Distributed and Parallel Databases manuscript No.
(will be inserted by the editor)

Indexing Views to Route Queries in a PDMS*

Lefteris Sidirourgos[1], **George Kokkinidis**[1], **Theodore Dalamagas**[2], **Vassilis Christophides**[1], **Timos Sellis**[2]

[1] Institute of Computer Science - FORTH Heraklion, Greece,
e-mail: {`lsidir, kokkinid, christop`}`@ics.forth.gr`

[2] School of Electr. and Comp. Engineering, NTU Athens, Greece
e-mail: {`dalamag, timos`}`@dblab.ece.ntua.gr`

Received: date / Revised version: date

Abstract P2P computing gains increasing attention lately, since it provides the means for realizing computing systems that scale to very large numbers of participating peers, while ensuring high autonomy and fault-tolerance. Peer Data Management Systems (PDMS) have been proposed to support sophisticated facilities in exchanging, querying and integrating (semi-) structured data hosted by peers. In this paper, we are interested in routing graph queries in a very large PDMS, where peers advertise their local bases using fragments of community RDF/S schemas (i.e., views). We introduce an original encoding for these fragments, in order to efficiently check whether a peer view is subsumed by a query. We rely on this encoding to design an RDF/S view lookup service featuring a statefull and a stateless execution over a DHT-based P2P infrastructure. We finally evaluate experimentally our system to demonstrate its scalability for very large P2P networks and arbitrary RDF/S schema fragments, and to estimate the number of routing hops required by the two versions of our lookup service.

1 Introduction

Scientific or educational communities are striving nowadays for highly autonomous infrastructures enabling to exchange queries and integrate (semi-)structured data hosted by peers. In this context, we essentially need a *P2P data management system* (PDMS), capable of supporting loosely coupled communities of databases in which each peer base can join and leave the network at free will, while groups of peers can collaborate on the fly to provide advanced data management services on a very large scale (i.e., thousands of peers, massive data). A number of recent PDMSs [4, 11, 14, 20] recognize the importance of intensional information (i.e., descriptions about peer contents) for supporting such services. Capturing explicitly the semantics of databases available in a P2P network using a schema enables us to (a) support expressive queries on (semi-) structured data, (b) deploy effective methods for locating remote peers that can answer these queries and (c) build efficient distributed query processing mechanisms.

In this paper, we are interested in routing graph queries addressed to an RDF/S based PDMS. More precisely, we consider that peers advertise their local bases using fragments of community RDF/S schemas (e.g., for e-learning, e-science, etc.). These advertisements are specified by appropriate RDF/S views and they are employed during query routing to discover the partitioning (either horizontal, vertical or mixed) of data in remote peer bases. The main challenge in this setting, is to build an effective and efficient lookup service for identifying, in a decentralized fashion, which peer views can fully or partially contribute to the answer of a specific query. Our work is motivated by the fact that a sequential execution of the routing and planning phases for a specific query is not feasible solution in a PDMS context. As a matter of fact, due to the very large number of peers that can actually contribute to the answer of a query, an interleaved query routing and planning will enable us to obtain as fast as possible the first answers from the most relevant peers while the query is further processed by others. The results presented in this paper[1] is the first step towards this goal [25]. More precisely, we make the following contributions:

– we propose a novel encoding of arbitrary RDF/S schema graph fragments for checking whether a peer view is subsumed by a query;
– we introduce a DHT-based schema index to smoothly distribute view advertisements over peers;
– we design an RDF/S view lookup service that identifies which peers can fully or partially contribute to the answer of a graph query;
– we experimentally demonstrate the scalability of our DHT-based schema index for networks of different sizes, as well as, estimate the number of

[1] This work has been presented in the 4th Hellenic Data Management Symposium (HDMS'05), Aug 2005. The symposium does not publish official proceedings.

routing hops required by a statefull and a stateless execution of the proposed lookup service.

To the best of our knowledge no other PDMS offers the aforementioned functionality. Compared to the data indexes maintained by *data-driven PDMSs* that publish directly peer bases on the network [27,13,5,6,2], the distributed index on peer views maintained in our system is smaller in size. In fact, it corresponds to the number of fragments that can be extracted from the RDF/S schemas. Also, it requires a considerably smaller number of messages to be exchanged when peers join or leave the network. Moreover, since schema fragments advertised by peers evolve less frequently than their actual bases, such a schema index does not need frequent updates. As a result, in our approach index maintenance costs are reduced. Unlike other *schema-driven PDMSs* [20,12] which maintain a simple inverted list of the RDF/S classes (or properties) actually populated in peer bases, our framework is capable of routing in one step complex graph queries. The proposed lookup service is able to immediately identify peers matching an RDF/S schema graph fragment without the need to further decompose queries. In this way, we are able to return the first results from those peers that can actually answer the initial query as a whole. Further, our framework exploits the computing power of the P2P network for fairly distributing in different peers the routing, planning and execution load of queries. Finally, it is worth noticing that PDMSs like Piazza [14] rely on the mappings established between the individual peer schemas to route queries on semantically related peers rather than on a distributed index of graph fragments from multiple schemas. We consider that schema heterogeneity is an orthogonal issue, although our system can be extended to also address query reformulation issues [8] when complex mapping rules are available between the schemas employed by peers.

We believe that our framework is particularly suited for supporting large scale autonomous organizations for which neither a centralized warehouse nor an unlimited data migration from one peer to another are feasible solutions due to societal or technical restrictions. However, peers agree to publish and query their bases according to a number of globally known schemas (e.g., defined by various standardisation bodies). Moreover, our framework can be used as a base system, where one can build upon it sophisticate data management services, such as workload balance and data replication mechanisms. Although, we rely on RDF/S schemas and Chord for deploying a structured P2P infrastructure, the results presented in this work can be easily adjusted to other data models, like XML, and DHT protocols, like CAN [22].

The rest of the paper is organized as follows. In Section 2, we overview the proposed framework by focusing on how expressive RDF/S queries employed to retrieve data from the P2P network are matched against the views published by the peers to advertise their bases. In Section 3, we introduce our encoding of arbitrary RDF/S schema fragments. In Section 4, we detail how this encoding can be employed to build a DHT-based schema index supporting effective and efficient lookup of intensional peer base advertisements. In Section 5, we analyse the experimental figures of our framework. Finally, Section 6 discusses related work and Section 7 summarizes our contributions and future work.

2 RDF/S based PDMS

In our framework, we consider that every peer provides descriptions about information resources available in a P2P network that conform to a number of community schemas (e.g., for e-learning, e-services, etc.). Peers employing the same schema to construct such description belong essentially to the same *Semantic Overlay Network (SON)* [11]. The notion of SONs appears to be an intuitive way to cluster together peers sharing the same model for a particular domain or application for expressing useful queries and exchange information with others. Of course, a peer may belong to more than one SON, depending on the semantics of its base. Moreover, a peer may host only a part of the semi-structured descriptions available in the network.

The assumption about the existence of several globally known schemas in large scale networks seems more reasonable than forcing thousands of peers to design and integrate their disparate schemas. There is a natural pathway for adopting such globally schemas: the information produced by popular software and standardization bodies [15]. Since our framework is orthogonal to mappings, advances in the area of data integration can only benefit our work.

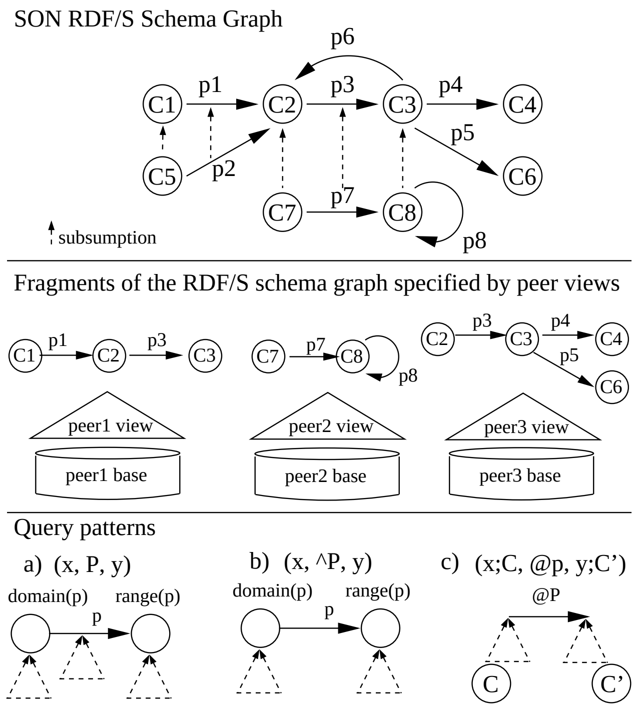

Fig. 1 An RDF/S schema graph, fragments advertised by peers and query patterns.

In order to design an effective and efficient P2P query routing mechanism we need to address the following issues: (a) how a SON can be defined? (b) how peers advertise their bases in a SON? (c) how peers formulate queries in a SON and finally (d) how peers decide which views of a SON match their queries? In the following subsections, we will present the main design choices of our framework in response to the above issues.

2.1 RDF/S schemas

A natural candidate for representing descriptive data (ranging from simple structured vocabularies to complex reference models [18]) about various information resources available in a SON is the Resource Description Framework and Schema Language (RDF/S) [23]. The core primitives of RDF/S schemas are classes and properties. Classes describe general concepts or entities. Properties describe characteristics of classes or relationships between classes. Both classes and properties may be related through subsumption. Every property defined in an RDF/S schema has a *domain class* (i.e., the class that has this property) and a *range class* (i.e., the value of this property). A property and its domain and range classes form a *schema triple*, denoted by $(domain(p),\ p,\ range(p))$. An RDF/S schema is a set of schema triples forming a directed labelled (multi)graph, called in the sequel *RDF/S schema graph*. For example, consider the RDF/S schema graph shown in the upper part of Figure 1. The circular nodes are labeled with class names (e.g., $C2,\ C3$), while the solid edges with property names (e.g., $p3$). The dashed edges represent the subsumption relationships of classes (e.g., between $C7$ and $C2$) or properties (e.g., between $p7$ and $p3$). Formally, an RDF/S schema graph is defined as follows.

Definition 1 *An RDF/S schema graph is a directed multigraph* $\mathcal{R} = (\{C \cup L\}, P, \prec^c, \prec^p)$*, where:*

1. *C is a set of nodes labelled with an RDF/S class name.*
2. *L is a set of nodes labelled with a data type (RDF/S literals).*
3. *P is a set of edges (c_1, p, c_2) from a node c_1 to a c_2 labelled with a property p, where $domain(p) = c_1$ with $c_1 \in C$ and $range(p) = c_2$ with $c_2 \in C \cup L$.*

4. *$\prec^c$ is a partial order imposed on nodes in C (RDF/S class subsumption).*
5. *$\prec^p$ is a partial order imposed on edges in P (RDF/S property subsumption).*

The framework presented in this paper can be applied to a wide-range of applications needs: from one SON defined by a unique RDF/S schema, to a SON defined by several interconnected RDF/S schemas until several SONs defined by different RDF/S schemas. This is due to the expressiveness of the RDF/S data model which (a) allows easy reuse or refinement of descriptive schemas employed by peers through subsumption of both classes and properties; (b) permits irregular heterogeneous descriptions in the sense that a resource may be multiply classified under several classes from one or several peer schemas (identified by appropriate namespaces) and (c) extends the scope of a resource description beyond the physical boundaries of an XML file hosted by a peer.

2.2 RDF/S peer base advertisements and queries

Each peer should be able to advertise the content of its local base to others with respect to the RDF/S schemas of the SONs they belong to. Using these advertisements a peer can become aware of the data hosted in remote peer bases. However, since an RDF/S schema may contain numerous classes and properties not necessarily populated in a peer base, we need a fine-grained definition of schema-based advertisements. To this end, we employ views to specify the *fragment* of an RDF/S schema graph for which all classes and properties are populated in a peer local base. In a similar way, peers can retrieve data from the PDMS by issuing queries, which also specify a particular RDF/S schema *fragment* of interest.

Queries in our framework are formulated in RQL [17], a full-fledged RDF query language which provides sophisticate pattern matching facilities against RDF/S schema and data graphs. Additionally, peers employ RVL [19], an extension of RQL, for defining views as virtual RDF/S graphs. Both languages employ patterns to extract the RDF/S schema graph fragments which are relevant to the data requested by a query/view. In the rest of the paper we stick on the notion of RDF/S schema fragments specified by these patterns, rather than their syntax on RQL or RVL.

Definition 2 *Given an RDF/S schema graph $\mathcal{R} = (\{C \cup L\}, P, \prec^c, \prec^p)$, a fragment specified by a query or view pattern over $\mathcal{R}$ is a subgraph $\mathcal{R}' = (C', P')$ such that $C' \subseteq C$ and $P' \subseteq P$.*

The lower part of Figure 1 illustrates three peer bases and their view advertisements. Each view specifies a different fragment of the SON RDF/S schema graph depicted in the upper part of Figure 1. The right part of Figure 1 illustrates some of the basic patterns that can be employed to formulate complex queries and their corresponding fragments of the SON RDF/S schema graph. More precisely, the pattern of Figure 1a can be used to retrieve all the instances (X, Y) of the domain and range classes of property p. Note that this pattern takes also into account the class and property subsumption relationships (denoted by the dashed triangles) to include in the result transitive instances of domain/range classes. For example, the query pattern $(X, p3, Y)$ will return the direct and transitive instances of $C2$ (i.e., domain class) and $C3$ (i.e., range class) related through the property $p3$ and its subsumed property $p7$. The pattern of Figure 1b is similar to the first, with the exception of considering only the strict interpretation of property p (i.e., no properties subsumed by p will be included in the result). Finally, the pattern of Figure 1c will return all the properties relating instances of the classes C and C', respectively. These properties can be either defined to have C and C' as domain and range classes respectively but also any of the classes subsuming them.

We can easily observe the similarity in the intensional representation of both peer base advertisements and query requests as RDF/S schema graph fragments. By representing in the same logical framework what data are requested by a SON (i.e., queries) and what data are actually hosted in each peer base of the SON (i.e., views), we can easily understand the data partitioning (horizontal, vertical, mixed) in remote peers relative to a query. This framework can be easily extended to reformulate queries expressed against a SON RDF/S schema in terms of the heterogeneous schemas or data models (e.g., relational, XML) employed locally by the peer bases [8].

2.3 RDF/S query and view subsumption

In order to decide which peer advertisements match a SON query, we need to check whether the classes and properties of the RDF/S schema fragments specified by the corresponding peer views are subsumed by those of the query. As studied in [24] checking containment of conjunctive RVL views and RQL queries involving arbitrary RDF/S schema and data patterns is an NP-complete problem. For this reason we restrict our framework to patterns returning data according to a known schema fragment[2] as follows:

Definition 3 *Let the RDF/S schema graph* $\mathcal{R} = (C, L, P, \prec^c, \prec^p)$*. Let also* $\mathcal{R}' = (C_1, P_1)$ *and* $\mathcal{R}'' = (C_2, P_2)$ *be two fragments of* $\mathcal{R}$*, specified by a query pattern* Q *and a view pattern* V*, respectively* ($C_1, C_2 \subseteq C$ *and* $P_1, P_2 \subseteq P$)*.* Q *subsumes* V *(or* V *is subsumed by* Q*) if:*

1. $\forall c_1 \in C_1,\ \exists c_2 \in C_2,\ c_1 = c_2$ *or* $c_2 \prec^c c_1$*, and*
2. $\forall p_1 \in P_1,\ \exists p_2 \in P_2,\ p_1 = p_2$ *or* $p_2 \prec^p p_1$*.*

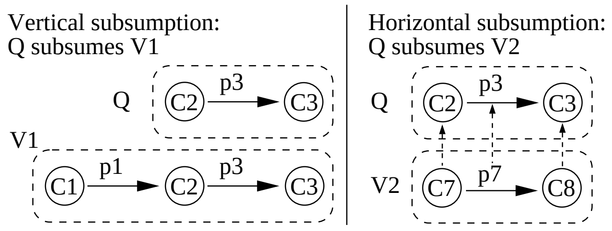

Fig. 2 Two cases of view subsumption.

Notice that in the above definition, all classes/properties in Q must be present or subsume a class/property in V. However, V may have additional classes and properties. Figure 2 illustrates two different subsumption cases. In the left part of Figure 2, query Q *vertically* subsumes view $V1$ in the sense that $V1$ has property $p1$ with domain class $C1$ that are not present in Q.

[2] As a matter of fact we consider an RQL core fragment including also range restricted schema variables as appearing in pattern c) of Figure 1 [24].

However, $V1$ is subsumed by Q since it contains a fragment that matches Q. On the right part of Figure 2, Q *horizontally* subsumes $V2$ since all classes and properties in $V2$ are subsumed by the classes and properties of Q. A query may subsume a view in either the above two ways or in any combination of them. Therefore, we need efficient support to decide subsumption of RDF/S schema graph fragments. For this purpose, we will present in the next section an encoding allowing to check whether an RDF/S schema fragment is subsumed by another, in linear time to the size of the fragments.

3 Encoding RDF/S schema fragments

This section introduces a succinct representation of RDF/S schema graphs, based on a structure called *Adjacency and Subsumption Cube* (in short *AdjSub Cube*) allowing to derive an encoding for fragments of arbitrary size and structural form (i.e., linear, tree or graph). The *AdjSub Cube* is a conceptual structure that helps us derive an encoding, but it does *not* need to be implemented by any peer. An *AdjSub Cube* provides (a) adjacency information for nodes (i.e., whether a class is related to another one via a certain property) and (b) subsumption information for classes and properties (i.e., whether a class/property subsumes another).

The *AdjSub Cube* extends the concept of the adjacency matrix by adding a third dimension capturing labeled edges (i.e., the properties). For each property, the first dimension (vertical) represents the nodes that appear as

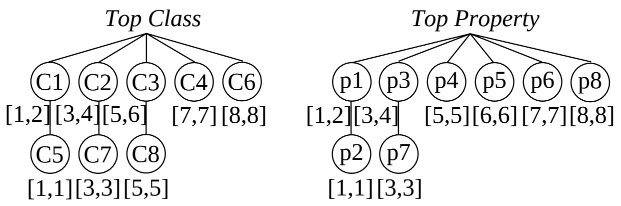

Fig. 3 Interval encoding of the class/property hierarchies.

the domain classes, the second (horizontal) the nodes that appear as the range classes and the third one the property name. Moreover, it imposes an ordering of classes and properties on each dimension based on an interval encoding of the RDF/S class and property subsumption hierarchies.

In general, an interval encoding over a subsumption hierarchy is maintained using labels of the form [*start, end*] such that every interval of a child node is contained in the interval of its parent. In this paper we employ the encoding of [3] where a tree node[3] u is labeled with $[\mathit{index}(u), \mathit{post}(u)]$: $\mathit{post}(u)$ is the number assigned to u when a postorder tree traversal is considered, while $\mathit{index}(u)$ is the lowest of the *post* numbers assigned to u's descendants. Note that $\mathit{index}(u) \leq \mathit{post}(u)$ and that $u \prec^c v$ (or $u \prec^p v$) iff $\mathit{index}(u) \geq \mathit{index}(v) \wedge \mathit{post}(u) < \mathit{post}(v)$. Figure 3 illustrates the interval encoding of the class and property subsumption hierarchies given in the SON RDF/S schema of Figure 1. An *AdjSub Cube* exploits the *post* numbers of the classes and properties to arrange them in each dimension. It follows that when we organize classes (properties) in the inverse order from the one

[3] For simplicity, we consider only tree shaped subsumption hierarchies although this work can be extended to DAGs [9] by applying a left/right DFS labeling schema.

obtained by the postorder traversal, subsumed classes (or properties) will succeed the subsuming classes (or properties).

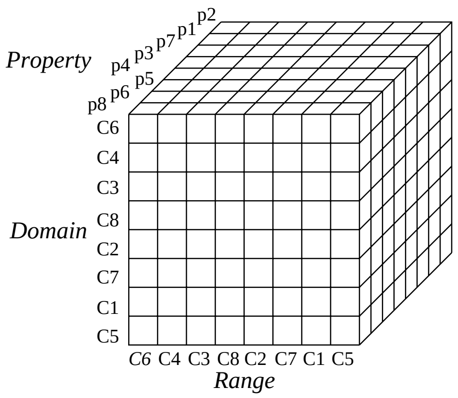

Fig. 4 The *AdjSub Cube* of an RDF/S schema.

Given an RDF/S schema graph $\mathcal{R}$, we define the corresponding *AdjSub Cube AS* as follows: for every *schema triple* (d, p, r) of $\mathcal{R}$, where p is the property, d the domain class and r the range class of the property, we set cell $AS_{ijk} = 1$, where $i = |C| - post(d)$, $j = |C| - post(r)$ and $k = |P| - post(p)$. The remaining cells in the *AdjSub Cube* are set to 0. An example of the *AdjSub Cube* constructed for the RDF/S schema of Figure 1, is depicted in Figure 4. We next define formally the *AdjSub Cube*.

Definition 4 *Let an RDF/S schema graph $\mathcal{R} = (C, L, P, \prec^c, \prec^p)$, and $d, r \in C$, $p \in P$. An AdjSub Cube AS for $\mathcal{R}$ is an $|C| \times |C| \times |P|$ bitmap such that:*

$$AS(i,j,k) = \begin{cases} 1 \text{ } \textit{if domain}(p) = d \textit{ and range}(p) = r \\ 0 \text{ } \textit{otherwise} \end{cases}$$

where $i = |C| - post(d)$, $j = |C| - post(r)$ and $k = |P| - post(p)$.

Any fragment of the original RDF/S schema graph, can be represented in the same *AdjSub Cube* created for this schema, by setting to 1 only those cells of the cube that correspond to the schema triples of the fragment. We can enumerate all cells in the *AdjSub Cube* based on their position through the function $pos(AS_{ijk}) = k \times |C|^2 + i \times |C| + j$, where $i, j = 0, 1, \ldots, |C| - 1$ and $k = 0, 1, \ldots, |P| - 1$. Based on this enumeration, every fragment is encoded by assigning a unique number N as defined below:

Definition 5 *Let an RDF/S schema graph $\mathcal{R} = (C, L, P, \prec^c, \prec^p)$ and the corresponding AdjSub Cube AS. Every fragment of $\mathcal{R}$ represented in AS, is encoded by assigning a unique number N, such that*

$$N = a_{L-1}2^{L-1} + \ldots + a_1 2^1 + a_0 2^0 \text{ and } a_n = AS_{ijk},$$

where $n = pos(AS_{ijk})$, and $L = |C| \times |C| \times |P|$ the size of the AdjSub Cube AS.

The *unique number* N is a binary number, where the coefficient a_n of the factor 2^n is set to 0 or 1 depending on the value of the cell whose position is $pos(AS_{ijk}) = n$. One may easily compute N if for every *schema triple* (d, p, r) of the fragment locates its position in the *AdjSub Cube*. Since N may be a fairly large number, a simple way to store N is as a set of integers $\{n_1, n_2, n_3, \ldots\}$ where n_i is the $pos(AS_{ijk})$ of each *schema triple* in the fragment. For example, consider the view $V1$ of Figure 2 comprising the *schema triples* $t_1 = (C1, p1, C2)$ and $t_2 = (C2, p3, C3)$. For the first *schema triple* t_1 we have: $i = |C| - post(C1) = 8 - 2 = 6$, $j = |C| - post(C2) =$

$8-4=4$, $k=|P|-post(p1)=8-2=6$ and $pos(t_1)=k\cdot 8^2+i\cdot 8+j=6\cdot 64+6\cdot 8+4=436$. Consequently, for the *schema triple* t_2, $pos(t_2)=k\cdot 8^2+i\cdot 8+j=4\cdot 64+4\cdot 8+2=290$. Thus, the unique number of view $V1$ is $N(V1)=2^{436}+2^{290}\equiv\{436,290\}$. Given the above encoding, we can decide if a fragment is subsumed by another in linear time to its size.

Theorem 1 *Given two connected fragments $\mathcal{R}',\mathcal{R}''$ of an RDF/S schema graph $\mathcal{R}$ and their unique numbers defined by the above encoding $N(\mathcal{R}')=2^{n_1}+2^{n_2}+\ldots+2^{n_k}\equiv\{n_1,n_2,\cdots,n_k\}$ and $N(\mathcal{R}'')=2^{n_1'}+2^{n_2'}+\ldots+2^{n_l'}\equiv\{n_1',n_2',\cdots,n_l'\}$, respectively, it holds that:*

(a) if $\mathcal{R}'$ subsumes $\mathcal{R}''\Rightarrow N(\mathcal{R}'')>N(\mathcal{R}')$.

(b) if $\forall n_i\in\{n_1,n_2,\ldots\},\exists n_i''=n_j',n_j'\in\{n_1',n_2',\ldots\}\setminus\{n_1'',\ldots,n_{i-1}''\}:n_j'\in\bigcup_\delta\bigcup_\lambda[n_i+\lambda|C|+\delta|C|^2,n_i+\lambda|C|+\delta|C|^2+\kappa]\Rightarrow\mathcal{R}'$ subsumes $\mathcal{R}''$, where $\delta=post(prop(t))-index(prop(t))$, $\lambda=post(domain(t))-index(domain(t))$, $\kappa=post(range(t))-index(range(t))$ and t is the schema triple corresponding to the cell with pos(AS)=n_i.

Sketch of proof. If $\mathcal{R}'$ subsumes $\mathcal{R}''$ then from the construction of the *AdjSub Cube* is easy to prove that $N(\mathcal{R}'')>N(\mathcal{R}')$. Given two fragments $\mathcal{R}',\mathcal{R}''$, in order for $\mathcal{R}'$ to subsume $\mathcal{R}''$, we need to check whether for every schema triple in $\mathcal{R}'$ ($\forall n_i$), there exists a subsumed schema triple in $\mathcal{R}''$ ($\exists n_j'$). Given that the schema triple corresponding to the cell n_j' is subsumed by the schema triple corresponding to the cell n_i, n_j' is positioned somewhere in a sub-cube of the *AdjSub Cube*. This sub-cube is confined by the subsumed triples of n_i.□

4 DHT-Framework for RDF/S

Structured P2P systems based on Distributed Hash Tables (DHTs) can support large, highly distributed networks while ensuring a fair load distribution among peers at the cost of an extra message overhead when peers join or leave the network. A popular DHT-based protocol for storing and retrieving pairs of *(key, data)* is Chord [26]. More precisely, the main service supported by Chord is *lookup(key)*, which returns in at most $O(\log n)$ routing hops (i.e., network messages) the peer's address that is responsible for storing the pair *(key, data)*. Peers are associated with keys through their identifiers. A peer's identifier is chosen by hashing the peer's IP address, while a *key identifier* is produced by hashing the key. Identifiers are ordered on an identifier circle *modulo* 2^m, called the *Chord ring*. Key k is assigned to the first peer whose identifier is equal to or follows the identifier of k in the identifier circle. This peer is called the *successor peer* of key k, denoted by *successor(k)*. To locate a key in the Chord ring, each peer maintains a routing table, called the *finger table*, where the i^{th} entry contains the identifier of the first peer that succeeds its identifier by at least 2^{i-1}.

In our context, we build a *DHT-based schema index* where a key is the unique number of the RDF/S fragment specified by the view of a peer. Rather than data objects a key designates peer IPs populating the corresponding fragment. To produce the *key identifier* and place it on the *Chord ring*, an order preserving hash function is used to maintain the ordering over subsumed views given by the *AdjSub Cube*. For example, if $N = \{n_1, n_2, n_3\}$ a simple order preserving hash function is $H(N) = (n_1 \cdot n_2 \cdot n_3) \mod 2^m$.

To summarize, each peer stores pairs of *(view, {peers})* and replies to a *lookup(view)* request with the set of peers that had advertised the specific view. However, there might be views hashing to the same key identifier or same keys may correspond to views that are defined over different SONs. The first problem can be easily bypassed by sending along with the lookup request, the unique number of the encoded view. The second problem can be addressed by distinguishing the lookup requests via the unique namespace of the RDF/S schema defining a SON. When SONs are interconnected, the *AdjSub Cube* is defined using the class and property hierarchies of all involved RDF/S schemas. Next, we describe how our DHT-based infrastructure evolves when peers join or leave the network or even update their views.

4.1 Peer joins, departures and updates

Each peer joining the network advertises through a view the fragment of the RDF/S schema which is actually populated in its local base. Recall that a peer is able not only to answer queries that match exactly its view, but also any of its fragments (i.e., views that vertically subsume its view). When the joining peer wishes to inform about its capability to answer queries

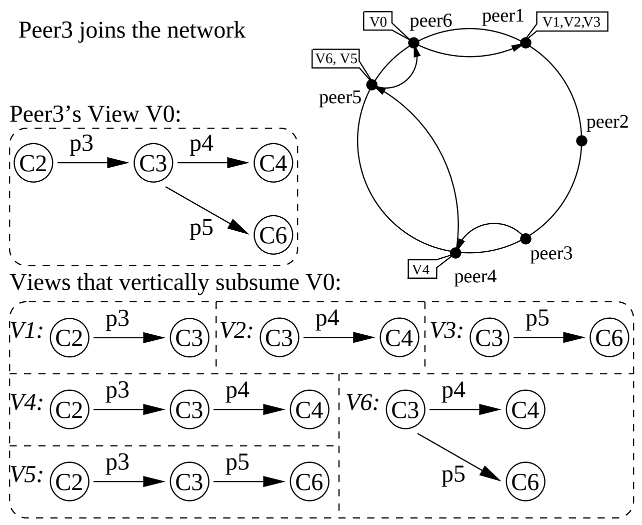

Fig. 5 Peer 3 advertise its view.

related to a vertically subsuming view, it issues *lookup(view)* requests to identify the successors responsible for the subsuming views accompanied by a *store(view, IP)* request. Moreover, Chord provides a mechanism for key reallocation for inserting the newly added peer to the DHT index. The predecessor of this new peer sends the *(view, {peers})* pairs for which it will be responsible from now on.

It should be stressed that a joining peer may advertise only the vertically and not the horizontally subsuming views. Advertising the horizontally subsuming views too, implies that queries are systematically extended to include peer data classified under subsumed classes and properties. However, this functionality is specified by the peer queries and not the peer views (e.g., see patterns (a) and (b) of Figure 1). On the other hand, if vertically subsuming views were not explicitly advertised but considered during query processing, the lookup service would have to search a larger portion of the

P2P network and thus incurs an increased cost in routing hops (i.e., to discover all keys that are a multiplier of the key associated with the requested view). The cost of advertising the vertically subsuming views is significant less than trying to explicitly locate them each time a new lookup request is issued. However, advertising all possible vertical subsuming views at once can be stressfull for the newly joining peer, since the number of distinct vertical views can crow in the worst case exponential to the number of properties in the view. For this reason, when a peer joins the network it may advertise only the vertical subsuming views with one triple (fragments of size 1). In this way the peer makes available all of its contents without having the overhead of advertising all of its vertical subsuming view. In a later stage, a peer may decide, depending the query workload and the duration of staying in the network, to advertise the rest of its vertical subsuming views. For example, in Figure 5, peer3 joins the network and publishes the view $V0$ and all views of size one that vertically subsume $V0$, namely $V1$ to $V3$. In this way peer3 has made available all of it contents. In a later stage, peer3 will also advertise views $V4$ to $V6$. Since peer5 is the successor peer of key identifiers $H(N(V5))$ and $H(N(V6))$, it receives a request to advertise peer3's views $V5$ and $V6$ and adds peer3 to the set of peers associated with views $V5$ and $V6$.

To ensure consistency of the DHT-index when a peer leaves the system, it must pass the key identifiers that holds to its successor peer[4]. In addition, it must notify all peers that have indexed its views to modify their stores.

There two ways to deal with an update of a peer view. The simplest way is to consider an update as a departure followed by a join. Another solution for trivial updates (such as adding or removing a schema triple from a view) is to let the peer to decide what changes must be performed to the DHT-based schema index to reflect the update effect. Consider for example that peer3's view $V0$ has been updated by adding the (C_1, p_1, C_2) schema triple. Then, the DHT-based schema index can be updated incrementally by advertising only the additional views subsuming the newly created one, since $V0, V1, \ldots, V6$ are still valid. Finally, a schema update is handled as if it was a creation of a new SON. Peers that are aware of the updates may republish their views in the updated SON by leaving and rejoining the network according to the new schema. This implies that for a period of time there will be two SONs (i.e., for the old schema and the updated) and peers should gradually pass from one to another.

We can categorize data, peer's view and schema updates according to their frequency and their cost in terms of the number of messages exchanged during these updates. Data updates are the most frequent updates that occur. However, in defense to our design they have no impact on the DHT-

[4] This operation is supported by the Chord protocol. Moreover, Chord provides services like *stabilize()* and *fix_fingers()* to support peers departure without any prior notification.

based schema index and thus no messages are exchanged. Peer view updates are the next most frequent updates. When view updates occur, a small number of messages must be exchange in order to reflect the changes into the schema index. Finally, the most expensive updates in terms of messages, are the schema updates but such updates are very rare compared to data and peer's view updates.

4.2 Lookup Service

In this section we present the *lookup service* which identifies all peers whose views are horizontally (or vertically) subsumed by a query. The first step is to lookup the view that specifies exactly the same fragment of the RDF/S schema graph as the one requested by the input query. This view is called the *strict view* and the peer that stores the *(strict view, {peers})* pair, the *initial peer*. The initial peer is identified through the *lookup(strict view)* service of the original Chord protocol. The next step is to locate all other views that are horizontally subsumed by the strict view through the invocation of a *sublookup()* service issuing a sequence of lookup requests. The sequence in which these lookup requests are performed is very important since there must be no lookups that address preceding peers. The encoding and the order preserving hash function guarantees that the key identifiers of all the subsumed views will be greater than than the key identifier of the strict view. As a result, all peers storing pairs of horizontally subsumed views will succeed the initial peer in the Chord ring. Thus we avoid to travel all over

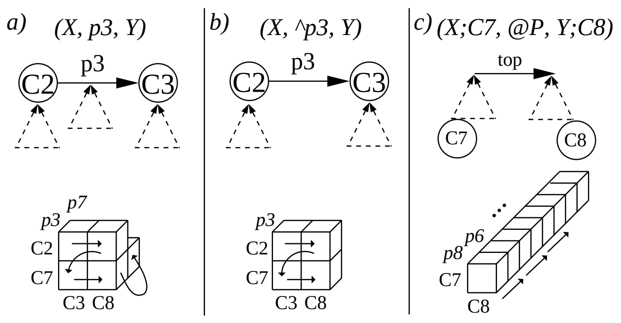

Fig. 6 AdjSub Cube traversal for RDF/S schema fragments.

the Chord ring to reach the destination peer. It should be stressed that the answer to a specific lookup request contains the peers whose views not only match strictly but also are vertically subsumed by the query.

The main intuition for the *sublookup()* algorithm is that the sequence in which the views are looked up is given by the *AdjSub Cube* (introduced in Section 3). Different regions of the *AdjSub Cube* define different sequences of lookups that must be issued to discover the views which are horizontally subsumed by an input query. The lower part of Figure 6 illustrates the three regions of the *AdjSub Cube* corresponding to the RDF/S schema graph fragments of the three patterns, depicted in the upper part of the figure. The fragments represented by solid edges and nodes in the middle part of the figure, are essentially the views matching strictly the three query patterns. In the first example (Figure 6a), the strict view to look up corresponds to the RDF/S schema fragment, which comprises the default domain (class $C2$) and range (class $C3$) of property $p3$. The second example (Figure 6b) has the same strict view. The third example (Figure 6c) illustrates a query pattern that requests all views that may have the class $C7$ as domain and the class $C8$ as range. In this case, the strict view is the one that has the top property (not visible in the *AdjSub Cube*), since we need to check all the properties defined in the corresponding RDF/S schema of the SON. For the three patterns of Figure 6, starting from the cell that corresponds to the schema triple of the strict view, the arrows designate the sequence in which the next subsumed view is chosen from the *AdjSub Cube*. In order to always choose the view that has the immediate larger key identifier, we traverse the *AdjSub Cube* by first moving to the right (substituting the range class of the triple), then down (substituting the domain class) and finally inwards (substituting the property). In more complex patterns, for each substitution of either the domain, range or property of each schema triple we substitute recursively the domain, range and property of all of its remaining schema triples. We next describe two versions of the *sublookup()* algorithm suitable for a stateless and a statefull execution in a DHT-based network.

Figure 7 gives the pseudocode for the stateless version of the *sublookup()* algorithm. When a query is issued, the initial peer is identified through a *lookup(strict view)* request. Next, *sublookup(strict view, strict view, initial peer)* is invoked. In order to guarantee that no view is looked up twice, a *substit()* function checks if the given domain or range class has already been substituted. For every substitution done to either the domain, range or property of a schema triple, a new view V is created and looked up. The peer that stores the new view V creates a new instance of the *sublookup()* function and continues its execution until all schema triples of the view are

```
**sublookup**(strict view $V_{strict}$, current view $V$, peer $pr$)

1: for all triples $t$ in $V$ that are not marked and
   the corresponding triple $t'$ in $V_{strict}$
2:     $r' \leftarrow low_{\prec^c}(t'.range, t.prop.default\_range)$
3:     $d' \leftarrow low_{\prec^c}(t'.domain, t.prop.default\_domain)$
4:     $r \leftarrow$ next class of $t.range$ on the $AdjSub\ Cube$
5:     $d \leftarrow$ next class of $t.domain$ on the $AdjSub\ Cube$
6:     $p \leftarrow$ next property of $t.prop$ on the $AdjSub\ Cube$
7:     if substit($t.range$) && $(r \prec^c r')$
8:         $t.range \leftarrow r$
9:     else if substit($t.domain$) && $(d \prec^c d')$
10:        if substit($t.range$)
11:            $t.range \leftarrow r'$
           end if
12:        $t.domain \leftarrow d$
13:    else if $(p \prec^p t'.prop)$
14:        if ($t.range \preceq^c p.default\_range$ || substit($t.range$))
           && ($t.domain \preceq^c p.default\_domain$
               || substit($t.domain$))
15:            $r'' \leftarrow min_{\prec^c}(t'.range, p.default\_range)$
16:            $d'' \leftarrow min_{\prec^c}(t'.domain, p.default\_domain)$
17:            if substit($t.range$)
18:                $t.range \leftarrow r''$
               end if
19:            if substit($t.domain$)
20:                $t.domain \leftarrow d''$
               end if
21:            $t.prop \leftarrow p$
22:        else
23:            mark $t$ and continue
           end if
24:    else
25:        mark $t$ and continue
       end if
26:    $pr \leftarrow lookup(V)$
27:    $sublookup(V_{strict}, V, pr)$
28:    mark $t$
   end for
```

Fig. 7 Stateless sublookup algorithm.

marked. The statefull *sublookup()* algorithm is a simplified version of the stateless one. The initial peer instead of passing the execution of *sublookup()* to the succeeding peers, it computes all the horizontally subsumed views and issues series of lookup requests for each view. As we can see in Figure 8, an

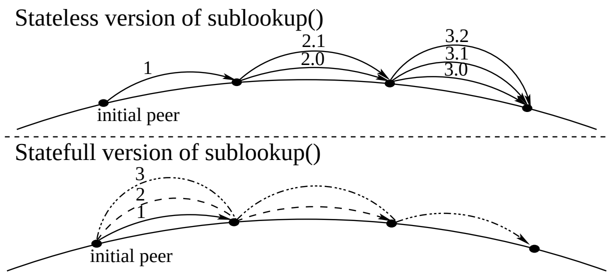
Stateless version of sublookup()
1
2.1
2.0
3.2
3.1
3.0
initial peer
Statefull version of sublookup()
3
2
1
initial peer

Fig. 8 Example of the execution on Chord.

advantage of the statefull version is that the whole execution is monitored by the initial peer. Thus, in the case where a succeeding peer stores more than one subsumed views, the statefull version will retrieve all views at once, contrary to the stateless one contacting the same peer as many times as the number of subsumed views it stores. However, in the statefull version each new lookup request requires in principle more routing hops, since the next subsumed view will succeed the previous one in the Chord ring. It is worth noticing that the stateless lookup service scans the Chord ring in a highly parallel way in order to process a query. We believe that for queries involving a large number of peers the most important factor is the degree of parallelization of the lookup requests rather than the absolute number of routing hops. Moreover, such a parallelization reveals in a natural way the design choices that must be taken when an *interleaved* execution of the routing and planning phases is considered.

To conclude, in a network of N peers, each lookup requires $O(\log N)$ routing hops. Therefore, the total number of routing hops required to locate all peer bases that can contribute to the evaluation of a query is:

$O(\log N)$ to locate the initial peer plus $S \times O(\log N)$ to locate the S subsumed views, where S depends on the size of the involved subsumption hierarchy. However, especially for views with a small number of schema triples, the subsumed views are located in peers that are close to each other in the *Chord ring*, thus as we will see in the sequel much less than $O(\log N)$ routing hops are required in practice for each lookup.

5 Experimental Evaluation

The goal of the experimental results presented in this section is to demonstrate the scalability of the DHT-based schema index with respect to the distribution of the keys for the encoded peer views, as well as to estimate the number of routing hops required to locate peer views that are subsumed by an input query. We conducted our experiments for different sizes of peer networks and views with varying number of schema triples and structural form (linear, tree or graph form). The RDF/S schemas that were used in our experiments were created synthetically based on real application examples [18]. Our experiments rely on the original Chord protocol simulator [7], modified to support the distributed index of RDF/S fragments, as well as to implement the two versions of the lookup service presented in Section 4.2. The stabilization algorithm of the Chord protocol and the key reallocation algorithms when peers join, leave or die unexpectedly, were kept intact. Thus, we considered that the system was formed and stabilized before any view was stored or looked up.

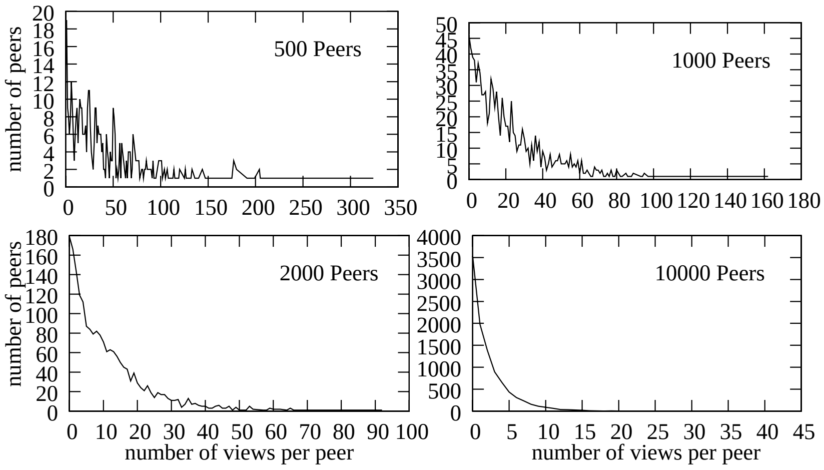

Fig. 9 Distribution of views over peers in networks of different size.

Schema	#Classes	#Prop.	#Fragm.
sch.1	27	30	6300
sch.2	25	23	6000
sch.3	22	18	5200
sch.4	19	15	5000

Table 1 Structural characteristics of the synthetic RDF/S schema graphs.

An important characteristic of the proposed encoding function (based on the *AdjSub Cube*) is related to the distribution of the views (keys) over the DHT nodes. For this purpose we stored fragments of 4 different RDF/S schema graphs on networks of varying size. Each RDF/S schema graph had more than 5000 distinct fragments[5]. Table 1 depicts the number of classes and properties of each RDF/S schema graph, as well as the number of distinct fragments that were extracted from each schema. The total number[6] of views stored in each network was 22500. The size of the network varied

[5] Since our query patterns capture only the structure and the semantics of an RDF/S schema (and not arbitrary joins on property values) the views considered here correspond to distinct fragments of a specific RDF/S schema.

[6] Given that the discriminating factor in our experiments is the ratio of views per peers, we keep constant the number of views stored in each network.

from 500 to 20000 peers. Table 2 presents the average number of views that should be stored per peer, the weighted average of views that are stored per peer, as well as the standard deviation and the maximum number of views per peer.

#Peers	**Avg.**	**W.Avg.**	**Std.D.**	**Max**
500	45.16	45.176	45.21	324
1000	22.5	22.24	21.62	162
2000	11.25	11.23	11.87	92
5000	4.5	4.51	5.27	61
10000	2.25	2.17	2.84	41
20000	1.12	1.11	1.72	21

Table 2 Distribution of views for networks of varying size ($m = 32$).

By examining the measurements of Table 2 one can observe that in all cases the weighted average is very close to the theoretical average. In addition, for large networks with more than 2000 peers, the standard deviation is acceptable. However, for networks of small size (i.e., 500 and 1000 peers) although the weighted average is still close to the theoretical average, the standard deviation is high. In small networks there are few peers that store no views and also few peers that store a high number of views. The skewed view distribution in these few peers (reflected in the standard deviation) is due to the fact that the unique identifiers of the views comprising only one schema triple, have small hash values thus they are stored exclusively on the peers placed at the beginning of the Chord ring. Moreover, peers that are placed at the end of the Chord ring come with no views. Figure 9 depicts the exact distribution of views over peers for networks of size 500, 1000, 2000 and 10000 peers. From the above figure it is clear that in each

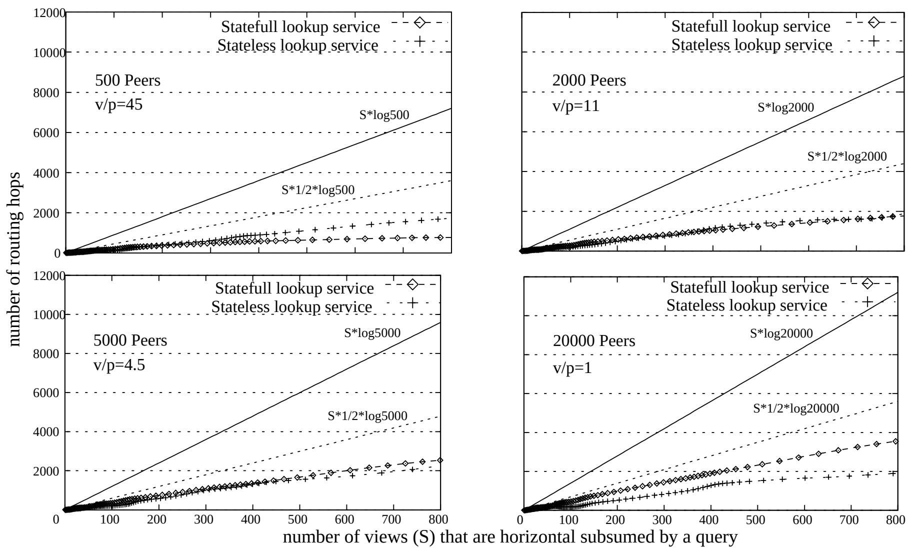

Fig. 10 Number of routing hops required by lookup service in networks of different size.

case less than 5 peers store an increased number of views. To overcame this problem one can force the peers that are placed at the end of the Chord ring to hash their identifier in the beginning of the Chord ring, and thus make more peers responsible for views with few schema triples. Alternatively, a simpler solution for small networks will be to consider a smaller identifier circle (i.e., by decreasing m of *modulo* 2^m) and thus place peers closer to each other. In the experiments presented so far we have set m to be 32. Table 3 show the experiments conducted with $m = 12$ for small networks, where the standard deviation has dropped to less than 10.

#Peers	**Avg.**	**W.Avg.**	**Std.D.**	**Max**
500	45.16	44.94	9.86	89
1000	22.5	22.41	8.27	63

Table 3 Distribution of views ($m = 12$).

The four graphs of Figure 10 illustrate the total number of routing hops per number of subsumed views involved in the evaluation of a query pattern, for both the statefull and stateless versions of the lookup algorithm. In addition, in each graph we illustrate both the theoretical ($S \times \log n$) and experimental ($S \times \frac{1}{2} \times \log n$) number of routing hops required by the original Chord [7] protocol. Clearly, the two versions of the lookup service decrease the number of routing hops up to 50% than those required by Chord. For networks of small size, the statefull lookup service outperforms the stateless one since it requires less than half routing hops for large views.

This can be easily justified by the fact that the stateless algorithm fails to identify the peers that can answer more than one subsumed view at once, and therefore it contacts the same peers over and over. This is not the case of the statefull lookup since the *initial peer* gathers all views from a peer and never contacts the same peer twice. However, when the network grows, the statefull version exhibits poor performance. As a matter of fact, as the network grows it becomes more unlikely to find peers storing more than one view, hence the advantages of the statefull approach fade out. On the other hand, the number of hops required by the stateless version only slightly increases when the size of the network doubles and clearly outperforms the statefull one when the ratio views per peer falls bellow 1.

The main conclusions drawn from our experiments are: (a) as the network grows, the DHT-based schema index succeeds to distribute the encoded views over all peers in the network and (b) the proposed lookup algorithms outperform the lookup service of the original Chord. Moreover, the stateless version of the lookup algorithm is more suitable for large networks as it scales gracefully, compared to the statefull one that is more beneficial for small networks. One can then decide, depending of the size of the network which lookup version to use.

6 Related Work

Closely related to our work are DHT-based PDMS addressing routing issues for queries over RDF, XML and relational databases. RDFPeers [6] is a distributed RDF repository based on an extension of Chord, namely MAAN, that stores each triple at three places in the network by applying a globally known hash function to its subject, predicate and object (i.e., *data triple*). A DHT index is built over RDF triples which ignores the semantics of the RDF/S schema during query routing. Such an extensional index extremely increases the total amount of data stored on the network and comes with a significant message overhead when triples are likely to frequently change. Finally, to overcome the increased workload of peers storing triples with subject popular URIs, RDFPeers consider a threshold after which it refuses to store any triples. In contrast, our framework favours the scalability, thus there will be no increased workload in a single peer and no data will be lost.

In [13], a distributed catalog for XML data is proposed along with appropriate load balancing techniques to fairly distribute the catalog service and adapt the system's behaviour to the query workload. In the DHT-based catalog keys are XML fragments associated with a set of structural summaries (i.e., the XPaths leading to these fragments). A B+-tree is used by each peer to match a given XPath query against the stored summaries. The system can then locate which peers can answer an entire XPath query using the leaf element as the key. In the case of an XPath query of the form $p = /a_1[b_1]/\ldots/a_n[b_n]\,op\ value$ where each b_i is in turn a path, the system must first extract all k linear paths and invoke the lookup service for each of them. In contrast to this system we do not distinguish between linear, tree or graph queries and thus in either cases the lookup service is invoked only once and $O(\log N)$ hops are required (without considering subsumed view) to locate peers that can answer the query. Also, there is no need for a search over a secondary index structure since our index is build directly on the RDF/S schema fragments. For load balancing, the authors propose techniques of splitting and replicating the catalog, while in our framework these are done a priory.

A unifying framework for relational query processing over structured P2P networks has been proposed in [27]. Each tuple of a relation $R(DA_1, \ldots, DA_k)$ is stored $k+1$ times over the network: one copy of the tuple with consistent hashing over it's primary key, and k replicas distributed in the peers according to an order-preserving hash function based on its k attributes. The authors also introduce the notion of *Range Guards*, i.e., a number of peers which keep additional replicas of all tuples whose values for a specific attribute fall in a specific range. They are used to evaluate range queries over relational data by avoiding costly traversals of the entire Chord ring. The two major drawbacks of this system are (a) the need to replicate data several times and thus increasing the maintaining cost, and (b) multi-relation/attribute or range queries are costly to route (in some cases needing $O(N)$ hops). In our framework, range queries over schema triples are efficiently evaluated since the proposed RDF/S fragment encoding ensures that they will be indexed closed to each other on the Chord ring, without the need of replication and range guards. Finally, in [21] a family of order preserving hash functions were presented, along with a tunable data replication mechanism which allows trading off replication costs for fair load distribution. Since our framework is independent from the order preserving hash function, such mechanisms can be implemented on top of our framework to support (a) workload balance during query evaluation and (b) data replication facilities. Notice that these data management services are independent of the advertisement and routing algorithms presented in this paper.

7 Conclusion

In this paper, we presented a DHT-based framework to efficiently route expressive RDF/S queries. We introduced a succinct representation of RDF/S schema graphs, called *AdjSub Cube*, for encoding arbitrary RDF/S schema fragments. This encoding ensures a fast view/query subsumption checking in order to understand the partitioning of data in remote peer bases. Based on this encoding, we designed a DHT-based schema index to distribute view advertisements over peers. Additionally, we implemented a lookup service for identifying which peers can completely or partially contribute to the answer of a graph query.

We experimentally demonstrated that the proposed DHT-based schema index scales gracefully for very large number of peers. Moreover, we compared the routing hops required by a distributed and a centralized version of our lookup service versus the routing hops required by the original Chord protocol. The main conclusion drawn from our experiments is that our lookup service requires less than half of the routing hops required by the original Chord protocol. The results presented in this paper can be easily adjusted to other DHT-based protocols and schema formalisms defining SONs. For example, we can build an *AdjSub Cube* for encoding fragments of an XML schema tree.

We intend to extend our work with query planning capabilities of the peer views returned by the lookup service. In particular, we are interested to an *interleaved* execution of the routing and planning algorithms in several rounds, allowing to obtain as fast as possible the first results of a query available in peer bases. This interleaved execution not only favors intra-peer processing, which is less expensive than the inter-peer one, but additionally exhibit the benefit of a parallel execution of the query routing, planning and evaluation in different peers [25]. Finally, we intend to investigate the potential of a P2P infrastructure based on distributed trees [10, 1, 16], for implementing the *AdjSub Cube*, in order to further reduce the number of hops required by our lookup service.

References

1. K. Aberer. P-Grid: A Self-Organizing Access Structure for P2P Information Systems. In *Proceedings of the 9th International Conference on Cooperative Information Systems*, 2001.
2. K. Aberer, P. Cudre-Mauroux, M. Hauswirth, and T. V. Pelt. GridVine: Building Internet-Scale Semantic Overlay Networks. In *Proceedings of the 3rd ISWC*, Hiroshima, 2004.
3. R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In *Proceedings of the ACM SIGMOD*, Oregon, USA, 1989.
4. P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data Management for Peer-to-Peer Computing: A Vision. In *Proceedings of the 5th International Workshop on the Web and Databases*, Madison, Wisconsin, 2002.
5. P. Boncz and C. Treijtel. AmbientDB: Relational Query Processing in a P2P Network. In *Proceedings of the International Workshop DBISP2P*. Springer Verlag, 2003.
6. M. Cai and M. Frank. RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network. In *Proceedings of the 13th International WWW Conference*, New York, USA, May 2004.
7. The Chord Project. http://pdos.csail.mit.edu/chord/.

8. V. Christophides, G. Karvounarakis, I. Koffina, G. Kokkinidis, A. Magkanaraki, D. Plexousakis, G. Serfiotis, and V. Tannen. The ICS-FORTH SWIM: A Powerful Semantic Web Integration Middleware. In *Proceedings of the SWDB'03 International Workshop*, Humboldt-Universitat, Berlin, Germany, 2003.
9. V. Christophides, D. Plexousakis, M. Scholl, and S. Tourtounis. On Labeling Schemes for the Semantic Web. In *Proceedings of the 12th International WWW Conference*, Budapest, Hungary, May 2003.
10. A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. Querying peer-to-peer networks using P-trees. In *Proceedings of the 7th International Workshop on the Web and Databases*, 2004.
11. A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, Computer Science Department, Stanford University, 2003.
12. M. Ehrig, C. Tempich, J. Broekstra, F. van Harmelen, M. Sabou, R. Siebes, S. Staab, and H. Stuckenschmidt. SWAP - Ontology-based Knowledge Management with Peer-to-Peer Technology. In *Proceedings of the 1st National "Workshop Ontologie-basiertes Wissensmanagemen"*, 2003.
13. L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt. Locating Data Sources in Large Distributed Systems. In *Proceedings of the 29th VLDB Conference*, Berlin, Germany, September 2003.
14. A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Piazza: Data management infrastructure for semantic web applications. In *Proceedings of the 12th WWW Conference*, Budapest, Hungary, 2003.
15. R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proceedings of the 29th VLDB*

Conference, 2003.

16. H. Jagadish, B. C. Ooi, and Q. Vu. BATON: A Balanced Tree Structure for Peer-to-Peer Networks. In *Proceedings of the International Conference on VLDB*, 2005.
17. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A Declarative Query Language for RDF. In *Proceedings of the 11th WWW Conference*, Honolulu, Hawaii, USA, 2002.
18. A. Magkanaraki, S. Alexaki, V. Christophides, and D. Plexousakis. Benchmarking RDF Schemas for the Semantic Web. In *Proceedings of the First ISWC*, Sardinia, Italy, June 2002.
19. A. Magkanaraki, V. Tannen, V. Christophides, and D. Plexousakis. Viewing the Semantic Web Through RVL Lenses. In *Proceedings of the 2nd ISWC*, 2003.
20. W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Loser. Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks. In *Proceedings of the 12th WWW Conference*, Budapest, Hungary, 2003.
21. T. Pitoura, N. Ntarmos, and P. Triantafillou. Replication, Load Balancing, and Efficient Range Query processing in DHT Data Networks. In *Proceedings of the 10th EDBT Conference*, 2006.
22. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proceedings of the ACM SIGCOMM*, San Diego, CA, USA, August 2001.
23. Resource Description Framework (RDF). http://www.w3.org/RDF/.
24. G. Serfiotis, I. Koffina, V. Christophides, and V. Tannen. Containment and Minimization of RDF/S Query Patterns. In *Proceedings of ISWC*, Galway,

Ireland, November 2005.

25. L. Sidirourgos. Indexing Views to Route and Plan Queries in a PDMS. Master's thesis, University of Crete, Computer Science Department, 2005.
26. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM*, CA, USA, 2001.
27. P. Triantafillou and T. Pitoura. Towards a Unifying Framework for Complex Query Processing over Structured Peer-to-Peer Data Networks. In *VLDB '03 Workshop on DISP2PC*, Germany, 2003.