Month 200X, Vol.21, No.X, pp.XX–XX *J. Comput. Sci. & Technol.*

Building a Distributed Infrastructure for Scalable Triple Stores

Jing Zhou[1], Wendy Hall[2], and David De Roure[2]

[1] *School of Computer Science, Communication University of China, Beijing, China*

[2] *School of Electronics and Computer Science, University of Southampton, Southampton, UK*

E-mail: zhoujing@cuc.edu.cn; {wh, dder}@ecs.soton.ac.uk

Received MONTH DATE, YEAR.

Abstract Built specifically for the Semantic Web, triple stores are required to accommodate a large number of RDF triples and remain primarily centralized. As triple stores grow and evolve with time, there is a demanding need for scalable techniques to remove resource and performance bottlenecks in such systems. To this end, we propose a fully decentralized peer-to-peer architecture for large scale triple stores in which triples are maintained by individual stakeholders, and a semantics-directed search protocol, mediated by topology reorganization, for locating triples of interest. We test our design through simulations and results show anticipated improvements over existing techniques for distributed triple stores. In addition to engineering future large scale triple stores, our work will in particular benefit the federation of stand-alone triple stores of today to achieve desired scalability.

Keywords peer-to-peer, RDF data management, semantics awareness, scalability, topology reorganization

1 Introduction

The Semantic Web is intended to allow people to find, share and integrate information more easily than ever before by giving it well-defined meaning expressed by RDF (Resource Description Framework) and OWL (Web Ontology Language) which is built on RDF and RDF Schema. RDF encodes the meaning in set of "triples", where each triple con-

* This research was primarily conducted while Jing Zhou was affiliated with the School of Electronics and Computer Science, University of Southampton, UK and is supported in part by the Leading Academic Discipline Program, 211 Project for Communication University of China (the 3rd phase).

sists of a subject, predicate and object that correspond to a resource, property and property value. RDF triples form the web of information about resources identifiable on the Web and can be utilized by machines to effectively process information on the Web based on the attached meaning (in the form of metadata). As the development and deployment of practical Semantic Web applications require technologies that support efficient storage and retrieval of RDF data (that is, triples), a number of *triple stores* (e.g. Sesame[1], 3store[2], Joseki (http://www.joseki.org/), Kowari (http://www.kowari.org/) and Jena[3]) came into existence which provide facilities for persistent storage of RDF data by means of a relational database, an XML file or proprietary information repositories.

Most of the triple stores are centralized in the sense that RDF data management occurs at a single place and scalability is achieved by supporting more triples in a single store. The number of triples that are handled by existing triple stores can be up to 1 billion, loading time for which is about several hours[1]. To deal with triple stores hosting triples of higher orders of magnitude, satisfactory solutions have yet to appear on the horizon.

Relying upon uninteroperable centralized triple stores to realize the vision of the Semantic Web, is unrealistic. This is because the real magic of the Semantic Web comes from the fact that there is *one* Semantic Web and from the network effects of sharing (of information), but obviously, centralized triple stores do not facilitate full exploitation of information as expected. Although URIs help identify distributed resources (that is, distribution of data is supported), the centralized nature of triple stores severely limits its extent of distribution of data management which, as well as distribution of data, is considered crucial in removing resource and performance bottlenecks in such systems.

In view of this, decentralized approaches have been proposed using peer-to-peer (P2P hereafter) as the main paradigm for managing RDF data[4,5,6,7,8,9,10,11]. P2P computing[12] is notable for supporting collaboration: it provides individual nodes (or peers) with autonomy in control of their resources and empowers sharing of the resources in a decentralized, scalable and ad hoc fashion. By consolidating their resources (bandwidth, storage capacity and processing power for example), an array of peers in a P2P network are capable of carrying out distributed computing tasks.

In this work (referred to as S-RDF) we pro-

[1] http://esw.w3.org/topic/LargeTripleStores

posed an unstructured P2P architecture (see Section 3.3) for large scale RDF data management systems in which no centralized control is present: the corpus of RDF triples is distributed among peers and each peer is responsible for maintaining and publishing (or advertising) the triples that they would like to share with others. Triples are managed in multiple files, or *RDF data files*, and are only loaded into triple stores when needed. Location of RDF data files is supported by a semantics-directed search protocol (see Section 3.7.1) which makes use of the semantic relationship between resources hosted by neighboring peers to propagate queries. The operation of the search protocol is mediated by topology reorganization that aims to achieve a desirable global performance by adapting the neighborhood of peers locally.

Note that although our approach is based on unstructured P2P, the semantics-directed search (SDS) protocol differs much from canonical searching techniques for unstructured P2P, e.g. flooding and random walk, in that queries are forwarded to peers with a high probability of satisfying the queries in most cases. The anticipated performance gain of our design over existing techniques is confirmed through simulations. In contrast to database researchers[4,5] who focus studies on acquiring semantically correct answers for scalable RDF data management, we explore techniques in large scale triple store to enable efficient query routing. It is assumed in this work that queries propagated to other peers that use different ontologies[13] can be reformulated by employing ontology mapping tools (e.g. [14] and [15]).

We review related work in the following section. The design issues of S-RDF are discussed in Section 3. In Section 4 we present the evaluation results of our work. Finally, we outline some conclusions and future work in Section 5.

2 Related Work

In recent years, there has been a plethora of research on P2P computing and we restrict our discussion to those which aimed to address RDF data management and closely related issues, and others that gave us much inspiration.

Edutella[6] aimed to provide an RDF-based metadata infrastructure for P2P networks based on the JXTA framework[16]. Peers register the queries they may be asked through the query service. Queries are propagated through the network to peers that have registered their interest in this kind of queries. The results are sent back to the requester. In subsequent work on Edutella, Nejdl *et al.*[7] proposed a super-peer-based query routing mechanism with routing indices. Super-peers in the network are ar-

ranged in a hypercube topology that allows for efficient broadcast and search. Kokkinidis *et al.*[8] described a SQPeer Middleware (with two candidate architectures) for routing and planning queries in a P2P network. In the super-peer-based P2P alternative, each peer is connected to at least one super-peer. A peer forwards its corresponding view to a super-peer when it connects to the super-peer. All super-peers are aware of each other in order to answer queries. In SQPeer built on DHT-based structured P2P[17], peers are logically placed according to the value obtained by applying a hash function to their IP address. No single peer has a global knowledge about all peer views and the localization information about remote peer views is provided by a lookup service.

Piazza[9] is one of the very few unstructured P2P systems designed to support data management in Semantic Web applications. Both local point-to-point mappings between small sets of nodes and collaboration through mediated schemas or ontologies are supported. A query is answered by rewriting it using the information, which is captured in mappings, about the relationship between schemas and about data instances. A flooding-like technique was employed to process queries and the designers claimed that they focus mostly on obtaining semantically correct answers.

P2P systems using flooding to propagate queries are notorious for poor scalability since query processing consumes a large amount of bandwidth as the network size increases. For scalable solutions, Lv *et al.*[18] proposed using multiple random walk and demonstrated that, with a fixed number of random walkers, the algorithm can locate the data object of interest almost as quickly as Gnutella's flooding while reducing the network traffic by two orders of magnitude in many cases at the expense of a slight increase in the number of hops.

Cai and Frank[10] proposed a scalable RDF repository called RDFPeers in which a triple is stored at three places in a multi-attribute addressable network. They extended Chord[19] by applying hash functions to the subject, predicate and object values of the triple. RDFPeers was demonstrated to provide very good scalability and fault resilience due to its roots in Chord.

Stuckenschmidt *et al.*[11] presented an architecture for querying distributed RDF repositories by extending the Sesame system[1]. Well-understood database techniques were borrowed to optimize queries against distributed RDF repositories. Their work focused on resolving queries in a limited scenario in which queries were sent from a single query originator and query answers might only reside on one-hop away neighbors.

The five approaches mentioned above (except [18]) were primarily intended to address RDF data management and associated issues in P2P environments. Among others, SQPeer (adopting a structured P2P architecture) and RDFPeers provide the best scalability because of the use of DHTs. However, as we will discuss in Section 3.3, DHTs-based solutions place severe limits on the network topology and the placement of the resources in the system, which makes them unable to cater for S-RDF that allows peers to maintain their own triples. Although Edutella, Piazza and the architecture presented in [11] solved this problem, Edutella (super-peer-based) imposed a deterministic hypercube shape on P2P networks, thus generating an overhead to establish it, and none of the others focused on scalable schemes for query routing as S-RDF did.

Our work was much motivated by Gia[20]. In view of the natural heterogeneity present in most P2P systems, Chawathe *et al.* proposed new mechanisms, including dynamic topology adaptation, active flow control, one-hop replication of pointers to content, and biased random walk-based search, to improve the scalability of Gnutella-like P2P systems. The aggregate of these design components was demonstrated to provide three to five orders of magnitude improvement in the total capacity of the resultant system with significant robustness to failures.

Also directly relevant is the work of Haase *et al.*[21] They proposed using expertise-based selection of peers and ontology-based matching with a similarity measure to improve the search performance of P2P systems. Queries were forwarded to the *best* n peers on a peer list. Dynamic semantic topologies and scalable search mechanisms have yet to be incorporated into their model. In this regard, our efforts in this paper complement their work. The technique of expertise-based selection of peers was later adopted by Bibster[22], a semantic-based P2P system built upon the JXTA platform, for exchanging bibliographic metadata.

Two other pieces of work examined the use of the shortcut-based search technique to implement efficient content location in P2P systems. Menascé[23] presented a probabilistic search protocol that made use of a directory cache at peers. When a resource of interest was found, a *ResourceFound* message was sent along the path that the query message had traversed until it reached the requester. This message updated the directory cache at every peer it visited. Sripanidkulchai *et al.*[24] proposed building the interest-based shortcuts on top of P2P systems. Peers sharing similar interest created shortcuts to one another and used these shortcuts to locate content in the first

place. When shortcuts failed, they resorted to the search mechanism provided by the underlying P2P system. Our work utilizes the variation of the aforementioned shortcuts to achieve efficient resource location, making peers with similar resources closer to one another through topology reorganization (see Section 3.6).

3 S-RDF

In this section, we begin with a discussion on how to disintegrate a large scale RDF repository and describe its fragments. Then, we detail the proposed architecture for S-RDF and describe the querying scenarios of S-RDF. Finally, we present the primary components of the solution to S-RDF: ontology-based matching, topology reorganization, and the SDS protocol.

3.1 Disintegrating Large Scale RDF Repository

Contrary to common practice of maintaining hundreds of millions of triples in a centralized triple store, we believe storing triples in multiple RDF data files, organizing the files in a semantics-aware hierarchy, and loading files or merely groups of triples into a triple store on demand will mitigate the inherent scaling problem in centralized triple stores and facilitate efficient resource discovery.

One can break down a very large RDF repository at several levels of granularity, ranging from a single triple and the URIrefs it comprises to a snippet (that is, a collection of triples with a common subject and made within a particular context)[25]. The level of granularity is closely related to the way the fragments of the RDF repository can be well described and efficiently queried.

Our previous work on unstructured P2P hypermedia link services[26] in which each resource was characterized by means of a number of *topics* that best represented its content, gives us insight into how to split and describe RDF data files in S-RDF and to perform efficient search over them. As the subject in an RDF triple similarity with the concept of topic that we used in [26], we assign triples to RDF data files in response to the type of their subject[2]. This, however, should not be interpreted as obviating the use of predicates or objects to group triples[3].

[2] Sayers and Wilkinson[25] adopted a similar granularity based on a combination of subject and context for caching and distribution of RDF triples.

[3] Property tables and vertical paritioning have shown to outperform the standard triple store approach by more than a factor of 2 and have superior scaling properties in a centralized setting[27]. However, for a decentralized RDF triple store like the S-RDF, grouping RDF triples based on subject seems more viable in terms of supporting efficient query resolving and routing, and thus can help deliver desirable scalability as demonstrated in this work.

```
<akt:Researcher-In-Academia rdf:about="http://194.66.183.26/WEBSITE/GOW/ViewPerson.aspx?Person=6610">
  <akt:family-name>Hall</akt:family-name>
  <akt:has-appellation rdf:resource="http://www.aktors.org/ontology/portal#Prof" />
  <akt:full-name>W Hall</akt:full-name>
  <akt:works-for rdf:resource="http://www.hesa.ac.uk/#H-0160" />
  <akt:works-in-unit rdf:resource="http://194.66.183.26/WEBSITE/GOW/ViewDepartment.aspx?Department=750" />
  <akt:has-telephone-number>+442380592388</akt:has-telephone-number>
  <akt:has-fax-number>+442380592865</akt:has-fax-number>
  <akt:has-email-address>wh@ecs.soton.ac.uk</akt:has-email-address>
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#4" />
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#23" />
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#31" />
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#45" />
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#61" />
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#62" />
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#67" />
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#97" />
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#86" />
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#88" />
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#123" />
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#145" />
  <akt:has-research-interest rdf:resource="http://194.66.183.26/WEBSITE/GOW/research_topics_def.htm#173" />
</akt:Researcher-In-Academia>
```

Figure 1: Snapshot of statements describing a common subject resource

The practice of grouping RDF triples based on subject can also be seen in 3store[2]. For instance, the statements in Fig. 1, extracted from the file at http://triplestore.aktors.org/data/EPSRC/epsrc-people.rdf, describe the properties of a subject resource specified by http://194.66.183.26/WEBSITE/GOW/ViewPerson.aspx?Person=6610 and this resource is an instance of *acm:Research-In-Academia*.

The subject-based split of an RDF triple repository may be accompanied by some issue. Suppose if a query involves a path in the RDF graph of length N, resolving the query would typically entail location of at least N different sources. To what extent this problem would affect the overall performance of RDF query resolving (only if these sources are associated with distinct peers), depends on the fraction of such queries that is different from application to application. The overhead incurred can be partially reduced by efficient search mechanisms.

3.2 Describing RDF Data Files

Apart from splitting a large scale RDF repository with a fine-grained level, describing its fragments (that is, the RDF data files) using characteristic terms is also crucial in achieving

increased scalability since we can then develop a search mechanism to locate and load associated RDF data files.

We use the subjects of the triples that an RDF data file contains to describe the file since a large scale triple repository is divided based on the subject of triples. This process can be carried out by the *description generator*, an automatic tool we developed for this work. The description generator takes an RDF data file as input, parses it, and extracts distinct types of subjects to generate a description of the file.

In certain cases, the description generator may resort to ontologies for analyzing the relationship between the types of subjects and then select the most appropriate one to describe an RDF data file. For instance, Figure 2 presents the description of a peer's RDF data files that contain triples describing three different kinds of resources. Of all resources, we observed that the one identified by http://triplestore.aktors.org/data/EPSRC/epsrc-institutions.rdf needs to be examined further. In this file, some resources are instances of class *akt:Organization*[4], whereas others are instances of *akt:University*. We eventually chose *akt:Organization* as the descriptive term since the related ontology at http://www.aktors.org/ontology/portal shows that *akt:University* is a subclass of *akt:Organization*. There can be more than one term used to describe an RDF data file.

3.3 Selecting a Software Architecture

The primary goal of this work is to explore scalable techniques to eliminate resource and performance bottlenecks in large scale triple stores. To this end, we chose the unstructured P2P[5] as the architecture of the S-RDF since it helps satisfy our requirements.

Of all P2P systems, structured P2P accomplishes satisfactory scalability by employing the DHT (Distributed Hash Table) abstraction. However, Chawathe *et al.*[20] discovered in P2P file sharing systems that peer clients were extremely transient and the high rate of churn would cause significantly more overhead for structured P2P systems than for those based on an unstructured overlay network (e.g. Gnutella). Moreover, the network topology in structured P2P is assumed to be tightly controlled and the placement of resources is

[4] http://www.aktors.org/ontology/portal#Organization

[5] P2P systems can be simply divided into hybrid P2P (e.g. Napster), unstructured P2P (e.g. Gnutella), and structured P2P (e.g. CAN[28], Chord[19] and Pastry[29]). This classification may not be as rigid as it used to be. However, it emphasizes the different ways resource discovery is carried out in P2P systems and is sufficient to distinguish our approach from others.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:ex="http://www.example.com/">
      <ex:rdf-data-file rdf:about="http://triplestore.aktors.org/data/ccs98.rdf">
           <ex:subject rdf:resource="http://www.acm.org/class/1998/Research-Area"/>
      </ex:rdf-data-file>
      <ex:rdf-data-file rdf:about="http://triplestore.aktors.org/data/EPSRC/epsrc-people.rdf">
          <ex:subject rdf:resource="http://www.aktors.org/ontology/portal#Researcher-In-Academia"/>
     </ex:rdf-data-file>
     <ex:rdf-data-file rdf:about="http://triplestore.aktors.org/data/EPSRC/epsrc-institutions.rdf">
          <ex:subject rdf:resource="http://www.aktors.org/ontology/portal#Organization"/>
     </ex:rdf-data-file>
</rdf:RDF>
```

Figure 2: Example descriptions of a peer's RDF data files

precisely determined. We believe that an unstructured P2P paradigm can better model our collaborative scenario in which peers maintain *their* triples and share them with others.

A large scale RDF triple repository to be disintegrated may involve triples belonging to multiple users. In this case, triples are always allocated to their own users in the first place. They are then locally partitioned on the basis of their subject.

3.3.1 S-RDF Architecture

S-RDF is based on unstructured P2P and the individual components of each peer include the user interface, the query processor, the router, the triple store, and the repository for local RDF data files as in Fig. 3. The user interface interacts with the query processor which in turn deals with the repository of RDF data files. The query processor is an important component in S-RDF and it hides the details of query execution from the users—users specify the result whilst the query processor determines how this result is obtained. The router gives each peer a single interface to the peer network and it handles all messages going to and arriving from other peers. The triple store loads local and remote RDF data files of interest for querying when needed.

A user query can be one looking for desirable resources or another that tries to identify the relationship between resources. Which kind of queries can be satisfied is subject to the capabilities that the involved triple store provides. Figure 3 describes a process in which a user query posed on one of the peers (that is, peer p_i) in S-RDF is satisfied. Upon receiving a query Q (1), the user interface of p_i wraps it and passes

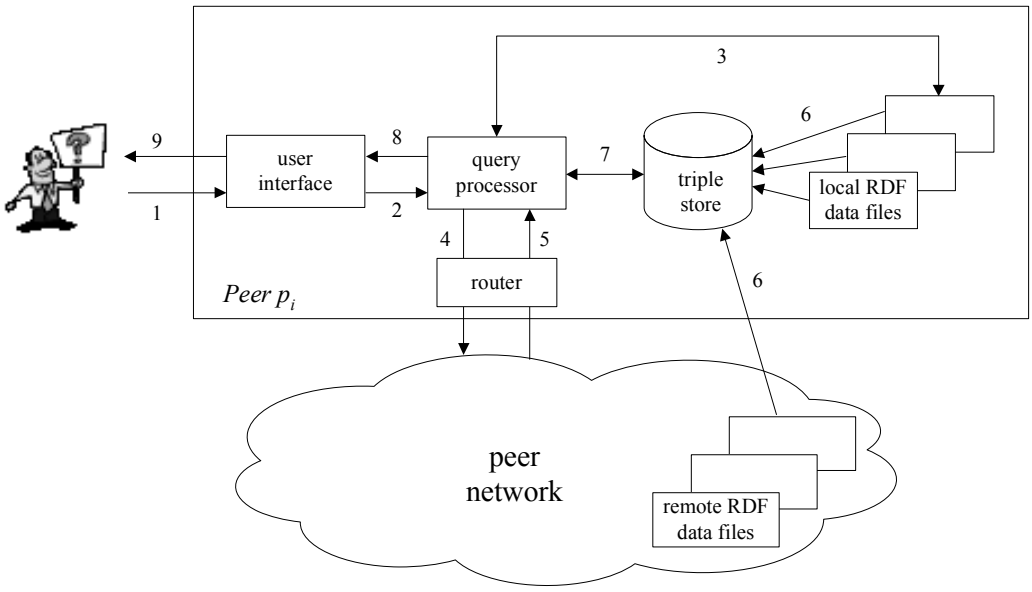

Figure 3: Process of resolving a user's query in S-RDF

it onto the query processor (2). The query processor parses the incoming query and may convert it to several sub-queries $q_1, q_2, q_3, \cdots, q_n$ (e.g. conjuncts, $Q = q_1 \cap q_2 \cap q_3 \cap \cdots \cap q_n$)[6]. Against these sub-queries, the query processor examines the local RDF data files and discovers the targets (3). Meanwhile, these sub-queries are propagated in the network (4) conforming to a specified protocol (see Section 3.7.1). Once remote desirable RDF data files are located, their information is returned to the query processor (5). The query processor instructs the triple store to load all the located (local and remote) RDF data files for query resolution (6), collects the result (7), and presents it to the user (9) through the user interface (8). As operating in a decentralized environment, S-RDF can provide access to the RDF data files via query languages of choice such as SPARQL[7].

The peer network shown in Fig. 3 is an application-level overlay on top of the physical network. To join the network, a peer initially connects to several peers (already on the network) known by out-of-band mechanisms. The

[6] Query decomposition is a big research issue for RDF data and there is no trivial answer on how to decompose an RDF query into conjuncts. How S-RDF deals with query decomposition is what is currently supported and we leave in-depth research into this issue to future work.

[7] http://www.w3.org/TR/rdf-sparql-query/

Table 1: Example neighbor table of p_i with dscriptive terms A, B, C, D, E and G

neighbor	*descriptive terms*	*PP_ON*
p_j	A, B, C, D	4
p_k	B, E, F, G	3
p_m	C, D, H, I, J	2

peer exchanges the descriptive terms of its RDF data files with each of the new neighbors and establishes its neighbor table.

A peer's neighbor table comprise a number of entries (see Table 1 for an example). Each entry records the identifier of a neighbor, the descriptive terms of the neighbor, and *PP_ON*. PP_ON is defined as the number of the peer's descriptive terms that are semantically related to those of its neighbor. Table 1, for instance, reveals that p_i and p_k have 3 semantically related descriptive terms in common.

The departure of a peer results in a notification sent to its neighbors. The neighbors then update their neighbor table by searching for and discarding the entry that involves the leaving peer. If a peer does not inform neighbors before leaving, the latter can still detect the departure and remove the related entry from neighbor tables since all peers periodically probe their neighbors.

3.4 Querying for Data

We consider two querying scenarios of S-RDF. First, users may only query the triple store for a list of triples that satisfy their needs using the following function:

$$getTriples(s, p, o)$$

where s, p, and o are the subject, predicate, and object of a triple that can be either concrete URI references or literals (for the object only). Moreover, s, p, and o can be a wildcard '$*$' that indicates any URI reference or literal would match here. According to the information provided in the function, the query process formulates an internal query (contrary to the external query from users) and propagates it to the rest of the network. In this scenario, peers would return triples they have that match the request. Note that we assign triples to RDF data files according to the type of their subject, and therefore a query that involves retrieval of triples in the form of ($*$, p, o) may use flooding-like query routing in the worst case[8].

[8] To reduce the significant message overhead in flooding, other techniques, such as expanding ring in [18], can be adopted instead.

Second, a user can issue a complex request to a triple store that requires multiple remote RDF data files be located and then loaded into the local triple store prior to query resolving[9]. Under such a circumstance, the query processor is responsible for extracting useful information from the original user query and formulating an internal query to locate those RDF data files. Still, due to the way we allocate triples to RDF data files, users are suggested to provide in their queries information about pontential subjects in the desirable RDF data files. If no such information is available, flooding-like query routing or its more efficient variants will be conducted. Peers would return information (e.g. URI references) about the RDF data files of interest (rather than triples) to the query processor which subsequently instructs the triple store to load those files.

3.5 Ontology-supported Matching

In the context of S-RDF, matching is a process that determines the extent to which a query and the descriptive terms of a peer, or the descriptive terms of a pair of peers, are similar. Typically, this involves a similarity function that produces a numerical number to indicate the similarity between two data items being compared. As we will see in Section 3.7.1, matching plays a significant role in deciding to which neighbor(s) a peer should forward its queries.

Both queries and the descriptive terms of peers in S-RDF comprise terms from existing ontologies and these terms may have well-defined relationships among them. Hence, we chose to measure the semantic, other than syntactic, similarity in the matching process. We assume in S-RDF that, by relying on ontology mapping[10] tools, peers are able to identify the relationship between any terms.

The primary relationship between any terms is that of "being semantically related". Two terms, t_1 and t_2, "being semantically related" means that class C_1 represented by t_1 is the same as, or a subclass of, or a superclass of, or sharing the same ancestor with, class C_2 represented by t_2. The "same as", "subclass of", "superclass of" and "sharing the same ancestor with" relations are discovered by reasoning over associated ontologies. A match is found if any descriptive term in a query is semanti-

[9] For instance, as the FROM keyword in a SPARQL query identifies the data repository against which the query will be run, a SPARQL query may include multiple FROM keywords as a means to assemble larger RDF graphs for querying.

[10] A process whereby two ontologies are semantically related at the conceptual level and the source ontology instances are transformed into the target ontology entities according to these semantic relations.

cally related to at least one descriptive term of a peer. Our similarity function delivers a numerical value of n for a match if the number of the semantically related terms between a query and a peer's description (that is, overlap) is equal to n. This also applies to matching between the descriptive terms of a pair of peers.

3.6 Topology Reorganization

Peers may accept new neighbors or discard old ones, resulting in different topologies. This act is launched by the proactive behavior of peers and is termed as topology reorganization that alters the virtual neighborhood of peers. In this work, we anticipate to employ reorganization techniques to deliver an improved performance in resource discovery.

Reorganization occurs at a specified interval. Before the process starts, each peer examines whether the percentage of times it has successfully answered incoming queries exceeds a threshold. If not, the peer participates in reorganization. During reorganization, each involved peer discovers a set of neighbors known by out-of-band mechanisms or by exchanging neighbor information with others. These neighbors are most qualified to help achieve the objectives of reorganization. Within its capacity, the peer replaces some (or all) of its current neighbors with the new ones by updating its neighbor table.

We conjectured that by putting peers with semantically related descriptive terms together, a peer that answered a query successfully will be able to direct subsequent queries to their targets more easily, thus better performance could be achieved. However, if a peer shares no semantically related terms with any other peer, it should be closer (in the number of hops) to those that are able to answer future queries from the peer with high probability. Heuristics have been developed (see Section 3.8) to facilitate reorganization that is targeted at a better performance of the SDS and their efficiency is demonstrated in Section 4.3.

An evaluation metric taking these factors into account is defined as *potentiality*. We use potentiality as a relative metric to evaluate peers' capability of satisfying future queries. The higher the potentiality of peer p_i with respect to neighbor p_j is, the more likely p_i will successfully answer the future queries from p_j. Potentiality is an important concept used not only in topology reorganization but also in the SDS protocol described in the following section. We delay the detailed discussion on how to estimate its value until Section 3.8.

3.7 Developing an Efficient Search Protocol

Most research on RDF data management in P2P environments (see Section 2) focuses on producing *efficient* query routing in terms of high recall levels rather than high scalability, that is, the ability of the solution to handle increased volume or complexity.

In the context of S-RDF, the efficiency we pursue in searching remote RDF data files is reflected in metrics including a high recall, a small number of hops, and a low system load (e.g. messages/query and messages/node, see Section 4.2). Ideally, the search protocol should be able to handle a larger number of peers without significant performance degradation.

3.7.1 Semantics-directed Search Protocol

The search protocol in S-RDF was inspired by multiple random walk[18] in which a query message (that is, a walker) is forwarded to a randomly chosen neighbor at each step until the object of interest is found. We, however, believe that in some P2P systems, resources may be related to one another. For instance, RDF data files (or their descriptive terms) may have semantic relationship among themselves. We intend to take advantage of this relationship to *guide* query routing in S-RDF. In the SDS protocol presented in Fig. 4 and Fig. 5, we define QP_ON as the number of overlap of query terms and a peer's descriptive terms.

Figure 4 presents the procedure of query resolving and routing at query originators. Initially, query originator p_0 checks its local RDF files for potential matches and any applicable results are returned. If p_0 has no neighbor with QP_ON > 0, it will broadcast the query message to all neighbors. Occasionally, more than 1 neighbor, which shares the highest potentiality (that is, best satisfying queries of p_0 in the past), will receive the broadcast message. Of all the copies of the message, the one sent to a randomly chosen peer, say p_j, is labelled as COP (Continue tO be Propagated). However, if p_0 has at least 1 neighbor with QP_ON > 0, copies of the query message are forwarded to all such neighbors. Among all the messages sent, only one copy is marked as COP and it is the message sent to any of the neighboring peers, say p_k, with the unique greatest PP_ON. This is because, according to the principle of topology reorganization, peers with semantically related descriptive terms should be grouped together as neighbors, and directing a query to one peer with a greater PP_ON will potentially lead to more targets to be discovered than forwarding it to another with a very little PP_ON. The field

p_0 checks local RDF data files for matches;
Applicable results are returned;
if all neighbors P_{all} of p_0 have QP_ON > 0 **or**
    all neighbors P_{all} of p_0 have QP_ON $= 0$, **then**
  Add p_0 to $P_{broadcast\_peers}$ in the query message;
  Randomly choose a peer p_j from the neighbors sharing the unique highest potentiality;
  The query message is labelled as COP;
  p_0 sends a copy of the query message to p_j;
  for each p_i in $P_{all} - \{p_j\}$
    the query message is labelled as NONE_COP;
    p_0 sends a copy of the query message to p_i;
  end_for
else if some of p_0's neighbors P_{set} have QP_ON > 0, **then**
  Randomly choose a peer p_k from the neighbors (in P_{set}) sharing the unique
    greatest PP_ON;
  The query message is labelled as COP;
  p_0 sends a copy of the query message to p_k;
  for each p_i in $P_{set} - \{p_k\}$
    the query message is labelled as NONE_COP;
    p_0 sends a copy of the query message to p_i;
  end_for
end_if

Figure 4: Query resolving and routing of query originator p_0

of *broadcast_peers* in the query message is reserved for prevention of loops. If the retransmission of query messages involves all neighbors of a peer, then the identifier of the peer is added to set $P_{broadcast\_peers}$ and the subsequent message recipients will exclude nodes recorded in $P_{broadcast\_peers}$ when making decisions on the next hop node to forward queries.

The procedure of query resolving and decision-making on query routing at query routers[11] is described in Fig. 5. The recipient of a query message, p_0 for example, first checks for matches against its local RDF data files and sends results directly to the query originator. Subsequently, this peer needs to decide whether it should further propagate the query message. The peer examines all neighbors not in $P_{broadcast\_peers}$, that is, P_{all} - $P_{all} \cap P_{broadcast\_peers}$, and tries to find those with QP_ON > 0. Copies of the message will be forwarded to all neighbors that can satisfy the query. If the query message is a COP message, then the copy of the message sent to the neighbor with the greatest PP_ON should be labelled as COP. If more than one neighbor has the unique greatest PP_ON, we can randomly select one of them and label the message it receives as COP. However, if no such neighbor (with QP_ON > 0) exists, a NONE_COP message receiver will remain silent whereas a COP message recipient will broadcast the query message to all neighbors. Again, a randomly chosen neighbor with the highest potentiality (that is, best satisfying queries of p_0 in the past) will receive a COP message. The maximum number of hops a query message can be relayed is specified by a TTL (Time-To-Live) tag attached to the message.

The SDS protocol specifies that when a query message is broadcast to all applicable neighbors, only the copy sent to one (can be randomly chosen) of the peers with the highest potentiality/greatest PP_ON should be labelled as a COP message. The reason is that we intend to discover all peers having the answer to the query whilst consuming as little system resources as possible. We anticipate to achieve this by only allowing peers that may lead to other targets to further broadcast the message when necessary.

3.8 Revisiting the Concept of Potentiality

As we can see, both topology reorganization and the SDS protocol rely upon the metric of potentiality. One peer having a higher poten-

[11] In the context of S-RDF, a query router refers to the peer that determines the next-hop peer to which queries received should be forwarded.

p_0 checks local RDF data files for matches;
Applicable results are sent back from p_0 to the query originator;
$P_{set} := P_{all}$ - $P_{all} \cap P_{broadcast\_peers}$;
Add all peers in P_{set} with QP_ON > 0 to P_{some};
if $P_{some} \subset P_{set}$, **then**
    if the query message is labelled as COP, **then**
        Randomly choose a peer p_j from the neighbors (in P_{some}) sharing the unique greatest PP_ON;
        The query message is labelled as COP;
        p_0 sends a copy of the query message to p_j;
        for each p_i in $P_{some} - \{p_j\}$
            the query message is labelled as NONE_COP;
            p_0 sends a copy of the query message to p_i;
        end_for
    end_if
else if $P_{some} = P_{set}$ **or** $P_{some} = \phi$, **then**
    if the query message is labelled as COP, **then**
        Add p_0 to $P_{broadcast\_peers}$ in the query message;
        Randomly choose a peer p_k from the neighbors (in P_{set}) sharing the unique highest potentiality;
        The query message is labelled as COP;
        p_0 sends a copy of the query message to p_k;
        for each p_i in $P_{set} - \{p_k\}$
            the query message is labelled as NONE_COP;
            p_0 sends a copy of the query message to p_i;
        end_for
    end_if
end_if

Figure 5: Query resolving and routing of query router p_0

tiality with respect to another indicates that the former will answer future queries from the latter with high probability. Under different circumstances, potentiality should be computed in different ways. We developed the following heuristics to help evaluate the metric in the aforementioned cases.

In topology reorganization, p_j gives the highest potentiality to p_i if p_i has the greatest PP_ON with p_j, or, although p_i has 0 PP_ON with p_j, it can best answer queries from p_j in the past.

In semantics-directed search, we identify two situations under which p_j considers p_i as a neighbor with the highest potentiality.

- p_i has the greatest PP_ON with p_j if p_j satisfies the incoming query.
- p_i can best satisfy the queries encountered by p_j in the past if p_j does not satisfy the incoming query.

The rationale behind the second idea in both cases is that we assume the recent past can approximate the immediate future. Hence, if p_i can best satisfy queries received by p_j, it is very likely that p_i can also answer future queries from p_j.

To determine the extent to which p_i can satisfy queries encountered by p_j, we developed a data structure named query history. Query history is a collection of all queries a peer has encountered over a period of time. It is realized as a FIFO (First In First Out) queue. Each entry of query history includes the query identifer, the query, and the arrival time of the query. The set of query identifiers is Q, the capacity of query history of peer p_i is ${h^{max}}_i$ and the set of arrival time of queries is A. Query history of p_i can be represented by $H_i = \{({q^m}_i, {h^m}_i, {a^m}_i) | {q^m}_i \in Q, {h^m}_i \in T, {h^m}_i \times {h^m}_i \subseteq T, {a^m}_i \in A, 0 < m \leq {h^{max}}_i\}$. The oldest entry of the query history is discarded when the queue is full. By comparing the descriptive terms of p_i against all the entries in query history of p_j, we can determine the degree to which p_i can answer queries propagated to p_j in the past.

4 Simulations

We opted for a simulation study on scalability of S-RDF since the testbeds we have available cannot cope with the network sizes (up to 10,000) that we simulate in this section.

A number of simulations for each combination of different search protocols and the parameters described in Section 4.1, were conducted. We anticipated to see performance improvements that S-RDF (in particular its semantics-directed search mechanism) brings to flooding-based P2P networks such as Gnutella.

Moreover, we expected that bringing semantics awareness into the search method in S-RDF would enable better performance to be delivered than that of random walk. Since techniques, such as shortcuts/interest-based locality, were extensively utilized to implement semantic query routing in P2P networks, we were keen to examine whether incorportation of these approaches would further enhance the scalability of S-RDF. Results reported in Section 4.3 correspond with our anticipation and provide answers to our questions.

4.1 Methodology

By varying each of the parameters in Table 2, a number of network conditions can be simulated for testing purposes. We compared semantics-directed search with two well-established mechanisms: multiple random walk and constrained flooding[12], owing to the multiplicity of their variants for efficient search in P2P networks and their use as baseline for a large number of performance comparison. Also, we selected interest-based locality (see Section 2) as another baseline for comparison.

We started simulations with P2P networks the topology of which was generated using network topology generator Inet 3.0[30]. The simulations on the SDS used topology reorganization to reconfigure this initial topology. For multiple random walk, constrained flooding, and interest-based locality, there is no topology adaptation; the initial topology remained unchanged throughout the entire experiment.

We presented in Section 3.8 that both reorganization and the SDS employ analogous heuristics to calculate potentiality. To compute potentiality, one of the heuristics (getGreatestPP_ON) examines the value of PP_ON between a pair of peers and another (getHighestPotentiality) involves using all entries in query history of a peer captured during a specified period of time (time window). We were keen to investigate the impact of both heuristics on determining potentiality and further the SDS performance, so, for comparison purpose, we developed a third one (getRandomNeighbor) in which a neighbor peer is randomly chosen. Various combinations of these three heurisitics were applied to the SDS.

To test the variations in the system load as the network size increases, we generated networks of 3037[13], 5000, and 10,000 peers, respectively, using Inet 3.0.

[12] In constrained flooding, floods are constrained by means of a TTL field in the query message that is decremented every time the query is forwarded.

[13] The generator should be used to generate a network of no less than 3037 nodes, which is the number of Autonomous Systems on the Internet in November 1997.

Table 2: Configuration parameters

name	*description*
search method	The way that specifies how search queries are forwarded and served within a network
network size	The number of all peers in a network
resource replication	The distribution of the instances of resources across all peers
replication ratio	The fraction of peers that store the instances of resources following a discrete uniform distribution
query distribution	The distribution of the instances of resources across all queries issued within a specified period of time
minimum degree	The minimum number of neighbors (for reorganization use)
walker number	The number of query messages sent to a randomly chosen neighbor at each step (for multiple random walk use)

Our practical experience on unstructured P2P networks[31] has shown that the distribution of resource instances, the distribution of queries, and their combinations have an explicit performance impact on the networks. Hence, we took this into account and defined the distribution of resource replication as the distribution of the resource instances across all peers. This distribution may follow Gaussian distribution, Zipf distribution, and the discrete uniform distribution. Also, we characterized the query distribution by the distribution of the resource instances across all queries issued within a specified period of time and this distribution may follow discrete uniform distribution and Zipf distribution[14].

To understand caching and replication strategies for development and deployment of large scale triple stores, we also investigated the performance of the SDS protocol that enables caching of query results (see Fig. 8).

In all experiments, queries are single resource-based and the result for multiple resource queries can be deduced as we assume each resource is semantically independent of others. Our simulator generated a series of queries at the same rate across all the experiments and randomly selected a peer to be the originator for each query. We set TTL for all queries to 60.

[14] Unlike others using real-world datasets to test systems and algorithms, we made use of synthetic but reasonable datasets to keep our evaluation independent of any particular application.

4.2 Performance Metrics

We evaluate our design against others by the extent to which scalability can be obtained as measured by several metrics described below. These metrics are intended to capture the fundamental properties relevant to this comparison.

recall: The fraction of relevant resources that are retrieved by a search.

hops: The delay in finding all answers as measured in number of hops.

messages/query: The number of search messages (exclusive of control messages) generated in search of the answers to a query.

messages/node: The overhead of the semantics-directed search algorithm as measured in number of search messages each node has to process.

peak # messages: The maximum number of search messages in the message queue that any node has to process.

4.3 Results

We now describe the result of applying the basic metrics we have chosen to specific instances of generated P2P networks. All experimental results in this section are averaged over 20 runs[15].

4.3.1 Performance Comparison

Table 3 presents a comparison among four techniques: the SDS, interest-based locality, multiple random walk, and constrained flooding[16]. It can be seen that in the SDS the recall is proportional to the minimum degree allowed for each peer. Meanwhile, as the minimum degree increases, more connections are set up between peers, thus shortening the number of hops to locate all targets. To achieve the same level of recall, 0.78 for example, multiple random walk (2048 walkers) generates 711.24 messages per node and 30952.00 peak # messages, whereas the SDS only incurs 87.28 messages per node and 694.33 peak # messages.

We also found that multiple random walk is

[15] For the sake of space, we omit the standard deviation of all metrics that shows the performance of S-RDF is almost always close to its average. For instance, the standard deviation of recall in Fig. 7 ranges from 0.0045 to 0.0158 and the standard deviation of hops from 0.2023 to 0.7387.

[16] We observed that topology reorganization (which also involves increasing the minimum degree for peers) has a very little impact on the performance of interest-based locatlity, multiple random walk and constrained flooding and therefore omitted their corresponding results.

[17] We performed the experiments on MRW with 256, 512, 1024 and 2048 walkers, respectively, and the data on the right (from the top to the bottom lines) captures the corresponding result.

Table 3: A comparison between the SDS, multiple random walk (MRW), constrained flooding (C-FLD), and interest-based locality (IBL), max. degree = 684

	median degree	*avg. degree*	*min. degree*	*recall*	*hops*	*msgs/query*	*msgs/node*	*peak # msgs*
	28.50	4.39	3	0.66	10.94	950.06	46.30	681.00
	29.50	6.17	5	0.71	11.03	1323.08	64.48	681.33
	30.50	8.04	7	0.78	9.92	1790.99	87.28	694.33
	31.50	9.96	9	0.81	8.63	2138.70	104.22	1379.33
SDS	32.50	11.90	11	0.83	6.66	2574.15	125.44	2117.67
	33.50	13.85	13	0.88	6.66	2976.59	145.06	2001.00
	35.00	15.80	15	0.88	6.38	3342.91	162.91	970.00
	37.00	17.77	17	0.89	5.87	3746.67	182.58	710.33
	38.50	19.74	19	0.91	5.10	4098.75	199.74	1196.67
	27.50	3.15	1	0.27	7.20	1856.48	90.47	257.00
MRW[17]	27.50	3.55	1	0.41	7.32	3684.68	179.56	1078.67
	27.50	3.15	1	0.59	7.10	7379.83	359.64	9038.67
	27.50	3.15	1	0.78	6.64	14594.82	711.24	30952.00
C-FLD	27.50	3.15	1	1.00	3.64	8284.43	403.72	32119.00
IBL	27.50	3.15	1	1.00	3.61	8284.47	403.72	49218.00

not necessarily more scalable than constrained flooding. In our experiments, constrained flooding can lead to a recall level of 1.00 at the cost of 403.72 messages per node. However, multiple random walk (with 2048 walkers) creates nearly twice as much the number of messages per node as constrained flooding but only obtains a recall level of 0.78.

By comparing the data from constrained flooding and interest-based locality, we discovered that in our experimental settings interest-based locality did not reduce a significant amount of flooding as claimed in [24]. We attribute this to the small number of queries issued within our experiments. According to its principles, efficient content location using interest-based locality is achieved by allowing peers that share similar interests to create shortcuts to one another. Typically, this requires that each peer accumulate sufficient shortcuts before they can efficiently route queries and locate content of interest. When we doubled the number of queries issued during an experiment on interest-based locality, the number of hops was decreased and the number of messages per query remained at a similar level, whereas the number of messages per node was doubled[18]. The advantage of the SDS over interest-based locality is that the former, by using topology reorganization, can deliver a comparable level of recall to the later with much lower system load incurred.

We applied the SDS to simulated P2P networks of different scales and collected experimental data in Fig. 6. The performance of the SDS degrades gracefully as the network size rises—the level of recall drops from 0.88 in one network of 3037 nodes to 0.79 in another of 10000 nodes whilst the load on each node, as indicated by the number of messages per node, remains nearly constant.

4.3.2 Impact of Heuristics on Reorganization and the SDS

A number of combinations of heuristics (getGreatestPP_ON, getHighestPotentiality and getRandomNeighbor) are listed in Table 4. We applied them to topology reorganization and the SDS in the simulation, and plotted the result in Table 5.

Table 5 indicates that using both getGreatestPP_ON and getHighestPotentiality does not necessarily deliver an enhanced performance for the SDS. By comparing results of heuristics 1, 2 and 3 we observed that, under certain circumstances, randomly selecting a peer to be a new neighbor (in reorganization) or to forward queries to (in the SDS) can bring a higher level

[18] Results are omitted from Table 3.

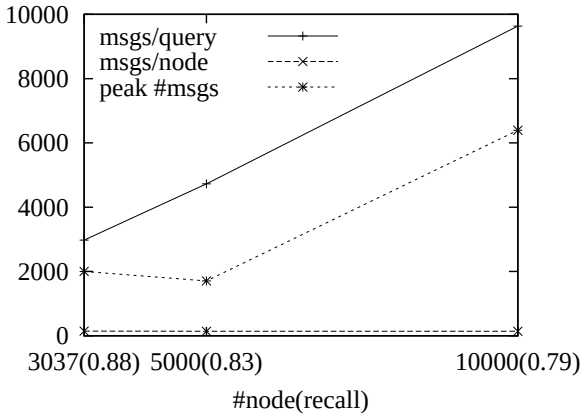

Figure 6: Applying the SDS to networks of different scales, min. degree = 13

Table 4: Heuristics applied to SDS

name	*method*
heuristic 1	substitutes getRandomNeighbor for getGreatestPP_ON and getHighestPotentiality
heuristic 2	substitutes getRandomNeighbor for getHighestPotentiality only
heuristic 3	employs getGreatestPP_ON and getHighestPotentiality (148 entries in query history)
heuristic 4	employs getGreatestPP_ON and getHighestPotentiality (226 entries in query history)
heuristic 5	employs getGreatestPP_ON and getHighestPotentiality (310 entries in query history)
heuristic 6	employs getGreatestPP_ON and getHighestPotentiality (406 entries in query history)

Table 5: Impact of various heuristics on SDS performance, min. degree = 13

method	*recall*	*hops*	*msgs/node*	*msgs/query*
heuristic 1	0.90	6.29	153.53	3161.09
heuristic 2	0.90	6.31	154.17	3163.68
heuristic 3	0.88	6.66	145.06	2976.59
heuristic 4	0.92	6.08	198.88	2672.60
heuristic 5	0.94	5.62	290.59	2846.85
heuristic 6	0.94	5.36	392.83	2938.45

of recall and less hops than adopting a combination of the two heuristics. However, this also results in higher system load, that is, more messages per node and more messages per query.

As we extended the time window, more queries captured in query history were used to calculate potentiality, which led to a much higher level of recall and less number of hops (see heuristics 4, 5 and 6). Similarly, the performance improvements are obtained at the cost of higher system load. This conclusion also applies to networks with different minimum degree (see Fig. 7).

We noticed that the highest possible level of recall (0.90) achieved by introducing randomness into reorganization and the SDS is lower than that (≥ 0.94 if the time window can be further extended) obtained by using both getGreatestPP_ON and getHighestPotentiality (see heuristics 1, 2 and 6) with less number of hops incurred.

4.3.3 Effect of Data Characterisitics

If we could know (or predict by some means) the distribution of the resource replication and queries in a P2P network beforehand, we would be able to determine whether topology reorganization should be applied to the network and how much improvements we can expect by investigating the effect of data (e.g. resource instances and queries) characterisitics on topology reorganization.

Table 6 shows the performance improvements that topology reorganization brings to networks characterized by a combination of different distributions of resource instances and queries[19]. We discovered that, without reorganization, networks in which resource replication follows discrete uniform distribution deliver the lowest level of recall and the lowest system load among others. Applying topology reorganization to such networks can make the greatest improvement to the recall level and also lead to the least number of hops. Like in all the other networks, this performance enhancement is obtained at the cost of increased load on nodes.

Meanwhile, Gaussian distribution for resource replication in a network without reorganization yields the highest level of recall and the highest system load regardless of query distribution. The use of topology reorganization only makes the least improvement to the recall level and results in the most number of hops. The system load incurred after using topology reorganization still ranks at the top.

[19] The *number* in uniform (number) represents the replication ratio.

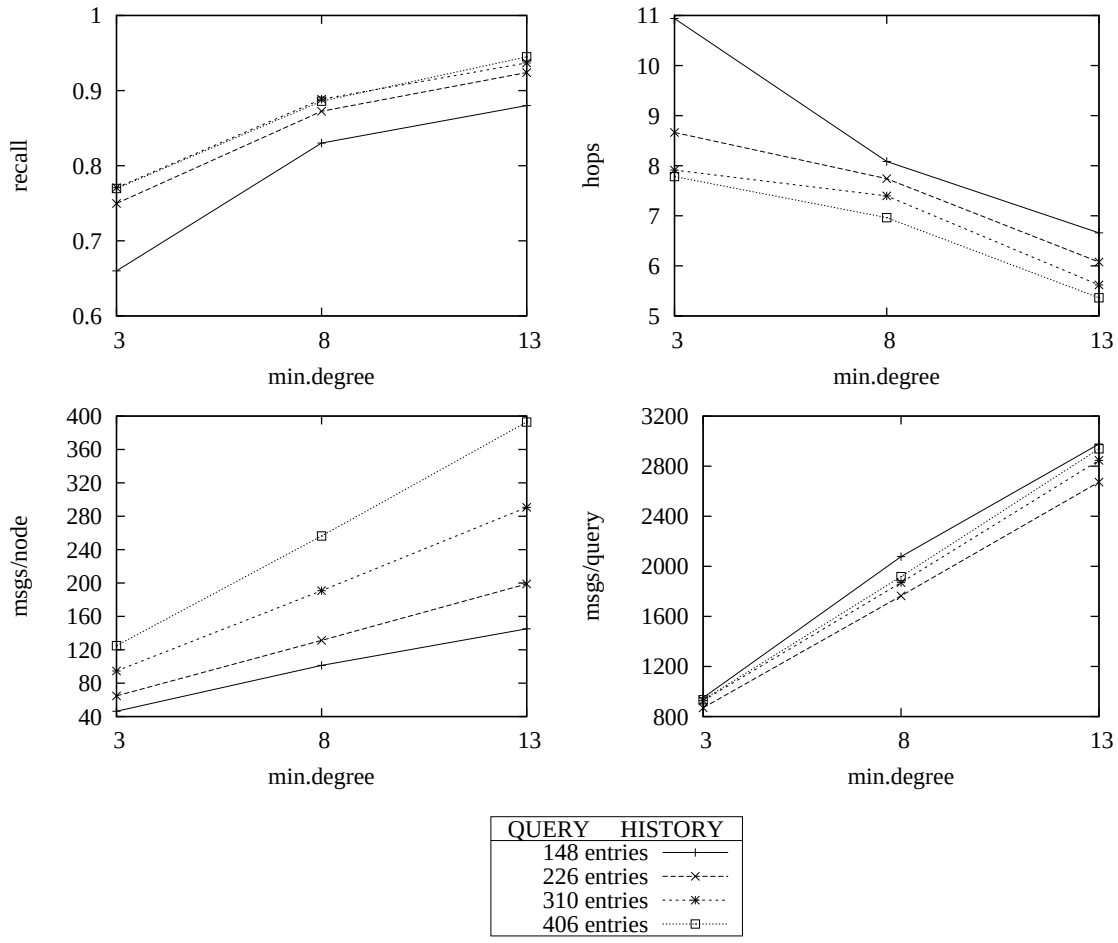

Figure 7: Performance achieved when increasing the length of time window

Table 6: Performance improvements introduced by topology reorganization

without topology reorganization					
resource replication	*query distribution*	*recall*	*hops*	*msgs/query*	*msgs/node*
Gaussian	uniform	0.65	3.45	569.28	18.74
	Zipf	0.64	3.43	585.95	28.55
Zipf	uniform	0.40	3.92	161.89	5.33
	Zipf	0.37	3.99	185.20	9.03
uniform (0.005)	uniform	0.36	3.63	107.14	3.53
	Zipf	0.39	3.89	138.39	6.74
uniform (0.01)	uniform	0.32	3.82	81.26	2.68
	Zipf	0.32	3.68	75.95	3.70
uniform (0.05)	uniform	0.29	3.80	114.07	3.76
	Zipf	0.27	3.82	106.53	5.19
with topology reorganization					
resource replication	*query distribution*	*recall*	*hops*	*msgs/query*	*msgs/node*
Gaussian	uniform	0.72	9.91	1430.17	47.09
	Zipf	0.81	12.59	1289.75	62.85
Zipf	uniform	0.81	8.22	791.94	26.08
	Zipf	0.88	6.66	2976.59	145.06
uniform (0.005)	uniform	0.88	5.88	323.10	10.64
	Zipf	0.88	6.08	348.32	16.97
uniform (0.01)	uniform	0.93	4.34	273.55	9.01
	Zipf	0.94	4.31	289.22	14.09
uniform (0.05)	uniform	0.97	4.21	1275.00	41.98
	Zipf	0.97	4.20	1182.57	57.63

4.3.4 Caching of Query Results

As reported in [23] and [20], scalable P2P search can be achieved by caching query results at each node along the reverse path that the query messages have traversed (that is, creating shortcuts to peers that answered the previous queries passed successfully), we conducted experiments to explore the impact of such a method on the performance enhancement of the SDS and presented the result in Fig. 8.

We observed that, apart from a marginal reduction in the number of hops and increase in the recall, caching query results does not yield any significant enhancement to the performance of the SDS but can incur three times as many query messages per query and query messages per node, thus giving the system a high load. We analyzed the simulation traces and observed that caching query results at each of the nodes along the reverse path to the query originator produces an excessive number of messages.

5 Conclusions and Future Work

In view of the fact that most ongoing efforts to implement large scale triple stores involve supporting more triples in any single triple store, we intended to provide scalable techniques to eliminate resource and performance bottlenecks in such centralized systems. Our work has achieved improved scalability in large scale triple stores by applying the following techniques: a fully decentralized P2P architecture in which a large scale RDF repository is split into multiple RDF data files maintained by individual nodes, ontology-based matching for identifying desired resources, a semantics-directed search protocol for efficiently routing query messages, and topology reorganization for further enhancing system performance. Simulations demonstrated the superior ability of our work over multiple random walk, constrained flooding, and interest-based locality to deliver desired scalability whilst incurring the least system load.

Due to the large scale of the triple stores in question, we have yet to demonstrate the efficiency of proposed techniques in any real deployment of RDF triple stores. In future work, we need to implement these techniques in related projects. We are in particular keen to confirm that breaking down a large scale RDF repository based on the subject type of triples provides the right level of granularity for organizing and describing RDF data files, and that taking advantage of semantics awareness can make existing triple stores more scalable in practice.

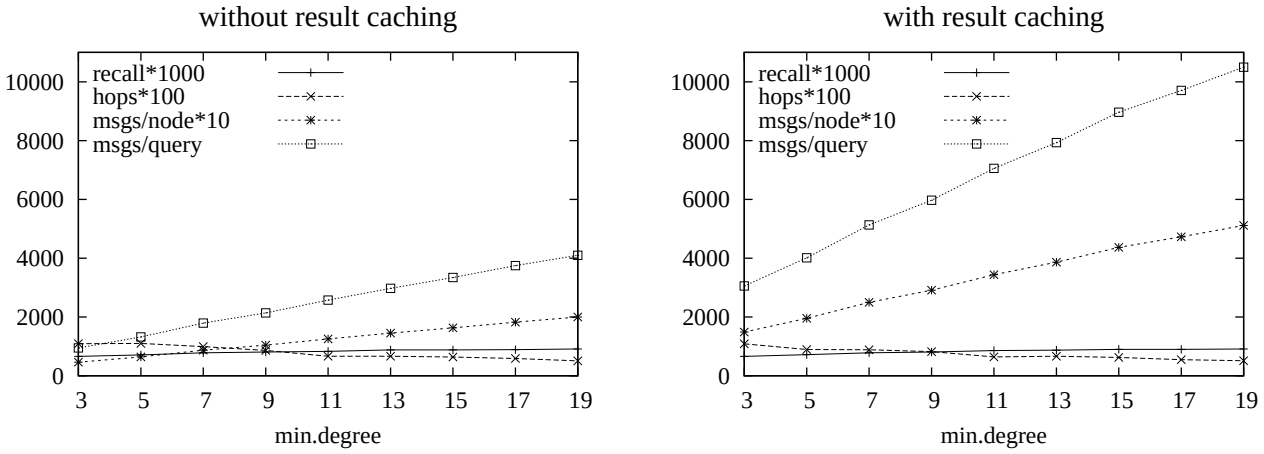

Figure 8: Performance comparison between with and without result caching

References

[1] J. Broekstra, A. Kampman, F. van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In *Proc. of the 1st International Semantic Web Conference*, Sardinia, Italy, 2002, pp.54–68.

[2] S. Harris, N. Gibbins. 3store: Efficient bulk RDF storage. In *Proc. of the 1st International Workshop on Practical and Scalable Semantic Web Systems*, Sanibel Island, Florida, USA, 2003, pp.1–15.

[3] B. McBride. Jena: A Semantic Web toolkit. *IEEE Internet Computing*, 6(6): 55–59, 2002.

[4] M. Arenas, V. Kantere, A. Kementsietsidis *et al.* The Hyperion project: From data integration to data coordination. *ACM SIGMOD Record*, 32(3): 53–58, September 2003.

[5] K. Aberer, P. Cudré-Mauroux, M. Hauswirth *et al.* GridVine: Building Internet-scale semantic overlay networks. In *Proc. of the 3rd International Semantic Web Conference*, Hiroshima, Japan, 2004, pp.107–121.

[6] W. Nejdl, B. Wolf, C. Qu *et al.* EDUTELLA: A P2P networking infrastructure based on RDF. In *Proc. of the 11th International Conference on World Wide Web*, Honolulu, Hawaii, USA, 2002, pp.604–615.

[7] W. Nejdl, M. Wolpers, W. Siberski *et al.* Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *Proc. of the 12th International Conference on World Wide Web*, Budapest, Hungary, May 2003, pp.536–543.

[8] G. Kokkinidis, L. Sidirourgos, V. Christophides. Query processing in RDF/S-based P2P

database systems. *Semantic Web and Peer-to-Peer*. S. Staab, H. Stuckenschmidt (eds.), Springer-Verlag, 2006, pp.59–81.

[9] A. Halevy, Z. Ives, P. Mork *et al.* Piazza: Data management infrastructure for Semantic Web applications. In *Proc. of the 12th International Conference on World Wide Web*, Budapest, Hungary, May 2003, pp.556–567.

[10] M. Cai, M. Frank. RDFPeers: A scalable distributed RDF repository based on a structured peer-to-peer network. In *Proc. of the 13th International Conference on World Wide Web*, New York, NY, USA, 2004, pp.650–657.

[11] H. Stuckenschmidt, R. Vdovjak, J. Broekstra *et al.* Towards distributed processing of RDF path queries. *International Journal of Web Engineering and Technology*, 2(2/3): 207–230, 2005.

[12] D. Clark. Face-to-face with peer-to-peer networking. *Computer*, 34(1): 18–21, 2001.

[13] T. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2): 199-220, June 1993.

[14] P. Mitra, N. Noy, A. Jaiswal. OMEN: A probabilistic ontology mapping tool. In *Proc. of the 4th International Semantic Web Conference*, Galway, Ireland, November 2005, pp.537–547.

[15] R. Pan, Z. Ding, Y. Yu *et al.* A Bayesian network approach to ontology mapping. In *Proc. of the 4th International Semantic Web Conference*, Galway, Ireland, 2005, pp.563–577.

[16] L. Gong. JXTA: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, May 2001.

[17] H. Balakrishnan, M. Kaashoek, D. Karger *et al.* Looking up data in P2P systems. *Communications of the ACM*, 46(2): 43–48, February 2003.

[18] Q. Lv, P. Cao, E. Cohen *et al.* Search and replication in unstructured peer-to-peer systems. In *Proc. of the 16th International Conference on supercomputing*, New York, NY, USA, 2002, pp. 84–95.

[19] I. Stoica, R. Morris, D. Karger *et al.* Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. of the ACM SIGCOMM Conference 2001*, San Diego, California, USA, 2001, pp.149–160.

[20] Y. Chawathe, S. Ratnasamy, L. Breslau *et al.* Making Gnutella-like P2P systems scalable. In *Proc. of the ACM SIGCOMM Conference 2003*, Karlsruhe, Germany, August 2003, pp.407–418.

[21] P. Haase, R. Siebes, F. van Harmelen. Peer selection in peer-to-peer networks with semantic topologies. In *Proc. of the 1st International IFIP Conference on Semantics of a Networked World: ICSNW 2004*, Paris, France, 2004, pp.108–125.

[22] P. Haase, J. Broekstra, M. Ehrig *et al.* Bibster - A semantics-based bibliographic peer-to-peer system. In *Proc. of the 3rd International Semantic Web Conference*, Hiroshima, Japan, November 2004, pp.122–136.

[23] D. Menascé. Scalable P2P search. *IEEE Internet Computing*, 7(2): 83–87, March/April 2003.

[24] K. Sripanidkulchai, B. Maggs, H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proc. of IEEE INFOCOM*, San Francisco, CA, USA, 2003.

[25] C. Sayers, K. Wilkinson. A pragmatic approach to storing and distributing RDF in context using Snippets. *Technical report HPL-2003-231, Enterprise Systems and Data Management Laboratory*, HP Laboratories Palo Alto, USA, November 2003.

[26] J. Zhou. DDLS: Extending open hypermedia systems into peer-to-peer environments. *Ph.D. thesis*. University of Southampton, UK, 2004.

[27] D. Abadi, A. Marcus, S. Madden, K. Hollenbach. Scalable Semantic Web data management using vertical partitioning. In *Proc. of the 33rd International Conference on Very large data bases*, Vienna, Austria, 2007, pp.411–422.

[28] S. Ratnasamy, P. Francis, M. Handley *et al.* A scalable content-addressable network. In *Proc. of the ACM SIGCOMM Conference 2001*, San Diego, California, USA, 2001, pp.161–172.

[29] A. Rowstron, P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, 2001, pp.329–350.

[30] J. Winick, S. Jamin. Inet-3.0: Internet topology generator. *Technical Report CSE-TR-456-02*, University of Michigan, 2002.

[31] J. Zhou, W. Hall, D. De Roure *et al.* Supporting ad-hoc resource sharing on the Web: A peer-to-peer approach to hypermedia link services. *ACM Transactions on Internet Technology*, 7(2), May 2007.