox/toxgene