A Distributed Selectivity-driven Search Strategy for Semi-structured Data over DHT-based Networks

Carmela Comito[1], Domenico Talia[2], Paolo Trunfio[2]

[1] *ICAR-CNR, Rende (CS), Italy*

[2] *DIMES, University of Calabria, Rende (CS), Italy*

{*ccomito,talia,trunfio*}*@dimes.unical.it*

Abstract

Distributed Hash Tables (DHTs) are widely used for indexing and locating many types of resources, including semi-structured data modeled as XML documents. A common distributed strategy to process an XML query over a DHT consists in splitting it into a set of simple path queries, and resolving each of them separately. The traffic generated by this strategy grows with the number of paths in the query. To overcome this drawback, an alternative strategy consists in resolving only the sub-query associated with the most selective path, and then submitting the original query to the nodes in the result set. A first goal of this paper is to provide an analytical and experimental study of the two strategies to assess their relative merits in different scenarios. On the basis of this study, we introduce an Adaptive Path Selection (APS) search technique that resolves an XML query in a distributed way by querying either the most selective path or the whole path set, based on the selectivity of the paths in the query. The effective use of APS requires that the querying nodes know in advance the selectivity of all the paths. Addressing this problem is another goal of the paper, which is achieved through: *i*) The definition of a space-efficient data structure, the Path Selectivity Table (PST), which given any path, returns an estimate of its selectivity. *ii*) The definition of an efficient strategy that builds the PST in a distributed way and propagates it to all nodes in the network with logarithmic performance bounds and without redundant messages. Experimental results show that the PST accurately estimates the path selectivity values, and that the traffic generated by the APS algorithm using PST-estimated selectivity values is comparable to that produced by APS assuming to know the real path selectivity values.

Keywords: Distributed Hash Tables, Semi-structured Data, Path Selectivity, Adaptive Path Selection.

1. Introduction

Distributed Hash Tables (DHTs) are decentralized systems providing scalable services for indexing and locating data in large-scale networks. DHT-based systems like Chord [1], Pastry [2], Tapestry [3], and Kademlia [4], assign to each node the responsibility for a specific part of the data to be shared. In a network of n nodes, when a node wants to find a data item identified by a given key, a DHT allows to locate the node responsible for that key in O(log n) hops, using only O(log n) neighbors per node. Thanks to their inherent reliability

and autonomic properties, DHTs can be effectively used in dynamic peer-to-peer networks with nodes continuously joining and leaving [1], as well as in static decentralized systems composed by a large number of nodes permanently connected to a wide-area network [34]. In both cases, an important system goal is limiting the network traffic generated by the distributed query processing. A key toward this goal is efficiently locating relevant data sources, so as to submit the queries only to the nodes where those data sources are stored.

Leveraging a DHT, complex queries over a large collection of distributed data can be processed with guarantee that all the relevant documents are located with logarithmic performance bounds. In a DHT-based system, a query Q can be processed in two phases: *i*) the DHT is looked up to identify all nodes that store data matching Q; *ii*) Q is submitted to each node identified during the previous phase, to get all the data matching Q. In this work we focus on the first phase of the query processing, with the goal of minimizing the amount of traffic generated to identify the nodes that will be queried during the second phase.

Even though DHTs can be used for indexing many types of data, in this paper we concentrate on XML-based semi-structured data, as XML is widely used as a language for information representation and exchange over the Internet. We assume that data sources are distributed over a large number of nodes (i.e., tens to hundreds of thousands) permanently connected to the network like, for instance, a world-wide network of Internet-enabled sensor stations organized in a DHT. Another example is a large-scale DHT-based network of service providers, where a large number of services, published by different providers in XML format, need to be dynamically discovered and integrated into complex distributed applications. Examples include e-commerce and e-science applications, where the single components are available as services independently specified by their providers [40].

The indexing of an XML document D in a DHT can be done by associating a key to each path p in D; then, the node responsible for the key associated with p keeps a pointer to the nodes storing all documents containing p, including D. To search for XML documents matching a complex tree pattern query formulated, for instance, as an XPath or XQuery expression, a basic strategy consists in splitting the query into a number of sub-queries, one for each path in the query. Each sub-query is resolved independently to find the set of nodes that store documents matching the corresponding path. The result sets coming from the different sub-queries are intersected at the querying node. Then all the nodes in the intersection set are queried with the original query to obtain all the documents matching that query.

The network traffic generated by the strategy above increases with the number of paths in the query. This can lead to system inefficiency in case of complex queries composed of several sub-queries, particularly in presence of many concurrent requests. To overcome this drawback, an alternative strategy consists in resolving only the sub-query associated with the most selective path, i.e., the path that matches the lowest number of nodes; then all nodes in the result set can be queried with the original query to get the documents that satisfy all the query constraints (including those associated with the other paths). The selectivity of a path is defined as follows:

Definition 1. *(Path Selectivity). The selectivity* $\mathrm{s_p}$ *of a path* p *is given by the equation:*

$$\mathrm{s_p} = \mathrm{n_p}/\mathrm{n} \tag{1}$$

where $\mathrm{n_p}$ *is the number of nodes that store at least one instance of path* p*, and* n *is the total number of nodes in the DHT. By definition,* $0 < \mathrm{s_p} \leq 1$.

The lower the selectivity value, the more selective the path; in other words, the lowest selectivity value corresponds to the most selective path. For instance, in a network with 10,000 nodes, a path stored in 50 nodes has a selectivity of 50/10,000=0.005, while a path stored in 5,000 nodes has a selectivity of 5,000/10,000=0.5. The former can be an example of highly selective path (low selectivity value), the latter of lowly selective path (high selectivity value).

A first goal of this paper is to provide an analytical and experimental study of the two strategies to assess their relative merits in different scenarios. On the basis of this study, we introduce an Adaptive Path Selection (APS) search technique that resolves an XML query by querying either the most selective path or the whole path set, based on the selectivity of the paths in the query. APS uses path selectivity values to calculate the traffic that would be generated querying the most selective path and the whole path set, which allows selecting the most efficient strategy to follow for a given query. Experimental results confirm that APS saves a significant amount of traffic compared to the two strategies from which it derives.

The effective use of APS requires that all nodes know in advance the selectivity of all the paths. If path selectivity values are not known a priori, techniques for estimating the path selectivity values can be used. We propose a compact data structure, called *Path Selectivity Table* (PST), which groups paths with similar selectivity values into a fixed number of buckets. Bloom Filters are used to represent the paths in each bucket so that, for a given path, the bucket containing the selectivity of the path can be quickly located. Thus, given any path, the PST returns an estimate of its selectivity.

Our solution differs from existing systems since it supports node-based selectivity estimation, which allows every peer to estimate the total number of nodes that store sources relevant to a query. This is a key advantage as it allows estimating in advance the network traffic that will be generated by any query, as the traffic produced by distributed query processing depends on the number of nodes with relevant sources. The node-based selectivity estimation strategy enables another unique feature of our solution, which is adaptive lookup. In fact, our APS search strategy allows peers to resolve a query by querying either the most selective path or the whole path set. This permits to achieve good traffic performance, while maintaining a basic indexing/search scheme that can be easily implemented on top of any DHT. Another unique feature exhibited by our system is local selectivity estimation. In fact, the PST allows every participant peer to estimate locally the selectivity of a path, without querying the network for this purpose.

A preliminary version of this work, which focused on introducing the APS technique, was presented in [30]. In the present version we make the following additional contributions:

1. A more formal definition of the APS algorithm and of the two basic strategies from which it derives, together with their analytical and experimental evaluation.
2. The definition of a space-efficient data structure, the PST, for XML path selectivity estimation in a distributed scenario.
3. The definition of a PST Construction and Propagation (PST_CP) algorithm that builds the PST in a distributed way and propagates it to all nodes in the network with logarithmic performance bounds.
4. An experimental evaluation demonstrating the PST accuracy for path selectivity estimation, as well as the efficiency of the PST in supporting APS search in distributed scenarios.

We point out that the APS technique could be exploited by non-holistic related approaches that similarly to us index XML data over a DHT using paths as indexing elements. In fact, the APS elaborates on the two basic query resolution strategies to process XML queries over a DHT, proposing an adaptive strategy that uses either the whole path set or the most selective path. This way, related approaches could achieve better results in terms of network traffic generated because depending on the specific scenario one strategy could outperform the other one. Moreover, both holistic and non-holistic approaches could exploit our PST Construction and Propagation (PST_CP) algorithm to construct a PST-like data structure in a distributed way to propagate selectivity estimation all over the network. This can be particularly relevant for centralized approaches that would be able this way to implement a distributed strategy to process XML queries over a DHT.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 presents the system model, including the data model and the way XML documents are indexed. Section 4 describes and compares the two basic approaches exploited by the APS search strategy to answer an XML query. Section 5 proposes the APS algorithm and evaluates it in different scenarios. Section 6 presents the PST-based approach to path selectivity estimation, including a detailed description of the PST_CP algorithm. Section 7 evaluates the accuracy of PSTs for path selectivity estimation and their effectiveness in supporting APS-based query processing. Finally, Section 8 concludes the paper.

2. Related work

The work that first introduced the WPS and MSP approaches that we rely on is [36], which proposes a Multi-Attribute Addressable Network (MAAN) that extends Chord to support multi-attribute and range queries. MAAN addresses range queries by mapping attribute values to Chord via uniform locality preserving hashing. It uses an iterative (corresponding to our WSP search) or single attribute dominated query (corresponding to our MSP search) routing algorithm to resolve multi-attribute queries. Apart than the similarities on the rational of query resolution, our work is quite different from the one in [36] as detailed in the following. A first difference is that the work in [36] is not tailored for XML data as it is in our case, thus, it is based on a different indexing structure. Moreover, we elaborate on the two query resolution strategies, proposing an adaptive one that uses either the whole path set or the most selective path. This way we achieve better results in terms of network traffic generated because depending on the specific scenario one strategy could outperform the other one. Another main difference is on the selectivity concept as we rely on a quite different selectivity definition. Furthermore, [36] does not focus on the way how selectivity values are estimated and propagated all over the network. Differently, an important contribution of our work is selectivity estimation, through the definition of a space-efficient data structure, the Path Selectivity Table (PST), which given any path, returns an estimate of its selectivity. Moreover, we also defined an efficient strategy that builds the PST in a distributed way and propagates it to all nodes in the network with logarithmic performance bounds and without redundant messages.

In the remainder of the section we review existing works related to the main topics addressed in this paper: *i*) estimating XML path selectivity; *ii*) XML data indexing and discovery using DHTs.

2.1. XML path selectivity estimation

Accurate estimation of XML path selectivity values represents a key issue to enable efficient query processing. As mentioned before, knowing the selectivity of the paths allows one to improve the execution of a multi-path XML query by querying directly the most selective path. This topic has been extensively studied in centralized environments and several approaches have been proposed.

One of the first works proposed in the field was presented in [14]. This work proposes two techniques for estimating the selectivity of simple path expressions: path trees and Markov tables. The path tree collapses the tree corresponding to the original XML document by deleting or coalescing low-frequency nodes, whereas the Markov table stores the counts of all frequently occurring paths of length less than a given threshold. The disadvantage of both approaches is that large structures must be constructed before they are pruned, which can be very space intensive.

A related system is XPathLearner [15], which includes a method for estimating selectivity of the most commonly used types of path expressions based on a feedback analysis. XPathLearner stores selectivity related statistics in a Markov table, which is space consuming.

Polyzotis and Garofalakis proposed TreeSketch [20], an extension to XSketch [19], which estimates the cardinality of XML twig queries by capturing the key structural information (i.e., label path and branching) and using graph synopses. In case of complex datasets, with the presence of recursions and a higher structural irregularity, the construction process of TreeSketch can be computationally complex and time consuming [23]. An example of complex XML dataset is TreeBank, which exhibits a complex structure and has a high degree of recursions. TreeSketch, however, is orders of magnitude more accurate than XSketch in estimating results cardinality and needs less time to construct.

Zhang et al. [26] proposed the Xseed synopsis to summarize the structural information of XML data. The information is stored in two structures, a small kernel, which summarizes basic structural information, and an hyperedge table (HET), which provides additional information about the tree structure. The HET enhances the accuracy of the synopsis and makes it adaptive to different memory budgets. Xseed supports recursion by recording "recursion levels" and "recursive path expression" in the synopses. Although the construction of Xseed is reported to be generally faster than that of TreeSketch, it is still time-consuming for complex datasets [23].

In [27], a synopsis based on a lossy compression of the XML document tree is proposed. The synopsis can be computed in one pass from the document. This work has two main advantages over existing approaches: *i*) the estimator returns a range within which the actual selectivity is guaranteed to lie, with the size of this range implicitly providing a confidence measure of the estimate, and *ii*) the synopsis can be incrementally updated to reflect changes in the XML database.

In general, most of the techniques proposed in literature rely on some kind of structure synopses, usually in the form of a compressed tree. The construction of such synopses is usually slow and costly, especially for complex XML datasets. Besides, as their efficiency strongly depends on the summarization of the structural relationships and simple statistics such as node counts, child node counts, etc., due to the compression of information, selectivity estimation heavily relies on the statistical assumptions of independence and uniformity. Consequently, they can suffer from poor accuracy when these assumptions are not

valid. Therefore, structure synopses could perform badly on complex XML datasets. Consequently, one of the main drawbacks of most of the techniques proposed in the literature, is that they fail in achieving both estimation accuracy and space efficiency.

The work in [31] successfully addresses both the issues by proposing a novel data structure, the Bloom Histogram (BH), which approximates XML paths frequency distribution by sorting on the frequencies, and using Bloom Filters to record values within each bucket. Compared to the other alternatives discussed above, which use data structures such as path trees and Markov Tables, the use of a Bloom Filter to maintain approximate values of path selectivity is of smaller size, and offers superior accuracy as the estimation error is small and can be probabilistically bounded.

The most important difference between our approach and the related works described above, is that our solution has been designed for distributed scenarios, while the others are meant for local settings. Another major difference between our work and all the others (but the one by Wang et al.[31]), is that we do not model the intricate structural relationships of XML documents, since this is not required to efficiently identify the nodes storing all documents containing the paths specified in a input query, which is the focus of our work.

Inspired by the BH data structure proposed in [31], we propose a Path Selectivity Table (PST) data structure, which given any path, returns an estimate of its selectivity. Consequently, we inherit all the advantages of the work in [31] in terms of small space occupancy and accuracy, since the estimation error is small and can be tuned probabilistically. However, our work is substantially different as it deals with Internet-scale distributed XML documents. To cope with the distributed scenario, the PST is based on a different selectivity definition as compared to that used in the BH. In particular, the selectivity of a path in our system is calculated by counting the number of distinct nodes holding that path. Additionally, our framework defines an algorithm to build the PST in a distributed way and to efficiently propagate it to all nodes in the network.

2.2. DHT-based XML data indexing and discovery

Several systems for sharing and querying distributed collections of XML data using a DHT as the core component have been proposed in literature. A survey presenting the state of the art of P2P XML data management can be found in [9] and another one in [10]; the latter is a more recent survey focusing on XML data indexing and retrieval in P2P systems.

Skobeltsyn et al. [6] proposed a DHT-based index for XPath queries. Paths are hashed to obtain keys that are then stored in the P-Grid DHT [41]. The set of XML element tags is used as the alphabet. Given an XML path p consisting of g element tags, g data items are stored in the P-Grid network using the sub-paths (suffixes) in p as keys. The key of each data item is generated using a prefix-preserving hash function.

Miliaraki et al. [12] proposed an approach for filtering distributed XML documents using a non deterministic finite automata (NFA) which encodes a set of XPath queries on top of a Chord DHT. The key aspect of such a work is a distributed implementation of YFilter, a state-of-the-art automata-based XML filtering system on top of Chord. The authors present and evaluate two methods for executing the NFA. In the iterative method a publisher peer is responsible for managing the execution of the NFA while states are retrieved from other network peers. The recursive method exploits the inherent parallelism of an NFA and executes several active paths of the NFA in parallel.

In [8], Karanasos et al. proposed the ViP2P system, a P2P platform for sharing XML data. While our work is mainly concerned with locating the peers storing information

relevant to a query, Karanasos et al. focus on query answering over a global XML database distributed over a Pastry overlay. The database is maintained in the form of distributed materialized XML views defined as XPath queries. The authors proposed three indexing strategies: (i) Label indexing (LI) strategy, which indexes a view by each node label in it (either element or attribute name, or word); (ii) Return label indexing (RLI), where a view is indexed by the labels of all nodes which project some attributes; (iii) Leaf path indexing (LPI) strategy, where a view is indexed by all the distinct root-to-leaf label paths in it. Here, a path is just the sequence of labels encountered as one goes down from the root to the node.

Abiteboul et al. [7] introduced the KadoP system that indexes XML data in the form of postings, where each posting encodes information on an element or a keyword. KadoP is built atop a Pastry DHT, in which a single node stores all the postings for a given term. KadoP supports holistic processing of XPath queries: a tree pattern matching algorithm is executed in order to compute the identifiers of the documents matching the input query exploiting holistic twig join. Since the posting lists for very popular terms grow very large limiting the system scalability, Bloom Filters are used for a compact representation of the set of postings of a given term.

The system proposed by Galanis et al. [5] indexes XML paths in a Chord DHT based on the parent-child relationships and using tag names as keys. A node responsible for an XML tag stores and maintains a data summary (corresponding to structural ancestor information) with all possible unique element paths leading to that tag. Thus, only one tag of a query is used to locate the responsible node. All the key-summary pairs are stored in the DHT, similarly to the way that keys and instances are organized into an inverted list. For a given XPath query the element tag at its leaf position is used as a lookup key in the DHT.

Garces-Erice et al. [11] proposed an approach that, for a given XML document, constructs an XPath expression that tests the presence of all the elements and values in that document. This expression represents the most specific query for the document. Given this expression, the system builds a hierarchy of indexes using a generic DHT containing query-to-query mappings such that a user can look up more specific queries for a given broader query, thereby refining her interests. Such an approach allows users to locate data even using scarce information, although at the price of a higher lookup cost.

In the psiX system proposed by Rao and Moon [13] each XML document is mapped into an algebraic signature that captures the structural summary of the document. An XML query pattern is also mapped into a signature to locate relevant document signatures. The signature scheme supports holistic processing of query patterns without breaking them into several simple path queries and processing them individually. The participating nodes in a Chord network collectively maintain a collection of distributed hierarchical indexes for the document signatures.

Finally, Slavov and Rao [16] proposed Xgossip, a gossip algorithm that given an XPath query, estimates the number of XML documents containing a match for that query [16]. In XGossip, a peer gossips the signature of an XML document, which is computed based on the method proposed by Rao and Moon [13]. XGossip adopts the Push-Sum protocol [29] and is composed of two main phases. In the initialization phase, each peer creates a list of tuples using only local information, where each tuple includes the signature of an XML document published by the peer. In the gossip phase, peers exchange their lists of tuples using an epidemic approach, so that the document signatures tend to be distributed to all

peers as the number of gossip rounds increases.

Table 1 compares the proposed solution with the DHT-based XML data indexing and discovery systems discussed above. The comparison takes into account the following features:

- *Main goal.* XML query processing in a DHT can be seen as a two-step task: *i*) *Locating sources*, where the DHT is looked up to identify all the sources relevant to the query; *ii*) *Query answering*, where the query is processed against all the relevant sources to get matching data. Therefore, we classify the systems based on the main goal they address, either locating sources or query answering. As shown in the table, some systems ([6], [12], [8] and [7]) focus on query answering, while others ([5], [11], [13], [16] and the proposed system) focus on locating sources. It is worth noticing that both approaches can be used to answer a query. In fact, with the second approach, once sources have been located, it is possible to ask those sources to process the query in order to get data matching it.

- *Indexed elements.* The systems can be also classified according to the elements used to index XML data on the DHT. The analyzed systems follow different approaches: *i*) indexing each label in the data (*labels*); *ii*) indexing each path or path query (*paths* or *path queries*); *iii*) the index is represented by some kind of data summarization (*document summaries*). Some of the systems ([6], [12], [8], [7], [11] and the proposed system) index either labels, paths or path queries, while the others ([5], [13] and [16]) adopt complex summarization techniques to process the queries holistically.

- *Supported queries.* Another important feature for classification purpose is the type of queries supported by the systems. Here we distinguish between *linear queries*, which are supported by [6], [12], [5], [11] and the proposed system, and *twig queries*, which are supported by [8], [7], [13] and [16].

- *DHT overlay.* The underlying distributed hash table used to implement the XML indexing and discovery system. Most of the systems are based on Chord ([12], [5], [13], [16] and the proposed system). Other systems use Pastry ([7] and [8]), P-Grid ([6]) or a generic DHT ([11]).

- *Selectivity estimation.* Whether a system supports or not selectivity estimation. In case of systems that do support selectivity estimation, we distinguish between *document-based* and *node-based* selectivity estimation. Given an XML query q, document-based approaches estimate the total number of XML documents in the network that contain a match for q, while node-based approaches estimate the total number of nodes that store XML sources that contain a match for q. The only two systems that support selectivity estimation are Xgossip [16] (document-based) and the proposed system (node-based).

- *Local selectivity estimation.* A system supports local selectivity estimation if every participant peer can estimate the selectivity of a path by consulting some data structure available locally, without querying the network for this purpose. The proposed system is the only one supporting local selectivity estimation, because each node in the DHT maintains locally a space-efficient data structure, the Path Selectivity Table (PST), which given any path, returns an estimate of its selectivity.

Table 1: Comparison with related DHT-based systems.

System	**Main goal**	**Indexed elements**	**Supported queries**	**DHT overlay**	**Selectivity estimation**	**Local selectivity estimation**	**Adaptive lookup**
Skobeltsyn et al. [6]	Query answering	Paths	Linear queries	P-Grid	NO	N.A.	NO
Miliaraki et al. [12]	Query answering	Path queries	Linear queries	Chord	NO	N.A.	NO
ViP2P [8]	Query answering	Paths and labels	Twig queries	Pastry	NO	N.A.	NO
KadoP [7]	Query answering	Labels	Twig queries	Pastry	NO	N.A.	NO
Galanis et al. [5]	Locating sources	Document summaries	Linear queries	Chord	NO	N.A.	NO
Garces-Erice et al. [11]	Locating sources	Paths	Linear queries	Generic	NO	N.A.	NO
PsiX [13]	Locating sources	Document summaries	Twig queries	Chord	NO	N.A.	NO
Xgossip [16]	Locating sources	Document summaries	Twig queries	Chord	YES (document-based)	NO	NO
Proposed system	Locating sources	Paths	Linear queries	Chord	YES (node-based)	YES	YES

- *Adaptive lookup.* The ability of dynamically choosing the lookup strategy taking into account the selectivity of the query elements, i.e. based on the number of sources that contain data relevant to the query. This feature is provided only by the proposed system. In fact, our Adaptive Path Selection (APS) search technique resolves an XML query in a distributed way by querying either the most selective path or the whole path set, based on the selectivity of the paths in the query.

As shown in Table 1, Xgossip [16] and the proposed system are the only two solutions supporting selectivity estimation for XML query processing over a DHT. As discussed earlier, enabling the estimation of XML path selectivity values allows improving the execution of multi-path XML queries. This is an important feature not only because it allows improving query processing, but also because it enables the formulation of selectivity-specific queries (i.e., estimating the total number of documents or nodes that store XML sources that contain a match for a given query), which are not supported by the other systems. In particular, our system differs from Xgossip because it supports node-based selectivity estimation, which allows every peer to estimate the total number of nodes that store sources relevant to a query. This is a key advantage because it allows estimating in advance the network traffic that will be generated by any query, as the traffic produced by distributed query processing depends on the number of nodes with relevant sources, according to Equation 2.

The node-based selectivity estimation strategy enables another unique feature of our solution, which is adaptive lookup. In fact, our Adaptive Path Selection (APS) search strategy allows peers to resolve a query by querying either the most selective path or the whole path set, based on the selectivity of the paths in the query. As detailed in Section 5, this permits to achieve good performance in terms of network traffic, while maintaining a basic indexing/search scheme that can be easily implemented on top of any DHT.

Finally, another unique feature exhibited by our system is local selectivity estimation.

This allows every participant peer to estimate locally the selectivity of a path, without querying the network for this purpose. In order to achieve so, each peer in the DHT maintains locally a space-efficient data structure, the Path Selectivity Table (PST), which given any path, returns an estimate of its selectivity. To this end, we build and propagate the PST using an efficient broadcast algorithm that, after a fixed number of steps, and without redundant messages, ensures that all peers share the same PST generated at a given time.

3. System model

We assume a system composed of a set $\mathcal{N}$ of autonomous nodes, organized in a DHT-based structured P2P network like Chord [1], and a set $\mathcal{D}$ of XML documents distributed over those nodes and indexed using the DHT to support their efficient identification and retrieval.

Example 1. *Consider the case of a world-wide network of Internet-enabled weather sensor stations, organized in a DHT-based Chord overlay. Each sensor station locally stores, in XML format, a sensor station descriptor (SSD) and a set of sensor readings (SRs). The SSD provides relevant information such as geographical location, available sensors, and the features of each sensor (e.g., instrument, type, precision). SRs are XML fragments containing data sensed over the time by the station's sensors. The DHT is used to index the SSDs to support the identification of sensor stations that meet the desired constraints (e.g., the presence of a given sensor type). The relevant SRs of the sensor stations so identified can be subsequently retrieved for application purposes.*

In the rest of the section we first describe the XML data model and the queries supported by the system, and then the strategy adopted to index the XML documents.

3.1. Data model and supported queries

XML documents are represented as ordered labeled trees, according to the DOM standard[1]. Each node in the tree corresponds to an element, or an attribute, or text data; edges between nodes represent element/subelement or element/attribute relationships. More precisely, leaf nodes correspond to data values and internal nodes define the document structure. According to this model, an XML path can be defined as follows.

Definition 2. *(Path). Let* D *be an XML document and* T *the associated tree, each branch in* T *identifies a path* p *from the root of* T *to a leaf node, where intermediate nodes are element labels, and edges are parent/child and/or ancestor/descendant relationships.*

The system supports path-based XML query languages such as XPath and XQuery. Such languages treat an XML document as a tree and offer an expressive way to specify and select parts of an XML document by navigating its tree structure. Thus, XML queries naturally impose a structural pattern on XML data and the queries are accordingly referred to as *tree-pattern* queries. This kind of queries are usually expressed by means of XPath

[1] http://www.w3.org/DOM

expressions, which define a way of navigating an XML tree and return the set of tree nodes which are reachable from one or more starting nodes through the paths specified by the expressions.

An XPath expression contains one or more location steps, separated by slashes ("/"). In its more basic form, a location step designates an element name; a more complex location step specifies an element name followed by zero or more predicates specified between brackets. Predicates are generally specified as constraints on the presence of structural elements, or on the values of tagged data using basic comparison operators. An XPath expression composed only by simple location steps identifies a unique path in the tree associated with an XML document. Conversely, an XPath expression containing also predicates identifies more than one path. Queries with predicates are also referred to as branching queries: each element involved in the predicate identifies a unique path in the XML tree.

We support a subset of the XPath query language denoted as $\{., /, [\,]\}$. The expressions of such a subset of XPath are given by the following grammar:

$$\mathrm{Q} \rightarrow \mathrm{l} \mid . \mid \mathrm{Q} \ / \ \mathrm{Q} \mid \mathrm{Q} \ [\ \mathrm{Q} \ ]$$

where Q is a generic query, l is any label (node test), "." denotes the current node, "/" indicates the child axis, and "[ ]" indicates a predicate.

3.2. Indexing strategy

The goal of the indexing strategy is to provide an efficient means to find the set of nodes that store the path(s) matching a query expression. Since each path is a textual string, it can be transformed into a numeric key through hashing. Accordingly, we define a path key as follows.

Definition 3. *(Path Key). Let* D *be an XML document and* p *a path* $\in$ D*, a key* $\mathrm{k_p}$ *for* p *is defined as* $\mathrm{k_p} = \mathrm{h_t(p)}$*, where* $\mathrm{h_t}$ *is a hash function that consistently maps a textual string to a large set of numeric keys.*

To keep association between path keys and nodes we assume the use of a Chord DHT, though any other similar DHT-based system could be used with minimal variations. In Chord, each node maintains a *Finger Table* (FT) and a *Key Table* (KT). The FT points to nodes at exponentially increasing distance, and allows to locate the node responsible for a given key in $\mathrm{O}(\log \mathrm{n})$ hops, where n is the number of nodes in the network. The KT keeps association between each key the node is responsible for, and all the nodes that contain documents matching that key. Details about the Chord protocol are given in [1].

Fig. 1 provides a small-scale example showing how XML documents are indexed in our model using a Chord DHT. In this example, a 4-bit identifier space is used, thus both node identifiers and path keys are assigned a value in the range $[0, 15]$. There are only five active nodes in the network, $\mathrm{N_0}$, $\mathrm{N_3}$, $\mathrm{N_6}$, $\mathrm{N_{10}}$, and $\mathrm{N_{13}}$. Each node locally stores a set of XML documents. For example, $\mathrm{N_6}$ stores, among the others, two documents $\mathrm{D_1}$ and $\mathrm{D_2}$, while one of the documents stored by $\mathrm{N_{13}}$ is $\mathrm{D_3}$. Each distinct path in $\mathrm{D_1}$, $\mathrm{D_2}$, $\mathrm{D_3}$ is assigned a unique key. As shown in the figure, documents $\mathrm{D_2}$ and $\mathrm{D_3}$ contain the same paths, $\mathrm{p_1}$, $\mathrm{p_3}$ and $\mathrm{p_9}$, so they are identified by the same path keys ($\mathrm{k_{p_1}}$, $\mathrm{k_{p_3}}$ and $\mathrm{k_{p_9}}$). Moreover, $\mathrm{p_1}$ is present in all three documents. According to the Chord protocol, each key is assigned to the first active node whose identifier equals or follows the key value. Therefore, $\mathrm{k_{p_1}}$ and $\mathrm{k_{p_3}}$ are stored into the KT of $\mathrm{N_3}$, $\mathrm{k_{p_9}}$ is put into the KT of $\mathrm{N_{10}}$, and so on. A KT stores,

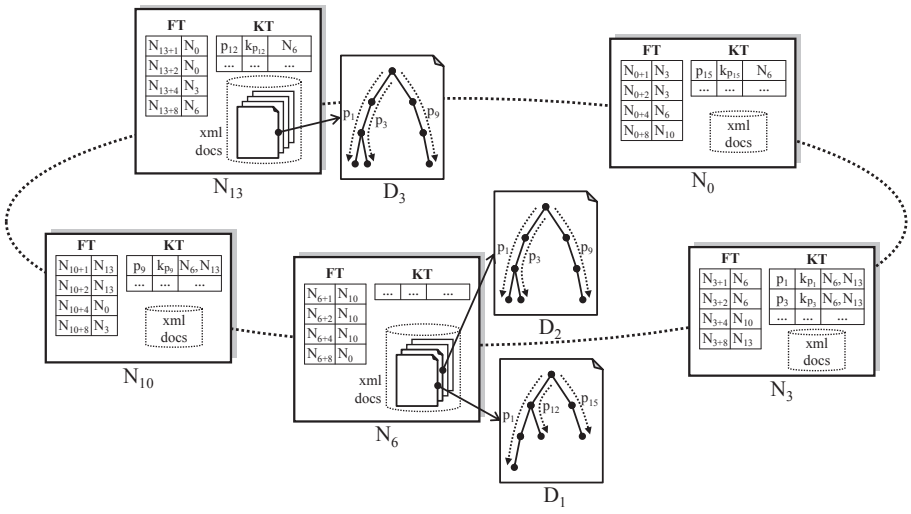

Figure 1: Indexing XML documents over a Chord DHT.

for each key k_p, the set of nodes that possess at least one document containing path p. For example, in the KT of node N_{10}, k_{p_9} points to nodes N_6 and N_{13}, since both of them contain at least one document (D_2 and D_3, respectively) associated with k_{p_9}. Even though - for the sake of clarity - KT entries in Fig. 1 contain only node identifiers, actually each entry contains also all the information (often referred to as "connection string", see next section) required to access the XML collection where the document is stored.

4. Query processing

In a DHT-based system, the query processing can be divided into two phases:

1. the DHT is looked up to identify all nodes that store XML documents matching a query Q;
2. Q is sent to the nodes identified in the previous phase, which will execute Q locally.

As stated earlier, we focus on the first phase of the query processing, with the goal of minimizing the amount of traffic generated to identify the nodes that will be queried during the second phase.

To search for XML documents indexed in a DHT matching a multi-path query, a commonly-used strategy consists in splitting the query into a number of sub-queries, one for each path. Each sub-query is resolved independently to find the set of nodes that store documents matching the corresponding path. The result sets coming from the different sub-queries are intersected at the querying node; then all the nodes in the intersection set are queried with the original query to obtain all the documents matching that query. The network traffic generated by this strategy increases with the number of paths in the query. To overcome this issue, an alternative strategy consists in resolving only the sub-query associated with the most selective path; then all nodes in the result set are queried with the original query. In the following we refer to those two strategies as *Whole Path Set* (WPS) and *Most Selective Path* (MSP) search, respectively. The idea behind MSP is that, having information about the selectivity of the various paths in the query, it can be more conve-

nient in terms of traffic generated to query the most selective path rather than the whole path set, as done by the WPS strategy.

Path selectivity is defined according to Eq. 1. To the purpose of the query processing techniques discussed in this section, we assume that the querying nodes know in advance the selectivity of all the paths in the query to be processed. We will discuss in Section 6 the solution employed in our framework to estimate path selectivity values when they are not known a priori.

```
<SensorStn id="Melbourne01">
  <Location>
    <Country>Australia</Country>
    <State>Victoria</State>
    <City>Melbourne</City>
    ...
  </Location>
  <Sensors>
    <Sensor id="Melbourne01_Thermometer01">
      <Instrument>Thermometer</Instrument>
      <Type>Infrared</Type>
      <Precision>0.01</Precision>
    </Sensor>
    <Sensor id="Melbourne01_Thermometer02">
      <Instrument>Thermometer</Instrument>
      <Type>Mercury</Type>
      <Precision>0.02</Precision>
    </Sensor>
    ...
  </Sensors>
</SensorStn>
```

Figure 2: Example of sensor station descriptor.

Example 2. *Referring to the scenario outlined in Example 1, assume that sensor station descriptors (SSDs) are structured like the one shown in Fig. 2. Suppose that a user issues the following XPath query to find the cities in Australia where there is a sensor station with a thermometer, infrared type, having precision equal to 0.01:*

```
/SensorStn[Sensors[Sensor[Instrument="Thermometer" and
Precision="0.01" and Type="Infrared"]]]
[Location[Country="Australia"]]/Location/City
```

The query consists of the following paths:

```
/SensorStn/Sensors/Sensor/Instrument="Thermometer"
/SensorStn/Sensors/Sensor/Precision="0.01"
/SensorStn/Sensors/Sensor/Type="Infrared"
/SensorStn/Location/Country="Australia"
/SensorStn/Location/City
```

In case of WPS search, five lookups (i.e., one for each path) must be performed on the DHT. Conversely, in case of MSP search, only the most selective path (e.g., the one that refers to the country) will be looked up on the DHT.

The remainder of this section describes the WPS and MSP search techniques and analyzes them in terms of the network traffic they produce to process a query. Since the traffic

generated depends both on the number and the size of messages, the following parameters will be used to calculate the size of the different messages exchanged during the query processing:

- S_H: the size of a message header, i.e., the fixed amount of traffic each message generates independently from the specific payload;
- S_S: the size of a sub-query, i.e., the average size of a path expression extracted from the original query;
- S_Q: the size of the original query expression;
- S_C: the average size of the "connection string" required to access an XML collection; it includes information about the database implementation, the location of the database server, the name of the collection, and credential information.
- $S_{\mathcal{A}}$: the size of the answer set $\mathcal{A}$, which is composed of all the XML fragments that match the query criteria.

4.1. WPS search

Chord offers efficient and scalable single-key or single-path lookup service for decentralized resources. However, it cannot support multi-path based lookup. The WPS approach addresses this problem by extending Chord with an iterative multi-path query resolution mechanism. Specifically, given a multi-path query, the WPS strategy consists in splitting the query into a number of sub-queries, one for each path. Each sub-query is resolved independently to find the set of nodes that store documents matching the corresponding path. The result sets coming from the different sub-queries are intersected at the querying node; then all the nodes in the intersection set are queried with the original query to obtain all the documents matching that query.

The WPS search algorithm is described in Fig. 3. The input of the algorithm are: the original query Q and the set of paths $\mathcal{P}$ extracted from Q. The output is the answer set $\mathcal{A}$, i.e. the set of all the XML fragments that match Q. The algorithm defines a set $\mathcal{N}_{\mathcal{P}}$ to contain the identifiers of all the nodes with documents that match all the paths in $\mathcal{P}$. $\mathcal{N}_{\mathcal{P}}$ is initialized to $\mathcal{N}$, the set of all nodes in the network. For each path p_i the following operations are performed:

1. the key k_{p_i} corresponding to p_i is computed through hashing;
2. the node responsible for p_i, $N^r_{p_i}$, is identified through a lookup operation performed over the DHT;
3. $N^r_{p_i}$ returns the set of nodes, $\mathcal{N}_{p_i}$, which store at least one document matching p_i;
4. $\mathcal{N}_{\mathcal{P}}$ is intersected with $\mathcal{N}_{p_i}$.

Once the final set $\mathcal{N}_{\mathcal{P}}$ is obtained, each node in $\mathcal{N}_{\mathcal{P}}$ is asked to process the original query Q producing a partial answer, i.e. the set of XML fragments stored on that node that match Q. The final answer set $\mathcal{A}$ is obtained as the union of all the partial answers.

Although the WPS search algorithm is simple, the traffic generated by this technique increases proportionally with the number of paths in the query, as shown in the following.

Algorithm WPS_Search
Input: query Q, set of paths $\mathcal{P}$
Output: answer set $\mathcal{A}$

```
begin
    A ← ∅;
    N_P ← N;
    foreach path p_i ∈ P do
        k_{p_i} ← h_t(p_i);
        N^r_{p_i} ← lookup(k_{p_i});
        N_{p_i} ← N^r_{p_i}.returnNodes(k_{p_i});
        N_P ← N_P ∩ N_{p_i};
    end
    foreach node N_i ∈ N_P do
        A_{N_i} ← N_i.answerQuery(Q);
        A ← A ∪ A_{N_i};
    end
    return A;
end
```

Figure 3: Whole Path Set search.

Theorem 1. *(WPS traffic). Let* $\mathrm{T_{WPS}}$ *be the average network traffic generated by the WPS search algorithm to process a query* Q *composed by a set of paths* $\mathcal{P} = (\mathrm{p_1}, ..., \mathrm{p_m})$ *submitted by a node* $\mathrm{N_Q}$ *in a Chord network with* n *nodes. Then,* $\mathrm{T_{WPS}}$ *is given by the following equation:*

$$\begin{aligned} \mathrm{T_{WPS}} = \mathrm{S_{\mathcal{A}}} + \mathrm{m} \cdot \mathrm{S_H} + \tfrac{1}{2} \cdot (\mathrm{S_H} + \mathrm{S_S}) \cdot \mathrm{m} \cdot \log_2 \mathrm{n} + \\ + \sum_{\mathrm{i}=1}^{\mathrm{m}} (\mathrm{S_C} \cdot \mathrm{n} \cdot \mathrm{s_{p_i}}) + (2 \cdot \mathrm{S_H} + \mathrm{S_Q}) \cdot \mathrm{n} \cdot \prod_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{s_{p_i}} \end{aligned} \tag{2}$$

Proof. We distinguish four steps in the WPS search, and use notation $\mathrm{T_{WPS_i}}$ to indicate the traffic generated during the i-th step:

- Step 1: $\mathrm{N_Q}$ performs m lookups to identify the m nodes, $\mathrm{N^r_{p_1}}, ..., \mathrm{N^r_{p_m}}$, which are responsible for the different paths in $\mathcal{P}$.

  Traffic: On average, each lookup requires $\frac{1}{2} \cdot \log_2 \mathrm{n}$ routing hops [1]. Thus $\frac{1}{2} \cdot \mathrm{m} \cdot \log_2 \mathrm{n}$ messages are generated. The size of each message is given by $\mathrm{S_H} + \mathrm{S_S}$. Therefore, $\mathrm{T_{WPS_1}} = \frac{1}{2} \cdot (\mathrm{S_H} + \mathrm{S_S}) \cdot \mathrm{m} \cdot \log_2 \mathrm{n}$.

- Step 2: $\mathrm{N_Q}$ receives m responses, one from each of nodes $\mathrm{N^r_{p_1}}, ..., \mathrm{N^r_{p_m}}$.

  Traffic: Each response is delivered directly to $\mathrm{N_Q}$, so m messages are generated. Response $\mathcal{N}_{\mathrm{p_i}}$ sent by a node $\mathrm{N^r_{p_i}}$ contains the identifiers of all nodes that store documents with path $\mathrm{p_i}$. Thus, the size of message $\mathcal{N}_{\mathrm{p_i}}$ is given by $\mathrm{S_H}$, plus a term $\mathrm{S_C} \cdot \mathrm{n} \cdot \mathrm{s_{p_i}}$ that is proportional to the number of node identifiers in $\mathcal{N}_{\mathrm{p_i}}$. Therefore, $\mathrm{T_{WPS_2}} = \sum_{\mathrm{i}=1}^{\mathrm{m}} (\mathrm{S_H} + \mathrm{S_C} \cdot \mathrm{n} \cdot \mathrm{s_{p_i}})$.

- Step 3: $\mathrm{N_Q}$ sends Q to all nodes that are in the intersection set, $\mathcal{N}_{\mathcal{P}}$, of all the m responses received on the previous step.

  Traffic: Assuming that the selectivity values $\mathrm{s_{p_1}}, ..., \mathrm{s_{p_m}}$ are independent of each other, on average the size of $\mathcal{N}_{\mathcal{P}}$, and so the number of messages, is given by $\mathrm{n} \cdot \prod_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{s_{p_i}}$, which is proportional to the product of all the selectivity values. The size of each message is $\mathrm{S_H} + \mathrm{S_Q}$. Therefore, $\mathrm{T_{WPS_3}} = (\mathrm{S_H} + \mathrm{S_Q}) \cdot \mathrm{n} \cdot \prod_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{s_{p_i}}$.

- Step 4: N_Q receives a response from each node queried on the previous step.

  Traffic: The number of response messages received is given by $n \cdot \prod_{i=1}^{m} s_{p_i}$, which is equal to the number of query messages sent on the previous step. The traffic is given by S_H for each message received, plus the size of the total answer set $S_{\mathcal{A}}$, which is is the union of all the partial answer sets received by the different nodes queried. Therefore, $T_{WPS_4} = S_H \cdot n \cdot \prod_{i=1}^{m} s_{p_i} + S_{\mathcal{A}}$.

By summing the traffic generated at the different steps, we finally obtain Eq. 2. ∎

Definition 4. *(WPS overhead traffic). We define the WPS overhead traffic,* $\overline{T}_{WPS}$*, as* T_{WPS} *minus the size of the answer set* $S_{\mathcal{A}}$*:*

$$\overline{T}_{WPS} = T_{WPS} - S_{\mathcal{A}} \tag{3}$$

4.2. MSP Search

While WPS is a balanced search strategy where all the paths in the query have the same weight, MSP gives more relevance to the most selective path. In fact, according to the WPS strategy, the search result of a multi-path query is an iterative process where the result set must satisfy all the sub-queries on each path and it is the intersection set of all resources that satisfies each individual sub-query.

The idea behind MSP is that, having information about the selectivity of the various paths in the query, it can be more convenient in terms of traffic generated to query the most selective path rather than the whole path set. In this case it is only necessary to compute a set of candidate resources that satisfies the sub-query on the most selective path. Then, all nodes in the result set can be queried with the original query to get the documents that satisfy all the query constraints (including those associated with the other paths).

The MSP search algorithm is shown in Fig. 4. The input parameters are: the original query Q and the most selective path p_{min} in $\mathcal{P}$. The output is the answer set $\mathcal{A}$. First, the key $k_{p_{min}}$ associated with the most selective path p_{min} is determined through hashing, and the node responsible for p_{min}, $N^r_{p_{min}}$, is identified through the lookup of $k_{p_{min}}$. $N^r_{p_{min}}$ returns the set $\mathcal{N}_{p_{min}}$ of nodes storing documents with path p_{min}. Then, each node in $N^r_{p_{min}}$ is queried with Q producing a partial answer. The union of all the answers produces the final answer set $\mathcal{A}$.

Algorithm MSP_Search
Input: query Q, most selective path p_{min}
Output: set of answers $\mathcal{A}$

```
begin
    k_pmin ← h_t(p_min);
    N^r_pmin ← lookup(k_pmin);
    N_pmin ← N^r_pmin.returnNodes(k_pmin);
    foreach node N_i ∈ N_pmin do
        A_Ni ← N_i.answerQuery(Q);
        A ← A ∪ A_Ni;
    end
    return A;
end
```

Figure 4: Most Selective Path search.

Theorem 2. *(MSP traffic). Let* T_{MSP} *be the average network traffic generated by the MSP search algorithm to process a query* Q *composed by a set of paths* $\mathcal{P} = (p_1, ..., p_m)$ *submitted by a node* N_Q *in a Chord network with* n *nodes. Then,* T_{MSP} *is given by the following equation:*

$$\begin{aligned} T_{MSP} = S_{\mathcal{A}} + S_H + \tfrac{1}{2} \cdot (S_H + S_S) \cdot \log_2 n + \\ + (S_C + 2 \cdot S_H + S_Q) \cdot n \cdot s_{p_{min}} \end{aligned} \tag{4}$$

Proof. We proceed as in Theorem 1 by distinguishing four steps in the MSP search, and using notation T_{MSP_i} to indicate the traffic generated during the i-th step:

- Step 1: N_Q performs one lookup to identify the node, $N^r_{p_{min}}$, which is responsible for the most selective path, p_{min}.

  Traffic: The lookup requires $\frac{1}{2} \cdot \log_2 n$ messages, on average. The size of each message is given by $S_H + S_S$. Therefore, $T_{MSP_1} = \frac{1}{2} \cdot (S_H + S_S) \cdot \log_2 n$.

- Step 2: N_Q receives one response from $N^r_{p_{min}}$.

  Traffic: The response is delivered directly to N_Q, so just one message is generated. This response message, $\mathcal{N}_{p_{min}}$, contains the identifiers of all nodes that store documents with p_{min}. Therefore, its size is given by S_H, plus a term $S_C \cdot n \cdot s_{p_{min}}$ that is proportional to the number of node identifiers in $\mathcal{N}_{p_{min}}$. Thus, $T_{MSP_2} = S_H + S_C \cdot n \cdot s_{p_{min}}$.

- Step 3: N_Q sends Q to all nodes in $\mathcal{N}_{p_{min}}$.

  Traffic: The number of nodes in $\mathcal{N}_{p_{min}}$ is proportional to $_{p_{min}}$. Thus, the number of messages is $n \cdot s_{p_{min}}$. The size of each message is given by $S_H + S_Q$. Therefore, $T_{MSP_3} = (S_H + S_Q) \cdot n \cdot s_{p_{min}}$.

- Step 4: N_Q receives a response from each node queried on the previous step.

  Traffic: The number of messages received is given by $n \cdot s_{p_{min}}$, which is equal to the number of messages sent on the previous step. The traffic generated is given by S_H for each message received, plus the size of the total answer set $S_{\mathcal{A}}$. Thus, $T_{MSP_4} = S_H \cdot n \cdot s_{p_{min}} + S_{\mathcal{A}}$.

Eq. 4 is finally obtained by summing the traffic generated at the different steps. ∎

Definition 5. *(MSP overhead traffic). We define the MSP overhead traffic,* $\overline{T}_{MSP}$*, as* T_{MSP} *minus the size of the answer set* $S_{\mathcal{A}}$*:*

$$\overline{T}_{MSP} = T_{MSP} - S_{\mathcal{A}} \tag{5}$$

4.3. Comparison between WPS and MSP

We evaluated the amount of overhead traffic generated by the WPS and MSP search techniques over a Chord DHT in different scenarios. To this end, we implemented a simulator that evaluates the amount of traffic generated by the two techniques to resolve a query, given the total number n of nodes in the network, the number m of paths composing the query, and the range $(0, u]$ of selectivity the paths belong to.

For each combination of the input parameters (n, m, and u), the simulator generates $q = 10^4$ different m-tuples, which correspond to q different queries. The values in each tuple $(a_1, a_2, ..., a_m)$ are real numbers uniformly distributed over the interval $(0, u]$, where

$0 < u \leq 1$. The i-th value of a tuple, a_i, represents the selectivity of the i-th path of the corresponding query, i.e., $a_i = s_{p_i}$. For each of the q different tuples, the values of $\overline{T}_{WPS}$ and $\overline{T}_{MSP}$ are calculated. Finally, the mean values of $\overline{T}_{WPS}$ and $\overline{T}_{MSP}$ are obtained as an average over the q results.

The following values have been used for the parameters introduced earlier: $S_H = 260$ Bytes; $S_S = 60$ Bytes; $S_Q = S_S \cdot m$; $S_C = 75$ Bytes. S_H has been determined by measuring the average traffic generated to transfer an empty message between two hosts using TCP. S_S is the average length of the path expressions observed in two well-known reference XML databases, namely Protein Sequence Database (PSD) and DBLP[2]. S_C has been chosen by observing the typical length of the connection string required to access an XML collection[3].

Fig. 5 shows the average values of $\overline{T}_{WPS}$ and $\overline{T}_{MSP}$ with an increasing number of paths ($m = 2$ to 12) in three scenarios: *a*) $n = 100,000$ and $u = 0.5$; *b*) $n = 200,000$ and $u = 0.5$; *c*) $n = 200,000$ and $u = 0.1$. In all three cases, there is a value of m under which the average value of $\overline{T}_{WPS}$ is lower than the average value of $\overline{T}_{MSP}$. In particular, MSP is better than WPS when $m > 4$ for all three cases. As the value of m increases, the advantage of MSP over WPS becomes higher. We can observe that this result is independent from both the size of the network (case *a* vs. case *b*) and the range of selectivity the paths belong to (case *b* vs. case *c*), except for a scale factor.

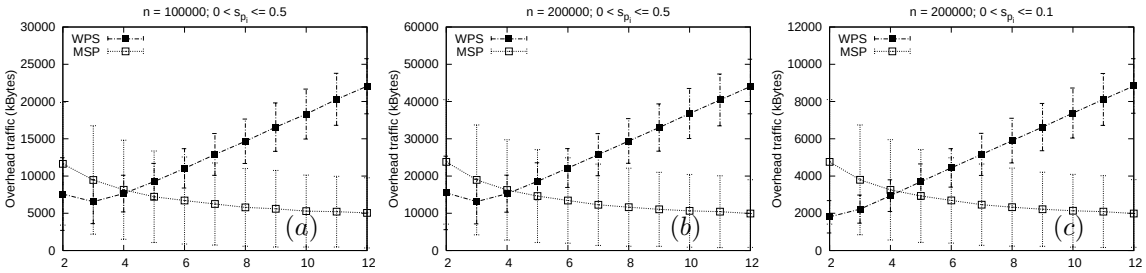

Figure 5: Average values of $\overline{T}_{WPS}$ and $\overline{T}_{MSP}$ in three scenarios: *a*) $n = 100,000$ and $u = 0.5$; *b*) $n = 200,000$ and $u = 0.5$; *c*) $n = 200,000$ and $u = 0.1$. Error bars represent the standard deviations from the average.

It is interesting to observe that the standard deviation (represented as error bars in Figs. 5*a*-*c*) of $\overline{T}_{MSP}$ is rather high. This means that in many cases the value of $\overline{T}_{MSP}$ can be significantly lower than the average value. To better highlight this aspect, Fig. 6 shows the percentage of cases in which $\overline{T}_{MSP}$ is lower than $\overline{T}_{WPS}$ in the same scenarios *a*-*c* introduced earlier. We can see that, even for low values of m, there is a percentage of cases in which MSP generates less traffic than WPS. For example, when $m = 2$ the percentage of cases in which MSP is better than WPS ranges from about 1% (for case *c*) to 13% (for case *b*). This percentage rapidly increases, reaching more than 90% when $m = 8$ in all scenarios.

[2]These databases are available from: `http://www.cs.washington.edu/research/xmldatasets/www/repository.html`

[3]Example of connection string, not including credentials: `xmldb:exist://gridlab1.dimes.unical.it:8080/exist/xmlrpc/db/collname`

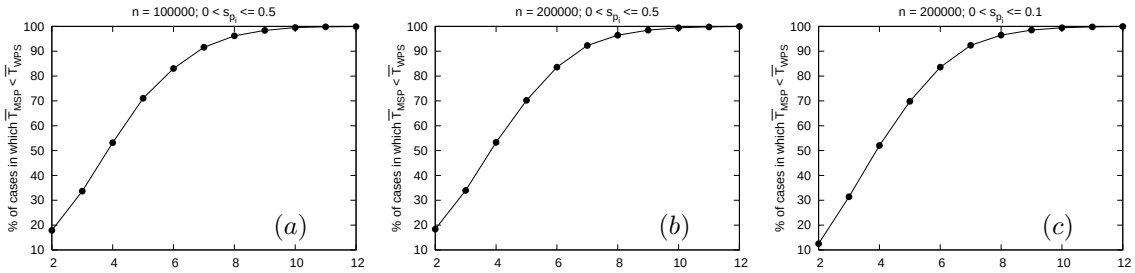

Figure 6: Percentages of cases in which $\overline{\mathrm{T}}_{\mathrm{MSP}} < \overline{\mathrm{T}}_{\mathrm{WPS}}$ in three scenarios: *a)* $\mathrm{n} = 100,000$ and $\mathrm{u} = 0.5$; *b)* $\mathrm{n} = 200,000$ and $\mathrm{u} = 0.5$; *c)* $\mathrm{n} = 200,000$ and $\mathrm{u} = 0.1$.

5. Adaptive Path Selection search

The results presented in the previous section highlight that, for any value of m, there are cases in which WPS generates less traffic than MSP, and viceversa. In particular, it can be shown that:

Theorem 3. *The traffic generated by MSP is lower than the traffic generated by WPS when* $\mathrm{s}_{\mathrm{p}_{\min}} < \mathrm{Th}$*, where:*

$$\begin{aligned}\mathrm{Th} = ((\mathrm{m}-1)\cdot(\mathrm{S_H} + \tfrac{1}{2}\cdot(\mathrm{S_H}+\mathrm{S_S})\cdot\log_2\mathrm{n}) + \\ +\mathrm{S_C}\cdot\mathrm{n}\cdot\sum_{\mathrm{i}=1}^{\mathrm{m}}\mathrm{s}_{\mathrm{p_i}} + (2\cdot\mathrm{S_H}+\mathrm{S_Q})\cdot\mathrm{n}\cdot\prod_{\mathrm{i}=1}^{\mathrm{m}}\mathrm{s}_{\mathrm{p_i}})\div \\ \div((\mathrm{S_C}+2\cdot\mathrm{S_H}+\mathrm{S_Q})\cdot\mathrm{n})\end{aligned} \tag{6}$$

Proof. The theorem can be easily proven by solving the inequality $\overline{\mathrm{T}}_{\mathrm{MSP}} < \overline{\mathrm{T}}_{\mathrm{WPS}}$ or, equivalently, $\mathrm{T}_{\mathrm{MSP}} < \mathrm{T}_{\mathrm{WPS}}$. ∎

We exploit this result by defining an *Adaptive Path Selection* (APS) search strategy that resolves a multi-path XML query by choosing, time by time, either the WPS or MSP strategy based on which one minimizes the amount of traffic generated. The path selection is referred to as "adaptive" because it is done on the basis of the characteristics of the query to be resolved. Hence, given a multi-path query, the APS search strategy performs an MSP search if the selectivity of the most selective path is lower than Th. Otherwise, APS performs a WPS search.

Example 3. *Consider a query* Q *composed by four paths* p_1, p_2, p_3 *and* p_4*, submitted in a network with* $\mathrm{n} = 100,000$ *nodes, where the values of parameters* $\mathrm{S_H}$, $\mathrm{S_S}$, $\mathrm{S_Q}$ *and* $\mathrm{S_C}$ *are as introduced in Section 4.3.*
Case 1) *Assume that the four paths have the following selectivity values:* $\mathrm{s}_{\mathrm{p}_1} = 0.001$, $\mathrm{s}_{\mathrm{p}_2} = 0.010$, $\mathrm{s}_{\mathrm{p}_3} = 0.012$ *and* $\mathrm{s}_{\mathrm{p}_4} = 0.020$*. According to Eq. 6,* $\mathrm{Th} = 0.00397$*. Since* $\mathrm{s}_{\mathrm{p}_{\min}} = \mathrm{s}_{\mathrm{p}_1} < \mathrm{Th}$*, APS will perform an MSP search to resolve* Q.
Case 2) *Assume the same selectivity values as above, except for the first path whose value has changed to* $\mathrm{s}_{\mathrm{p}_1} = 0.008$*. According to Eq. 6,* $\mathrm{Th} = 0.00460$*. In this case, since* $\mathrm{s}_{\mathrm{p}_{\min}} > \mathrm{Th}$*, APS will perform a WPS search instead.*

Case 3) *Assume the selectivity of the first path has changed to* $s_{p_1} = 0.010$*, and that of the fourth path has changed to* $s_{p_4} = 0.080$*. According to Eq. 6,* $\mathrm{Th} = 0.0102$*. This is a borderline case, since the lowest selectivity value is very close to the threshold. However, as* $s_{p_{min}} < \mathrm{Th}$*, an MSP search will be performed by APS to resolve* Q*.*

Fig. 7 describes the algorithm performed to process a query Q using the APS search technique, according to the strategy illustrated above.

Algorithm APS_Search
Input: query Q
Output: set of answers $\mathcal{A}$

```
begin
    A ← ∅;
    P ← identifyPaths(Q);
    s_min ← −1;
    p_min ← ⊢;
    foreach path p_i ∈ P do
        if s_{p_i} < s_min then
            s_min ← s_{p_i};
            p_min ← p_i;
        end
    end
    if 0 ≤ s_min < Th then
        A ← MSP_Search(Q,p_min);
    else
        A ← WPS_Search(Q,P);
    end
    return A;
end
```

Figure 7: Adaptive Path Selection search.

The query processing starts by identifying all the paths in Q. Then, the path with the lowest selectivity value (i.e., the most selective path) is identified and, if such value is under a threshold Th, calculated using Eq. 6, an MSP search is performed. Otherwise, a WPS search is executed. According to Theorem 3 and to the APS algorithm shown in Fig. 7, we can introduce the following:

Definition 6. *(APS overhead traffic). The overhead traffic generated by the APS search technique,* $\overline{\mathrm{T}}_{\mathrm{APS}}$*, is given by:*

$$\overline{\mathrm{T}}_{\mathrm{APS}} = \min(\overline{\mathrm{T}}_{\mathrm{WPS}}, \overline{\mathrm{T}}_{\mathrm{MSP}}) \tag{7}$$

5.1. Comparison with WPS and MSP

We evaluated the overhead traffic generated by the APS search compared with WPS and MSP. The evaluation has been performed using the same simulator and methodology described in Section 4.3.

Fig. 8 shows the average amount of traffic generated by the three search techniques for a fixed number of nodes ($n = 100,000$) in three scenarios: *a)* $u = 0.1$; *b)* $u = 0.5$; *c)* $u = 1.0$. As expected, APS always generates less traffic than WPS and MSP. For example,

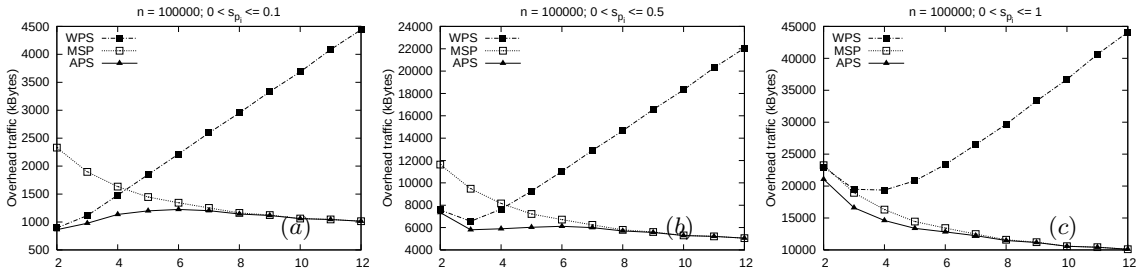

Figure 8: Average values of $\overline{T}_{APS}$, $\overline{T}_{WPS}$ and $\overline{T}_{MSP}$ for a fixed number of nodes (n = 100,000) in three scenarios: *a*) u = 0.1; *b*) u = 0.5; *c*) u = 1.0.

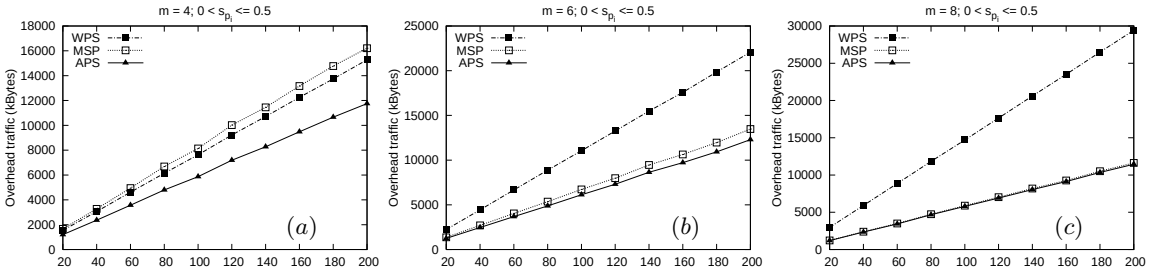

Figure 9: Average values of $\overline{T}_{APS}$, $\overline{T}_{WPS}$ and $\overline{T}_{MSP}$ with n ranging from 20,000 to 200,000 nodes for some selected values of m: *a*) m = 4; *b*) m = 6; *c*) m = 8

in case *b*, APS allowed to save: 72.7 % traffic compared to WPS when m = 12; 41.1 % traffic compared to MSP when m = 2; 18.7 % traffic compared to the best between WPS and MSP when m = 5. Similar trends are obtained for cases *a* and *c*.

After having evaluated the traffic generated for a fixed number of nodes, we compared the traffic generated by APS, WPS and MSP for some selected values of m with n ranging from 20,000 to 200,000 nodes. In particular, Fig. 9 shows the result of this evaluation in three scenarios: *a*) m = 4; *b*) m = 6; *c*) m = 8. The graphs show that the number of messages increases linearly with the number of nodes, with all three search algorithms. However, the slope coefficient of the APS curve is lower than those of the WPS and MSP curves, hence proving a better scalability of APS when the size of the network increases. The advantage of APS over MSP is higher for low values of m while it is marginal for higher values. Conversely, the advantage of APS over WPS is limited for low values of m, but it becomes significant when m increases.

In summary, the results presented above demonstrate that the APS search technique allows to reduce the amount of network traffic generated in all scenarios, by adaptively choosing the best technique to use on the basis of the selectivity of the paths in the query to be processed.

6. Path Selectivity Estimation

Like the MSP search technique, APS needs to know the selectivity of the paths in the query. This requires two issues to be addressed: *i*) defining a space-efficient data structure to store the selectivity of each path in the network; *ii*) devising an effective solution to build and propagate such data structure across the network. We address the first issue by defining a Path Selectivity Table (PST) for XML path selectivity estimation in a distributed scenario, and the second one by defining a PST Construction and Propagation (PST_CP) algorithm that builds the PST in a distributed way and propagates it to all nodes in the network with logarithmic performance bounds. The PST enables local selectivity estimation, i.e., every participant peer can use it to locally estimate the selectivity of a path without querying the network for this purpose.

The remainder of this section is structured as follows. Section 6.1 describes the PST data structure. Section 6.2 describes the PST_CP algorithm for PST construction and propagation and discusses its complexity. Section 6.3 outlines how the PST could be used in holistic approaches.

6.1. Path Selectivity Table

Each node, besides the data structures required by the DHT protocol (i.e., a FT and a KT as shown in Fig. 1), uses a *Path Selectivity Table* (PST) to obtain an estimate of the selectivity of all the paths that form the queries to be processed.

We remind that the selectivity s_p of a path p, as defined by Eq. 1, is a real number in the range $(0, 1]$. Given all the paths indexed in the system, we assume that the path selectivity range $[\min, \max]$, is divided into v intervals, $I_1 ... I_v$, where min (resp. max) is the minimum (resp. maximum) among all the path selectivity values, and $0 < \min \leq \max \leq 1$. Each selectivity interval, I_i, is characterized by three parameters: the lower bound $I_i.l$, the upper bound $I_i.u$, and the average value $I_i.avg$. The set of all the selectivity intervals is denoted as $\mathcal{I}$.

Given a set of XML documents $\mathcal{D}$, distributed across the network, a PST for $\mathcal{D}$ is a table with v rows, one for each selectivity interval. The i-th row of the PST, denoted as PST_i, is a pair $\langle \text{avg}, \text{BF} \rangle$, where $\text{avg} = I_i.avg$ and BF is a w-bit Bloom Filter (BF). $PST_i.BF$ is used to represent all the paths in $\mathcal{D}$ whose selectivity belongs to interval I_i, while $PST_i.avg$ is a representative selectivity value for all such paths.

We recall that a BF is a space-efficient data structure used to test whether an element is a member of a set [37]. An empty BF is an array of w bits, all set to 0. Associated with the BF are z independent hash functions $h_{f_1} ... h_{f_z}$, each one able to hash a data item d into a random number uniformly distributed over the range $[1, w]$. To insert an item d into the BF, all z hash functions are applied to the value of d, and every bit at position $h_{f_i}(d)$ is set to 1, for each $1 \leq i \leq z$. To test whether an item d is in the set represented by the BF, the same z hash functions are applied to d. The response is positive only if every bit at position $h_{f_i}(d)$ is set to 1. An important property of a BF is that false negatives are not possible, while false positives are possible with probability

$$P \approx (1 - e^{-zx/w})^z \tag{8}$$

where x is the number of elements inserted into the BF [39].

For a given w and x, the number of hash functions z_{opt} that minimizes the false positive probability expressed by Eq. 8, is:

$$z_{opt} = \frac{w}{x} \cdot \ln 2 \tag{9}$$

Given x and a desired false positive probability P, and using the optimal number of hash functions z_{opt}, the required number of bits w_{opt}, can be computed by substituting z_{opt} in Eq. 8, which results in:

$$w_{opt} = -\frac{x \cdot \ln P}{\ln^2 2} \tag{10}$$

Given a path $p \in \mathcal{D}$ and a PST for $\mathcal{D}$, we obtain an estimate of s_p as follows. First, we find a tuple PST_i such that $p \in PST_i.BF$, and then we use $PST_i.avg$ as an estimate of s_p. We remind that a path p belongs to a BF if all the bits at position $h_i(p)$ are set to 1, with $1 \leq i \leq z$. Since BFs are subject to false positives, it may happen that p belongs to multiple BFs of the PST. Since in this scenario we cannot know which is the correct BF to take, we minimize the error following the approach in [31] by taking, as an estimate of s_p, the mean of the average values of all the selectivity intervals associated with the BFs to which p belongs to. As the selectivity estimate may be inaccurate in the presence of false positives, we want to design the PST so as to ensure a given false positive rate.

We define *false positive rate* fr for a PST, the probability that a path p belongs to more than one BFs of the PST. Assuming that a path p is contained in one of the v BFs of the PST, the probability fr that at least one of the other $v - 1$ BFs contains p because of a false positive, is given by:

$$fr = 1 - (1 - P)^{(v-1)} \tag{11}$$

where P is the probability of false positives of each BF, assuming that P is the same for all the BFs.

If we want to obtain a specific value of fr for the PST, we must ensure that each BF of the PST will be designed to achieve a false positive rate P given by:

$$P = 1 - (1 - fr)^{1/(v-1)} \tag{12}$$

where Eq. 12 is obtained by inverting Eq. 11.

Given P and the average number x of paths that will be inserted in each BF, we can finally calculate w through Eq. 10, and z through Eq. 9.

Other than establishing the appropriate values for w and z, it is necessary to determine the optimal boundaries for the selectivity intervals. To this end, we proceed following the approach adopted by [31]: first, we collect statistics about the number of paths in the system and their occurrences; then, starting from these statistics, we use a V-Optimal histogram construction algorithm to find the optimal intervals boundaries.

The statistics about paths and their occurrences are gathered in the form of a *Path Count List* (PCL). A PCL is a list of pairs $\langle \#paths, \#nodes \rangle$. A pair $\langle x, y \rangle$ indicates that there are x different paths present in y nodes. To get significant statistics, we select a sample set of nodes, and each of them builds its own PCL based on local information. Then, the PCLs are merged into a single PCL, which provides an estimate of the overall path distribution. Afterwards, we construct a V-Optimal histogram to represent in compact form the PCL data.

V-Optimal histograms are designed to minimize the weighted variance of the source values, i.e., the quantity $\sum_{i=1}^{\beta} x_i V_i$ where x_i is the number of items in the i-th bucket and

V_i is the variance of the source values in the i-th bucket [35]. To efficiently build a V-Optimal histogram for the PCL, we use the algorithm proposed by Jagadish et al. [38], which is based on dynamic programming. This algorithm finds an optimal solution that is time quadratic in the number of paths and linear in the number of buckets.

Once a v-bucket V-Optimal histogram for the PCL has been built using the Jagadish's algorithm, we use the lower bound, average value, and upper bound, of the i-th histogram bucket as I_i.l, I_i.avg, and I_i.u respectively, where I_i is the i-th selectivity interval of the PST.

6.2. PST construction and propagation

The *PST Construction and Propagation* (PST_CP) process is composed of four phases:

1. *Path density estimation.* The goal of this phase is estimating the path density, i.e., the ratio between the number of paths and the number of nodes in the network.
2. *Path distribution estimation.* The goal of this phase is estimating the path distribution, i.e., how many paths belong to the different selectivity intervals.
3. *PST creation.* The goal of this phase is to create the PST starting from the KTs of all the nodes in the network.
4. *PST propagation.* The goal of this phase is to propagate the final PST to all the nodes in the network.

These phases exploit a *Modified Broadcast With Feedback* (MBWF) algorithm introduced in Section 6.2.1. Details about the PST_CP algorithm are provided in Section 6.2.2.

6.2.1. MBWF algorithm

The network structure of a DHT-based system can be exploited to efficiently distribute any kind of information across the overlay that is formed by the interconnection of peer nodes. This principle has been exploited by El-Ansary et al. [32] to design an efficient broadcast algorithm for Chord overlays.

The broadcast algorithm works as follows. The broadcasting node, N_j divides the identifier space of the Chord ring into f parts, where f is the number of unique fingers in its FT, and delegates to each finger the responsibility to cover all nodes associated with its part. In particular, the last unique finger, F_f, is delegated to cover all nodes in the interval (F_f, N_j); F_{f-1} is delegated to cover (F_{f-1}, F_f), and so on down to the first finger, F_1, which is delegated to cover (F_1, F_2). Each delegatee divides the part it is responsible for into parts that it in turn delegates to other nodes, recursively. In a network of n nodes, a broadcast message originating at an arbitrary node reaches all other nodes in $O(\log_2 n)$ hops with $n-1$ messages.

In [33], Ghodsi proposed an extension of the broadcast algorithm, referred to as *Broadcast With Feedback* (BWF), to address the case of a broadcasting node that wishes to receive a response from the nodes it is broadcasting to. In our framework we define a slightly modified version of BWF algorithm, called *Modified Broadcast With Feedback* (MBWF) which, differently from BWF, permits to limit the broadcast to a subset of nodes and to choose whether the feedback is needed or not. To this purpose, beyond the message to be broadcast, msg, MBWF allows the broadcasting node, N_j, to specify two additional parameters:

- a boolean fdbck, indicating whether the feedback is required or not.
- an integer last $\in [1, f]$, whose value represents the index of the last unique finger to be contacted from those in N_j's FT.

The value of last permits to limit the broadcast to a subset of nodes. If last = f, the message will be broadcast to all nodes, like in the original BWF algorithm. If last < f, the message will be broadcast (or, more properly, multicast) only to those nodes whose identifier lie in the range $[F_1, F_{last}]$.

Fig. 10 shows the pseudo-code of the MBWF algorithm. Broadcast is a local procedure invoked by a node N_j to start the broadcasting process. The procedure receives the three input parameters introduced above (msg, fdbck, last) and, if the feedback is required, returns a single cumulative feedback response (resp) from the nodes that received msg. BroadcastForward and BroadcastReply are remotely invokable procedures that must be present on every node, including the broadcast initiator.

According to [34], given the structural properties of the spanning tree associated with the broadcast process, the number of nodes n_x reached through the first x fingers of the broadcast initiator can be estimated as:

$$n_x = k \cdot 2^x \tag{13}$$

where k is a constant valid for every $1 \leq x \leq f$.

Moreover, the number of hops (i.e., broadcast forwarding operations) to reach the farthest node among those covered by the first x fingers can be estimated as:

$$d_x = \log_2 n_x \tag{14}$$

6.2.2. PST_CP algorithm

The PST_CP algorithm, shown in Fig. 11, is executed periodically by one node in the system, for example the one having the smallest identifier[4], referred to as N_{start}.

The algorithm receives the following input parameters (Section 7 will discuss which values can be in practice used for these parameters):

- nf. This parameter is used during the path density estimation phase. Such estimation is performed by N_{start} by querying all nodes that can be reached by executing a broadcast limited to its first nf fingers. Therefore, the value of nf determines the number of nodes that are involved in the path density estimation phase.

- mp. This parameter specifies the minimum number of paths needed to estimate the path selectivity distribution. Given the number of paths counted during the path density estimation phase, N_{start} determines the broadcast configuration to be executed during the path distribution estimation phase to reach a number of nodes cumulatively indexing at least mp paths.

- fr and v. The former represents the desired false positive rate of the PST; the latter represents the number of rows of the PST. Given fr and v, and the statistics gathered during the path distribution estimation phase, N_{start} can determine the parameters $\mathcal{I}$, w, and z of the PST to be constructed, as detailed below.

The algorithm is based on the exchange of messages listed in Table 2.

[4]In Chord, a node can know whether it is the node with the smallest id, by looking if its predecessor has a greater id.

Algorithm Modified Broadcast With Feedback (MBWF)
Input: Message to be broadcast: msg
Feedback required: fdbck
Last finger to be contacted: last $\in [1, f]$
Output: Feedback response: resp

```
procedure Broadcast(msg, fdbck, last)
    resp ← ⊢;
    finish ← false;
    N_j.BroadcastForward(msg, N_j, fdbck, last);
    if fdbck = true then
        wait until finish = true;
        return resp;
    end if
end procedure

procedure BroadcastForward(msg, limit, fdbck, last) invoked by N_k
    resp ← Process(msg);
    par ← N_k;
    Ack ← Ø;
    for i ← f downto 1 do
        if F_i ∈ (N_j,limit) then
            if i ≤ last then
                F_i.BroadcastForward(msg, limit, fdbck, f);
                Ack ← Ack ∪ F_i;
            end if
            limit ← F_i;
        end if
    end for
    if Ack = Ø and fdbck = true then
        par.BroadcastReply(resp);
    end if
end procedure

procedure BroadcastReply(r) invoked by N_k
    if N_k = N_j then
        finish ← true;
    else
        Ack ← Ack − N_k;
        resp ← Merge(resp, r);
        if Ack = Ø then
            par.BroadcastReply(resp);
        end if
    end if
end procedure
```

Figure 10: Modified Broadcast With Feedback algorithm.

Algorithm PST Construction and Propagation (PST_CP)
Input: Number of fingers contacted on first phase: nf $\leq$ f
Min. number of paths for selectivity estimation: mp > 0
Desired false positive rate: fr $\in (0, 1)$
Number of selectivity intervals: v > 0
Output: Updated PST on every node

```
begin
    // Path density estimation:
    last ← nf;
    msg1 ← PC_Request[];
    resp1 ← Broadcast(msg1, true, last);
    msg2 ← ComputeSampleSize(resp1);
    // Path distribution estimation:
    resp2 ← Broadcast(msg2, true, last);
    msg3 ← ComputePSTParameters(resp2);
    // PST creation:
    resp3 ← Broadcast(msg3, true, f);
    msg4 ← UpdateLocalPST(resp3);
    // PST propagation:
    Broadcast(msg4, false, f);
end
```

Figure 11: PST Construction and Propagation algorithm.

Table 2: Messages exchanged in the PST_CP algorithm.

Message	**Fields**	**Field Description**
PC_Request	(none)	
PC_Response	pc	Path counter
PCL_Request	(none)	
PCL_Response	PCL	Path Count List
	pc	Path counter
	nc	Node counter
PST_Request	$\mathcal{I}$	Set of selectivity intervals
	w	No. of bits of each BF
	z	No. of hash functions of the BFs
	$\hat{n}$	Estimated number of nodes
PST_Response	PST	Path Selectivity Table
PST_Broadcast	PST	Path Selectivity Table

To activate the *path density estimation* phase, N_{start} performs a broadcast-with-feedback to a subset $\mathcal{S}_1$ of the nodes in the network (those reachable through $[F_1, F_{nf}]$, i.e., with last = nf), asking them to reply with the number of paths indexed in their KTs. The broadcast request is represented by a PC_Request message (msg1), while the response is represented by a PC_Response message. The cumulative feedback response resp1 received by N_{start} contains the total number pc of paths in the key tables of $\mathcal{S}_1$, which provides an estimate of the path density in the whole network. Response resp1 is processed by a ComputeSampleSize procedure that, given pc and the input parameter mp, calculates the new last value and returns a PCL_Request message that will be used in the next phase.

The *path distribution estimation* phase performs a broadcast-with-feedback to all nodes $\mathcal{S}_2$ reachable through $[F_1, F_{last}]$ (with last as resulting from the previous phase). In response to the broadcast request, represented by the PCL_Request message (msg2), those nodes reply with the distribution of paths in their KTs, in the form of a PCL_Response message. Therefore, N_{start} will receive, as a feedback, a cumulative resp2 containing a PCL together

with two values, pc and nc, representing the total number of paths indexed in the KTs of nodes in $\mathcal{S}_2$, and the total number of such nodes, respectively. Response resp2 is processed by a ComputePSTParameters procedure that, given the cumulative PCL, nc and pc, and the input parameters fr and v, determines the PST parameters ($\mathcal{I}$, w, and z) and returns a PST_REQUEST message that will be used in the next phase.

The goal of the *PST creation* phase is to create the PST starting from the KTs of all nodes in the network. To this end, N_{start} performs a broadcast-with-feedback to all nodes in the network (i.e., last = f) of the PST_REQUEST message (msg3). Each node will reply with a PST_RESPONSE message, containing the PST calculated locally on the basis of the parameters $\mathcal{I}$, w, and z included in msg3. As a result of the recursive merging procedure of the broadcast-with-feedback, the PST received by N_{start} will contain path selectivity information of all the paths in the network. A UpdateLocalPST procedure is then invoked by N_{start} to locally store the final PST, and to generate a PST_BROADCAST.

The *PST propagation* phase performs a broadcast-without-feedback of the final PST to all nodes in the network (i.e., last = f) of the PST_BROADCAST message (msg4) generated during the previous phase.

Procedures ComputeSampleSize, ComputePSTParameters and UpdateLocalPST

The ComputeSampleSize procedure, shown in Fig. 12, receives a PC_RESPONSE message and returns a PCL_REQUEST message. As a side effect, it also modifies the value of the last variable. The goal is calculating a new value of last such that the broadcast that will be performed during the path distribution estimation phase will reach a number of nodes cumulatively indexing at least mp paths.

```
procedure ComputeSampleSize(msg)
    old_last ← last;
    last ← Min(⌈log2(mp · 2^old_last/msg.pc)⌉, f);
    msg ← PCL_REQUEST[];
    return msg;
end procedure
```

Figure 12: ComputeSampleSize procedure.

Accordingly to Eq. 13, the size of set $\mathcal{S}_1$, which includes all nodes reached by N_{start} through $[F_1, F_{old\_last}]$ during the path density estimation phase, can be calculated as follows:

$$|\mathcal{S}_1| = k \cdot 2^{old\_last} \tag{15}$$

Therefore, the path density can be estimated as:

$$\rho = msg.pc/|\mathcal{S}_1| \tag{16}$$

where msg.pc is the total number of paths indexed by the nodes in $\mathcal{S}_1$.

To find mp paths, we need to contact a set of nodes $\mathcal{S}_2$ such that $|\mathcal{S}_2| = mp/\rho$. As $|\mathcal{S}_2| = k \cdot 2^{last}$, the value of last to reach $|\mathcal{S}_2|$ nodes is given by:

$$last = \lceil \log_2(mp \cdot 2^{old\_last}/msg.pc) \rceil \tag{17}$$

Since the value of last resulting from Eq. 17 may be greater than the number of fingers, f, the actual value of last will be the minimum between the one calculated by Eq. 17 and f, as shown in Fig. 12.

The ComputePSTParameters procedure, shown in Fig. 13, receives a PCL_RESPONSE message msg and returns a PST_REQUEST. msg contains the PCL, pc and nc resulting from the path distribution estimation phase.

```
procedure ComputePSTParameters(msg)
    n̂ ← msg.nc · 2^(f−last);
    I ← VOPTHistIntervals(msg.PCL, n̂, v);
    p̂ ← msg.pc · 2^(f−last);
    x ← p̂/v;
    P ← 1 − (1 − fr)^(1/(v−1));
    w ← ⌈−x · ln P / ln² 2⌉;
    z ← ⌈w/x · ln 2⌉;
    msg ← PST_REQUEST[I, w, z, n̂];
    return msg;
end procedure
```

Figure 13: ComputePSTParameters procedure.

The procedure begins by estimating the total number of nodes in the network n̂. From Eq. 13, n̂ can be estimated as $msg.nc \cdot 2^{f-last}$, where f is the total number of fingers, and last is the number of fingers that have been contacted during the path distribution estimation phase.

The VOPTHistIntervals procedure receives the PCL, n̂ and v, and returns a set $\mathcal{I}$ of v selectivity intervals. Then, the total number of paths in the network p̂ is estimated as $msg.pc \cdot 2^{f-last}$, similarly to n̂, and the average number x of paths that will be inserted in each BF is calculated as p̂/v.

In order to determine the PST parameters that ensure the desired false positive rate fr for the PST, the algorithm calculates P using Eq. 12, w through Eq. 10, and z through Eq. 9. Note that, since the values returned by the last two equations are in the real domain, we approximate w and z to the next higher integer.

Finally, the UpdateLocalPST procedure, shown in Fig. 14, receives a PST_RESPONSE message containing the cumulative PST generated by all nodes in the network, locally stores a copy of such PST, and returns a PST_BROADCAST that will be used during the PST propagation phase to distribute the cumulative PST to all the nodes in the network.

```
procedure UpdateLocalPST(msg)
    PST ← msg.PST;
    msg ← PST_BROADCAST[PST];
    return msg;
end procedure
```

Figure 14: UpdateLocalPST procedure.

Procedures Process and Merge

The Process procedure, invoked by BroadcastForward, specifies the actions performed by each node on reception of a message (see Fig. 15). In response to a PC_REQUEST, the procedure returns a PC_RESPONSE specifying the number of paths, pn, which are present in the local KT. When a PCL_REQUEST is received, a local PCL is created by invoking the BuildLocalPCL procedure, described later; then, the procedure returns a PCL_RESPONSE, containing the localPCL, the path counter pc (initially set to pn), and the number of contacted

nodes nc (initially set to 1 to count the current node). The reception of a PST_REQUEST triggers the creation of a local PST by invoking the BuildLocalPST procedure, described later, starting from the parameters included in the request; then, the procedure returns a PST_RESPONSE containing the localPST. Finally, in the case a PST_BROADCAST is received, the node just updates its local PST to the value of the PST parameter received with the message.

```
procedure Process(msg)
   if msg instanceof PC_REQUEST then
      resp ← PC_RESPONSE[pn];
      return resp;
   else if msg instanceof PCL_REQUEST then
      localPCL ← BuildLocalPCL();
      resp ← PCL_RESPONSE[localPCL, c, 1];
      return resp;
   else if msg instanceof PST_REQUEST then
      localPST ← BuildLocalPST(msg.I, msg.w, msg.z, msg.n̂);
      resp ← PST_RESPONSE[localPST];
      return resp
   else if msg instanceof PST_BROADCAST then
      PST ← msg.PST;
   end
end procedure
```

Figure 15: Process procedure.

The Merge procedure, invoked by BroadcastReply, specifies how the local response of a node (generated by Process) is merged with the response received by another node (see Fig. 16). Two PC_RESPONSE messages, msg1 and msg2, produce a new PC_RESPONSE whose pc counter is the sum of msg1.pc and msg12.pc. In the case of two PCL_RESPONSE messages, the procedure returns a new PCL_RESPONSE containing PCL, pc and nc fields obtained by merging the corresponding fields carried by msg1 and msg2; note that the two PCLs are merged by invoking the MergePCLs procedure, described later. Finally, two PST_RESPONSE messages are merged into a new PST_RESPONSE message whose PST field is obtained by merging msg1.PST and msg2.PST, by invoking the MergePSTs procedure, described later.

Procedures BuildLocalPCL and MergePCLs

The BuildLocalPCL procedure is invoked by Process to construct a PCL from the local KT. We remind that the PCL associated with a KT is a list of pairs of the form $\langle$#paths, #nodes$\rangle$. Each pair $\langle$x, y$\rangle$ indicates that the KT contains x different paths owned by y nodes.

Fig. 17 shows an example of PCL construction starting from a local KT. The PCL in the example contains three entries:

- $\langle 1, 1\rangle$, since the KT contains 1 path (p11) owned by 1 node;
- $\langle 2, 2\rangle$, as the KT contains 2 paths (p3 and p17) owned by 2 nodes;
- $\langle 2, 3\rangle$, since the KT contains 2 paths (p1 and p7) with 3 owners;

MergePCLs is invoked by the Merge procedure to merge two PCLs, PCL1 and PCL2. The new PCL is obtained as follows:

```
procedure Merge(msg1, msg2)
    if msg1 instanceof PC_Response then
        pc ← msg1.pc + msg2.pc;
        resp ← PC_Response[pc];
        return resp
    else if msg1 instanceof PCL_Response then
        localPCL ← MergePCLs(msg1.PCL, msg2.PCL);
        pc ← msg1.pc + msg2.pc;
        nc ← msg1.nc + msg2.nc;
        resp ← PCL_Response[PCL, pc, nc];
        return resp
    else if msg1 instanceof PST_Response then
        localPST ← MergePSTs(msg1.PST, msg2.PST);
        resp ← PST_Response[PST];
        return resp;
    end
end procedure
```

Figure 16: Merge procedure.

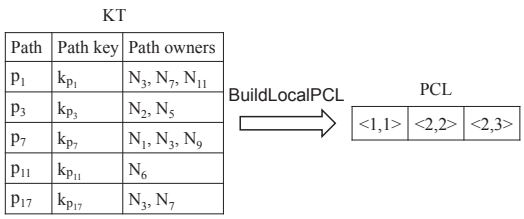

Figure 17: Example of PCL generated by BuildLocalPCL.

- Each pair $\langle x1, y1\rangle$ in PCL1 (resp. PCL2) generates an identical pair in the new PCL if does not exist another pair $\langle x2, y2\rangle$ in PCL2 (resp. PCL1) such that $y1 = y2$.
- Two pairs $\langle x1, y1\rangle$ and $\langle x2, y2\rangle$, present in PCL1 and PCL2 respectively, generate a single pair $\langle x1 + x2, y1\rangle$ in the new PCL only if $y1 = y2$.

Procedures BuildLocalPST and MergePSTs

The BuildLocalPST procedure, described in Fig. 18, is responsible for the construction of the local PST on a given node. It is invoked by Process when a PST_Request[$\mathcal{I}$, w, z, $\hat{n}$] is received.

The first step is the construction of the PST as a table of pairs $\langle avg, BF\rangle$. More precisely, the i-th row of a PST, denoted as PST_i, is a pair $\langle avg, BF\rangle$ where $avg = I_i.avg$ and BF is a w-bit BF used to test whether a path has a selectivity in interval I_i. We assume that all nodes share a suitable number of independent hash functions, and that the first z of them are used for the BF operations.

The next step is identifying to which selectivity interval belongs the selectivity value of each path p in the KT of the node and, thus, inserting path p into the BF associated with that selectivity interval. To this aim, for each path p in the KT, the estimated selectivity value s_p is calculated. At this point, three different cases may hold: **1)** If s_p is lower than the average selectivity of the first bucket of the PST, path p is inserted into the BF associated with the first selectivity interval. **2)** If s_p is greater than the average selectivity of the last bucket of the PST, path p is inserted into the BF associated with the last selectivity

```
procedure BuildLocalPST(I, w, z, n̂)
    for each interval I_i ∈ I do
        avg ← I_i.avg;
        BF ← BloomFilter(w, z);
        PST_i ← ⟨avg, BF⟩;
    end
    for each path p ∈ KeyTable do
        s_p ← n_p/n̂;
        if (s_p < PST_1.avg) then
            PST_1.BF.InsertPath(p);
        end
        else if (s_p > PST_v.avg) then
            PST_v.BF.InsertPath(p);
        end
        else
            for i ← 1 to v − 1 do
                if (s_p ≤ PST_{i+1}.avg) then
                    if ((s_p − PST_i.avg) < (PST_{i+1}.avg − s_p)) then
                        PST_i.BF.InsertPath(p)
                    end
                    else
                        PST_{i+1}.BF.InsertPath(p)
                    end
                end
            end
        end
    end
    return PST
end procedure
```

Figure 18: BuildLocalPST procedure.

interval. **3)** In the other cases, path p is assigned to the BF associated with the selectivity interval whose average value is the closest one to s_p.

MergePSTs (see Fig. 19) is invoked by the Merge procedure to merge two PSTs, PST1 and PST2, received in input. As a first step, the new PST is obtained by cloning one of the two PSTs in input (e.g., PST1). Then, a union operation is performed between each BF of such PST and the BF in the same row of PST2. Since the BFs have the same size and share the same set of hash functions, the union is simply performed with a bitwise or operation.

```
procedure MergePSTs(PST1, PST2)
    PST ← PST1;
    for i ← 1 to v do
        for j ← 1 to w do
            PST_i.BF_j ← PST_i.BF_j or PST2_i.BF_j;
        end
    end
    return PST;
end procedure
```

Figure 19: MergePSTs procedure.

6.2.3. Complexity of PST_CP

Theorem 4. *The PST_CP algorithm constructs and propagates a PST in* $O(\log_2 n)$ *hops, where* n *is the number of nodes in the network.*

Proof. The total number of hops, H, performed during the four phases of the PST_CP algorithm, is given by $\sum_{i=1}^{4} H_i$, where H_1, H_2, H_3, and H_4 are the number of hops performed during the *path density estimation*, *path distribution estimation*, *PST creation*, and *PST propagation* phases, respectively. We remind that each one of the four phases performs either a broadcast-without-feedback or a broadcast-with-feedback. A broadcast-without-feedback is performed in $O(\log_2 n)$ hops [32]. The same asymptotic complexity holds to a broadcast-with-feedback, since it requires exactly twice the number of hops of a broadcast-without-feedback [33]. Therefore, $H_i = O(\log_2 n)$ for any $1 \leq i \leq 4$, and consequently $H = O(\log_2 n)$. ∎

Theorem 5. *The PST_CP algorithm generates* $O(n)$ *messages to construct and propagate a PST, where* n *is the number of nodes in the network.*

Proof. This can be easily proven taking into account that a single broadcast-without-feedback generates exactly $n - 1$ messages [32], which double in the case of a broadcast-with-feedback [33]. Since each one of the four PST_CP phases performs either a broadcast-without-feedback or a broadcast-with-feedback, the total number of messages generated at the end of the four phases is $O(n)$. ∎

6.3. On the use of the PST in holistic approaches

The PST approach could be used in other systems where it is necessary to store information about selectivity of XML data, including the ones that follow a holistic query evaluation approach. In addition, the PST Construction and Propagation (PST_CP) algorithm could be adapted to these systems to construct a PST-like data structure in a distributed way and to propagate selectivity estimations all over the network.

As an example, let us consider the TreeSketch holistic approach [20] introduced in Section 2.1. TreeSketch builds a synopsis which allows to estimate the selectivity of the tree pattern queries in a given XML dataset. In a distributed DHT-based setting, we can envision the following scenario:

- The dataset $\mathcal{D}$ is distributed over the DHT nodes;
- The tree pattern queries in $\mathcal{D}$ are indexed in the DHT, by assigning each of them to a given DHT node through hashing;
- The PST_CP algorithm is used to build and propagate a PST containing selectivity information about tree pattern queries, instead of single paths, as detailed in the following.

Let $\mathcal{D}$ be an XML dataset, synopsis($\mathcal{D}$) the TreeSketch synopsis of $\mathcal{D}$ and q a tree pattern query in $\mathcal{D}$. We can define the estimated selectivity of q, denoted s_q, as the TreeSketch estimated selectivity.

As in the original design, the PST is a table with v rows, one for each selectivity interval. The i-th row of the PST, denoted as PST_i, is a pair $\langle avg, BF \rangle$, where avg is the average of

the i-th selectivity interval and BF is a w-bit Bloom Filter (BF). In a TreeSketch scenario, PST_i.BF can be used to represent all the tree pattern queries in $\mathcal{D}$ whose selectivity belongs to the i-th selectivity interval, while PST_i.avg is a representative selectivity value for all such tree pattern queries.

7. PST performance

This section evaluates the accuracy of PSTs for path selectivity estimation and their effectiveness in supporting APS-based query processing.

7.1. Accuracy evaluation

We experimentally evaluated how the accuracy of the estimates produced by a PST varies in function of the PST_CP input parameters. As a measure of a PST accuracy we adopt the *Average Relative Error* (ARE):

$$\mathrm{ARE} = \frac{1}{t} \cdot \sum_{i=1}^{t} \frac{|s_{p_i} - \hat{s}_{p_i}|}{s_{p_i}} \tag{18}$$

where t is the number of paths in the system, s_{p_i} is the real selectivity of a path p_i, and $\hat{s}_{p_i}$ is the selectivity estimate of p_i returned by the PST.

The evaluation has been carried out using a custom network simulator. The simulator builds the network in three steps:

1. A random Chord network with n nodes is created; as a result of this step, the FTs of all nodes are initialized.
2. t different paths are created; to each path p, a selectivity s_p is assigned in the range $(0, u]$.
3. Each path p is assigned to $s_p \cdot n$ nodes; as a result of this step, the KTs of all nodes are initialized.

After having built the network, the simulator runs the PST_CP algorithm, based on the four input parameters described in Section 6.2: nf, mp, fr, and v. As soon as the PST has been created and propagated to all nodes in the network, the simulator calculates the ARE associated with the PST.

We carried out several experiments aimed at evaluating how the ARE is influenced by the input parameters of the PST_CP algorithm. Out of the four input parameters, we found that the first two ones (nf and mp), are not critical. After some preliminary tests, we chose $nf = 7$ and $mp = 10,000$, which have been used in all the subsequent experiments. Additionally, all the experiments have been performed on a network composed by $n = 100,000$ nodes, with $t = 200,000$ paths, and selectivity values in the range $(0, 0.5]$.

Fig. 20 shows the ARE of a PST resulting from different numbers of selectivity intervals (v) and false positive rates (fr). In particular, Fig. 20*a* shows how the ARE varies when v passes from 10 to 100, for three selected values of fr (0.001, 0.005, 0.01), while Fig. 20*b* shows how the ARE varies when fr increases from 0.001 to 0.01, for three values of v (10, 50, 100).

The results in Fig. 20*a* show that, for a given fr, the ARE is inversely proportional to v. This depends on the fact that the higher the number of selectivity intervals, the lower their

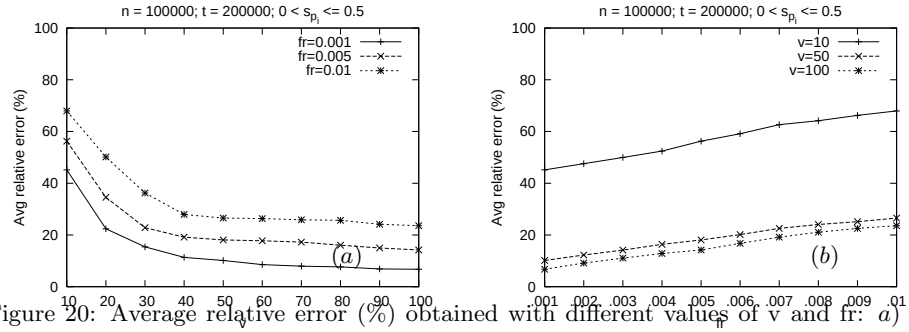

Figure 20: Average relative error (%) obtained with different values of v and fr: *a)* v ranging from 10 to 100; *b*) fr ranging from 0.001 to 0.01.

average width. Since the estimated selectivity of a path p is equal to the average value of the selectivity interval to which p belongs to, smaller-width selectivity intervals results in lower approximation errors. Fig. 20*b* shows that, fixed v, the ARE is directly proportionally to fr. This is due to the fact that the higher the false positive rate, the higher the number of paths belonging to multiple BFs of the PST. In these cases the average error increases, because the selectivity of a path p is estimated as the mean of the average values of all the selectivity intervals associated with the BFs to which p belongs to.

We carried out some additional simulations to evaluate the average overhead traffic generated by the APS search technique, assuming that the selectivity values of all the paths extracted from the query to be processed are estimated using the PST. The average overhead traffic is estimated using the same methodology adopted in Section 5.1 to estimate $\overline{\mathrm{T}}_{\mathrm{APS}}$. However, while in Section 5.1 we assumed that the APS algorithm knows in advance the real selectivity values of all the paths (scenario hereafter referred to as *ideal APS*), here we assume that APS can get only an estimate of such values using the PST. The difference between the PST-based APS overhead traffic and that of the *ideal APS* allows us to evaluate the effectiveness of the PST in supporting APS search.

Fig. 21 shows the PST-based APS overhead traffic generated in two scenarios: *a)* v=10, 50, 100, with fr=0.001; *b*) fr=0.001, 0.005, 0.01, with v=100. The overhead traffic is evaluated for queries with a number of paths m from 2 to 12. For comparison purposes, Fig. 21*a* and 21*b* show also the overhead traffic generated by the WPS algorithm, and by the APS algorithm using the real selectivity values of all the paths (*ideal APS*).

Fig. 21*a* shows that, fixed the false positive rate, the overhead traffic decreases by increasing the value of v. For v=100, the PST-based overhead traffic is close to that generated by the ideal APS. For lower values of v the overhead traffic slightly increases, according to the fact that also the ARE increases by using a lower number of selectivity intervals (see Fig. 20*a*). From Fig. 21*b* we can observe that, fixed the number of selectivity intervals, the overhead traffic decreases by choosing lower fr values. Also in this case, as one might expect, the overhead traffic follows the same trend of the ARE (see Fig. 20*b*), i.e., the lower the average relative error of a PST, the lower the overhead traffic resulting from the use of the PST to perform an APS search.

We conclude by observing that even when we choose relatively low values for v or high values for fr, the PST-based overhead traffic resulted significantly lower than that generated

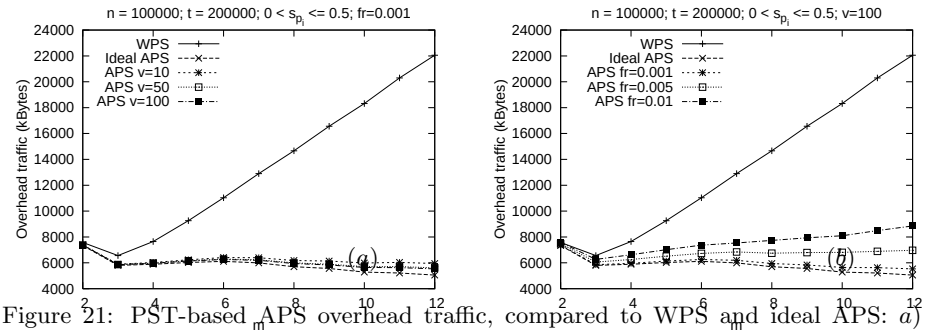

Figure 21: PST-based APS overhead traffic, compared to WPS and ideal APS: *a*) v=10, 50, 100, with fr=0.001; *b*) fr=0.001, 0.005, 0.01, with v=100.

by a WPS search, thus confirming the efficiency of the PST in supporting APS search in a distributed scenario.

7.2. Effect of churn

The results presented above are computed in a static scenario, i.e., in a network where nodes do not join and leave the system and data does not change over the time. This implies that the PST, once it is constructed and propagated, constantly ensures the same level of performance since it is always updated and fully available to all nodes in the network. To evaluate the system in dynamic conditions, we present here an additional set of simulations in which joins and leaves of nodes, as well as query submissions, are modeled as Poisson processes; therefore, the inter-arrival times of all the join, leave and query submission events are independent and obey an exponential distribution with a given rate. This model has been adopted in literature to evaluate several P2P systems (see, for example, [1]), for its ability to approximate real network dynamics reasonably well.

To simulate node churn, we defined a joining rate JR and a leaving rate LR. On average, every minute JR nodes join the network and LR nodes leave the network. In our simulation JR = LR to keep the total number of nodes approximatively constant during the whole simulation. In particular, we used three values for JR and LR: 1, 5 and 10, so as to evaluate the system under different churn rates. The stochastic behavior of queries is determined by a submission rate SR, which represents the average number of queries submitted every minute to the system. Seven values for SR have been considered: 100, 200, 400, 800, 1,600, 3,200 and 6,400. In addition, each query is characterized by a number of paths uniformly distributed between 2 and 12.

As for the above experiments, the network is initialized with n=100,000 nodes and t=200,000 paths. Then, the PST_CP algorithm is executed with the following input parameters: nf=7, mp=10,000, fr=0.001 and v=50. After the PST is constructed and propagated, the simulation proceeds for 24 hours, during which nodes join, leave and submit queries at the average rates introduced earlier. The PST_CP algorithm is never re-executed after the beginning of the simulation; this has two implications:

- The newly joined nodes do not possess a PST[5]. Therefore, when such nodes wants to

[5]This is a conservative assumption, in order to evaluate the effect of dynamism in the worst case scenario.

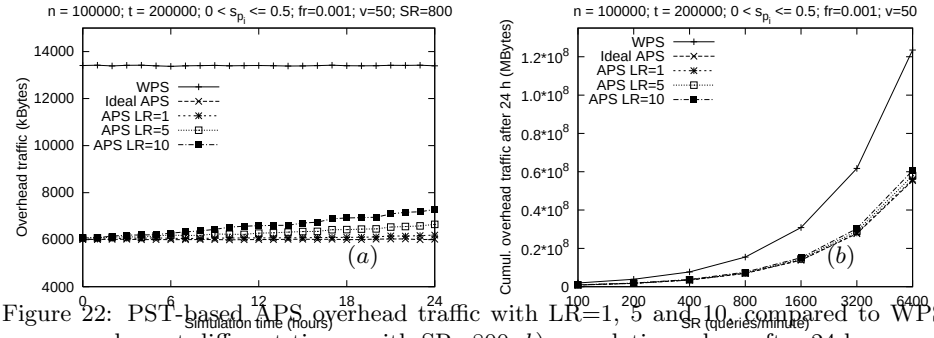

Figure 22: PST-based APS overhead traffic with LR=1, 5 and 10, compared to WPS and ideal APS: *a*) average values at different times, with SR=800; *b*) cumulative values after 24 hours, with SR ranging from 100 to 6400.

perform an APS search, they are forced to execute a WPS search instead.

- The nodes that did not leave the network possess a PST that gets gradually outdated whenever new nodes, and associated data, join the network. If the PST does not contain information about a path p, the average of all the selectivity values present in the PST is returned as an estimate of p's selectivity.

Figure 22(a) shows the average values of the PST-based APS overhead traffic, compared to WPS and ideal APS, in a dynamic network characterized by the system parameters introduced above, with LR=1, 5 and 10, and SR=800. The values are calculated at the end of each hour of simulation, so as to observe the effect of dynamism over the time. The figure shows that, while the overhead traffic of WPS and ideal APS remains approximatively constant over the time (around 13,400 kB and 6,000 kB, respectively), the average traffic generated by the PST-based APS increases linearly over the time. The increase strongly depends from LR: with the lowest churn level (LR=1) the performance degradation is negligible, since the overhead traffic passes from 6,021 kB at the beginning of the simulation, to 6,196 kB after 24 hours. In contrast, with the highest churn level (LR=10), the overhead traffic passes from 6,071 kB to 7,289 kB in 24 hours. Even in this case, however, the amount of traffic generated by APS is significantly lower than that produced by WPS.

To better highlight the last point, Figure 22(b) shows the cumulative overhead traffic generated during 24 hours by WPS, ideal APS, and PST-based APS, with LR=1, 5 and 10, and SR ranging from 100 to 6,400. The figure shows that the cumulative overhead traffic increases proportionally with the query submission rate with all the search techniques. However, the absolute amount of traffic saved by APS compared to WPS increases significantly with the submission rate, despite the slight increase of traffic (in all the cases lower than 8%) registered by APS when the leaving rate passes from 1 to 10.

7.3. Validation on a Cloud environment

We made an experimental evaluation of the system in a real network to validate the simulation results. The experiments have been performed on the Microsoft Azure cloud

Alternatively, we could have assumed that a newly joined node gets a copy of the PST from one of its neighbors in the DHT overlay.

Table 3: PST size (kBytes) as a function of fr and v.

	v=10	v=20	v=30	v=40	v=50	v=60	v=70	v=80	v=90	v=100
fr=0.01	12.2	13.6	14.3	14.9	15.3	15.6	15.9	16.1	16.3	16.5
fr=0.005	13.5	14.8	15.6	16.1	16.5	16.8	17.1	17.4	17.6	17.8
fr=0.001	16.4	17.7	18.5	19.0	19.4	19.7	20.0	20.3	20.5	20.7

platform using 128 servers (virtual machines), each one equipped with a single-core 1.66 GHz CPU, 1.75 GB of RAM, and 225 GB of disk space. For the construction of the Chord overlay we used Open Chord, an implementation of the Chord algorithm by the University of Bamberg[6].

We managed to run 16 Chord nodes on each server, thus we were able to test the system in a network composed by n=2048 nodes. The number of 16 nodes per server was determined experimentally as the maximum number of Chord nodes that was possible to run concurrently on a single server, given the amount of RAM available on each server.

For the experiments we used the TreeBank dataset[7], which contains t=7073 distinct paths, with a size of 86 MB. In order to create a distributed data scenario, we proceeded as follows:

1. From the original dataset, D_0, we generated n smaller datasets, D_1...D_n, each one containing between 5 and 15 percent of the instances of D_0, taken randomly.
2. Each one of the datasets D_1...D_n was randomly assigned to one of the n Chord nodes.
3. Each one of the t paths was assigned, through hashing, to the KT of a Chord node.

After having built the network, we measured an average of 3.45 distinct paths per KT, and path selectivity values ranging between 0.01 and 0.37.

As a first set of experiments, we measured the *Average Relative Error* (ARE) of a PST in estimating the selectivity values of all the t paths indexed in the network. The ARE has been measured using different PSTs, which differ for the false positive rate (fr=0,001, 0,005 or 0,01) and the number of selectivity intervals (from v=10 to v=100). In all cases, the other PST_CP parameters were nf=7 and mp=5,000. Table 3 reports the size of the different PSTs used in these experiments, as a function of fr and v.

The experimental results are reported in Fig. 23, which compares the measured values (dashed line) with the simulated values (continuous lines). The latter ones were obtained by simulating a system with the same parameters (number of nodes, number of paths, selectivity ranges and PST parameters) of the real network. As shown in the graphs, the measured ARE values are very close to the simulated ones, particularly for v greater than 50 in all three scenarios. In addition, by comparing Fig. 23*a* with Fig. 23*c*, we can observe the advantage of decreasing by a factor ten the fr value (from 0.01 to 0.001), which results in a significant decrease of the ARE values, with a negligible increase of the PST size (4 kBytes on average, as shown in Table 3).

As a second set of experiments, we measured the PST-based APS overhead traffic generated using three PSTs with fr equal to 0.01, 0.005 and 0.001, and v fixed to 100. The traffic has been measured considering a number of paths m ranging from 2 to 12.

[6]Open Chord. http://open-chord.sourceforge.net.

[7]The PENN Treebank Project, http://www.cis.upenn.edu/treebank

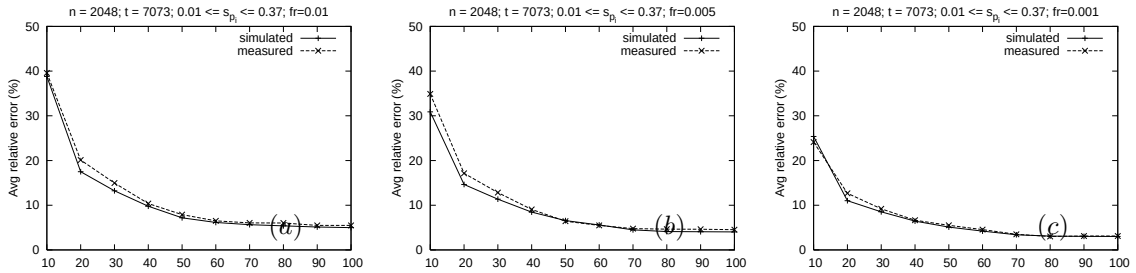

Figure 23: Measured vs simulated average relative error (%) obtained with v ranging from 10 to 100, with three values of fr: *a*) fr=0.01; *b*) fr=0.005; *c*) fr=0.001.

The results are reported in Fig. 24. The three graphs compare the traffic measured in the real network with that resulting by simulating a network with identical parameters. We registered a maximum difference of 7% between simulated and measured values with the first two PSTs (fr=0.01 and 0.005) particularly in correspondence of m=2 and 3. With the third PST (fr=0.001), the difference between measured and simulated values was below 3% for any value of m. Also in this case, we can observe the advantage of using a PST with fr=0.001, which is just 4kBytes larger than that with fr=0.01, but allowed us to reduce by several kBytes the average traffic generated by each single query (e.g., 13 kBytes for queries with m=2).

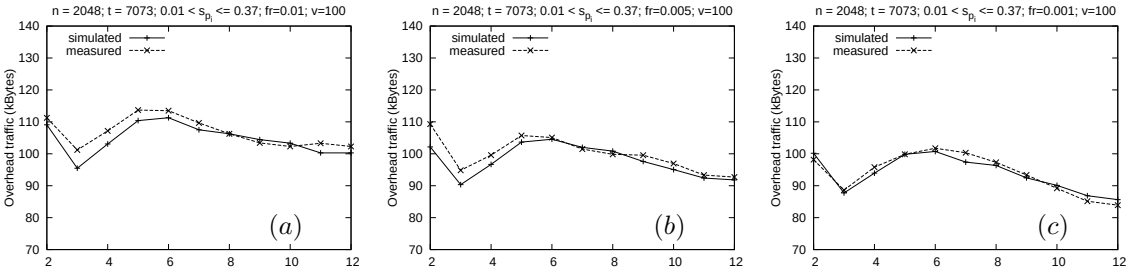

Figure 24: Measured vs simulated PST-based APS overhead traffic obtained with m ranging from 2 to 12, with v=100 and three values of fr: *a*) fr=0.01; *b*) fr=0.005; *c*) fr=0.001.

As a final remark, the results presented above show that the ARE and overhead traffic values measured in the real network are very close to those calculated by the simulator, which confirms the validity of the simulation results, in line with the theoretical findings discussed earlier in the paper.

8. Conclusions

We analytically compared the WPS and MSP search strategies to query XML data indexed in a DHT-based system. Based on this study, we defined the Adaptive Path Selection (APS) search strategy that adaptively resolves an XML query by querying either the most selective path or the whole path set. Experimental results confirmed that APS saves a significant amount of traffic compared to WPS and MSP.

Like MSP, APS needs to know the selectivity of the paths in the query. This required two issues to be addressed: *i*) finding a space-efficient data structure to store the selectivity of each path in the network; *ii*) defining an effective solution to build and propagate such data structure across the network. Concerning the first issue, we defined the PST data structure, allowing for accurate and efficient path selectivity estimation.

The second issue has been addressed by defining the PST_CP algorithm, which allows us to construct and propagate the PST with logarithmic performance bounds. Experimental results show that the PST accurately estimates the path selectivity values, and that the traffic generated by APS using PST-estimated selectivity values is comparable to that produced by APS assuming to know the real path selectivity values.

References

[1] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. ACM SIGCOMM'01, San Diego, USA, 149-160, 2001.

[2] A. Rowstron, P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. 18th IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany, 329-350, 2001.

[3] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, J. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. IEEE J. on Sel. Areas in Comm., 22(1):41-53, 2004.

[4] P. Maymounkov, D. Mazières: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. 1st Int. Workshop on Peer-to-Peer Systems (IPTPS 2002), Cambridge, USA, 53-65, 2002.

[5] L. Galanis, Y. Wang, S. R. Jeffery, D. J. DeWitt. Locating data sources in large distributed systems. 29th Int. Conf. on Very Large Data Bases (VLDB'03), Berlin, Germany, 874-885, 2003.

[6] G. Skobeltsyn, M. Hauswirth, K. Aberer. Efficient Processing of XPath Queries with Structured Overlay Networks. CoopIS/DOA/ODBASE Conf., Agia Napa, Cyprus, 2005.

[7] S. Abiteboul, I. Manolescu, N. Polyzotis, N. Preda, C. Sun. XML Processing in DHT Networks. 24th IEEE Int. Conf. Data Engineering (ICDE'08), Cancun, Mexico, 606-615, 2008.

[8] K. Karanasos, A. Katsifodimos, I. Manolescu, S. Zoupanos. ViP2P: Efficient XML Management in DHT Networks. 12th International Conference, ICWE 2012, Berlin, Germany, July 23-27, 2012.

[9] G. Koloniari, E. Pitoura. Peer-to-peer management of XML data: issues and research challenges. SIGMOD Record, 34(2), 2005.

[10] K. Aberer. Peer-to-peer data management. Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 3:87-94, 2011.

[11] L. Garces-Erice, P. A. Felber, E. W. Biersack, G. Urvoy-Keller, and K. W. Ross. Data Indexing in Peer-to-Peer DHT Networks. *Proceedings of the 24th IEEE International Conference Distributed Computing Systems (ICDCS'04)*, Hachioji, Tokyo, Japan, 200-208, 2004.

[12] I. Miliaraki, Z. Kaoudi, and M. Koubarakis. XML Data Dissemination Using Automata on Top of Structured Overlay Networks. 17th International conference on World Wide Web (WWW'08),865-874 2008.

[13] P. R. Rao, B. Moon. Locating XML Documents in a Peer-to-Peer Network Using Distributed Hash Tables. IEEE Trans. on Knowledge and Data Engineering, 21(12):1737-1752, 2009.

[14] A. Aboulnaga, A. R. Alameldeen, J. F. Naughton. Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. VLDB Journal, 591-600, 2001.

[15] L. Lim, M.Wang, S. Padmanabhan, J. S. Vitter, R. Parr. XPathLearner: An On-line Self-Tuning Markov Histogram for XML Path Selectivity Estimation. VLDB, 442-453, 2002.

[16] V. Slavov, P. Rao. A gossip-based approach for Internet-scale cardinality estimation of XPath queries over distributed semistructured data. The VLDB Journal,Volume 23 Issue 1, February 2014 pp. 51-76.

[17] Y. Wu, J. M. Patel, H. V. Jagadish. Estimating Answer Sizes for XML Queries. 8th International Conference on Extending Database Technology, Prague, 59017608, 2002.

[18] J. Freire, J. R. Harista, M. Ramanath, P. Roy, J. Simone. StatiX: Making XML Count. SIGMOD Conf., 2002.

[19] N. Polyzotis, M. Garofalakis. Statistical Synopses for Graph Structured XML Databases. SIGMOD Conf., 2002.

[20] N. Polyzotis, M. Garofalakis, Y. Ioannidis. Approximate XML Query Answers. SIGMOD Conf., 2004.

[21] N. Polyzotis, M. Garofalakis. XCluster Synopses for Structured XML Content. 22th Int. Conf. on Data Engineering, Atlanta, 2006.

[22] Riham Abdel Kader. Optimization in Relational Database Systems. In VLDB, 2007.

[23] Luo, Cheng and Jiang, Zhewei and Hou, Wen-Chi and Yu, Feng and Zhu, Qiang. A Sampling Approach for XML Query Selectivity Estimation. In EDBT, pp. 335–344, 2009.

[24] Alrammal, M and Hains, G. A stream-based selectivity estimation technique for forward XPath. In IIT, pp.209 - 214, 2012.

[25] M. Ramanath, L. Zhang, J. Freire, J. R. Haritsa. IMAX: Incremental Maintenance of Schema-Based XML Statistics. 21st Int. Conf. on Data Engineering, Tokyo, Japan, 27317284, 2005.

[26] 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, 61, Atlanta, GA, 2006.

[27] D. K. Fisher and S. Maneth. Structural Selectivity Estimation for XML Documents. In Proc. of the 23th IEEE Intl. Conference on Data Engineering, 62617635, Istanbul, Turkey, 2007.

[28] C. Luo, Z. Jiang, W.-C. Hou, F. Yu, Q. Zhu. A Sampling Approach for XML Query Selectivity Estimation. 12th Int. Conf. on Extending Database Technology, 33517344, Saint Petersburg, Russia, 2009.

[29] D. Kempe, A. Dobra, J. Gehrke. Gossip-Based Computation of Aggregate Information. 44th IEEE Symposium on Foundations of Computer Science, Cambridge, USA, 2003.

[30] C. Comito, D. Talia, P. Trunfio. Selectivity-based XML query processing in structured peer-to-peer networks. 14th Int. Database Engineering and Applications Symposium (IDEAS 2010), Montreal, Canada, 236-244, 2010.

[31] W. Wang, H. Jiang, H. Lu, J. Xu Yu. Bloom Histogram: Path Selectivity Estimation for XML Data with Updates. 30th Int. Conf. on Very Large Data Bases, Toronto, Canada, 240-251, 2004.

[32] S. El-Ansary, L. Alima, P. Brand, S. Haridi, Efficient Broadcast in Structured P2P Networks. 2nd Int. Workshop on Peer-to-Peer Systems (IPTPS'03), Berkeley, USA, 2003.

[33] A. Ghodsi. Multicast and Bulk Lookup in Structured Overlay Networks. In: X. Shen, H. Yu, J. Buford, M. Akon (Eds.) Handbook of Peer-to-Peer Networking, 933-958, Springer, 2010.

[34] D. Talia, P. Trunfio, Enabling dynamic querying over distributed hash tables. Journal of Parallel and Distributed Computing, 70(12):1254-1265, 2010.

[35] V. Poosala, Y. E. Ioannidis, P. J. Haas, E. J. Shekita: Improved Histograms for Selectivity Estimation of Range Predicates. SIGMOD Conf., 294-305, 1996.

[36] Cai, M., Frank, M., Chen, J., Szekely, P.: MAAN: A Multi-Attribute Addressable Network for Grid Information Services. Journal of Grid Computing 2(1): 3-14 (2004).

[37] B. Bloom. Space/Time Tradeoffs in Hash Coding with Allowable Errors. CACM 13(7), 422-426, 1970.

[38] H. V. Jagadish, V. Poosala, N. Koudas, K. Sevcik, S. Muthukrishnan, T. Suel. Optimal Histograms with Quality Guarantees. 24rd Int. Conf. on Very Large Databases, 275-286, 1998.

[39] Li Fan, Pei Cao, Jussara M. Almeida, Andrei Z. Broder: Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Trans. Netw. 8(3):281-293, 2000.

[40] G. Pirro, D. Talia, P. Trunfio. A DHT-Based Semantic Overlay Network for Service Discovery. Future Generation Computer Systems, vol. 28, n. 4, pp. 689–707, Elsevier Science, April 2012.

[41] K. Aberer, P. Cudr-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, R. Schmidt. P-Grid: a self-organizing structured P2P system. SIGMOD Record vol. 32, n. 3, pp. 29–33, 2003.