Efficient approximate SPARQL querying of Web of Linked Data

B.R.Kuldeep Reddy and P.Sreenivasa Kumar

Indian Institute of Technology Madras,
Chennai, India
{brkreddy,psk}@cse.iitm.ac.in

Abstract. The web of linked data represents a globally distributed dataspace which can be queried using the SPARQL query language. However, with the growth in size and complexity of the web of linked data, it becomes impractical for the user to know enough about its structure and semantics for the user queries to produce enough answers. This problem is addressed in the paper by making use of ontologies available on the web of linked data to produce approximate results. The existing approach, which generates multiple relaxed queries and executes them sequentially one by one, is improved by integrating the approximation steps with the query execution itself. Thus, by performing query relaxation on-the-fly at runtime, the shared data between relaxed queries are not fetched repeatedly, resulting in significant performance benefits. Further opportunities for optimization during query execution are identified and are used to prune relaxation steps which do not produce results. The implementation of our approach demonstrates its efficacy.

1 Introduction

The traditional World Wide Web has allowed sharing of documents among users on a global scale. The documents are generally represented in HTML, XML formats and are accessed using URL and HTTP protocols creating a global information space. However, in the recent years the web has evolved towards a web of data [1] as the conventional web's data representation sacrifices much of its structure and semantics [2] and the links between documents are not expressive enough to establish the relationship between them. This has lead to the emergence of the global data space known as Linked Data[2].

Linked data basically interconnects pieces of data from different sources utilizing the existing web infrastructure. The data published is machine readable that means it is explicitly defined. Instead of using HTML, linked data uses RDF format to represent data. The connection between data is made by typed statements in RDF which clearly defines the relationship between them resulting in a web of data.

Berners-Lee outlined a set of Linked Data Principles for publishing data on the Web [3] in a way that all published data becomes part of a single global data space:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards(RDF, SPARQL)
4. Include links to other URIs, so that they can discover more things

The RDF model describes data in the form of subject, predicate and object triples. The subject and object of a triple can be both URIs that each identify an entity, or a URI and a string value respectively. The predicate denotes the relationship between the subject and object, and is also represented by a URI. SPARQL is the query language proposed by W3C recommendation to query RDF data [4]. A SPARQL query basically consists of a set of triple patterns. It can have variables in the subject,object or predicate positions in each of the triple pattern. The solution consists of binding these variables to entities which are related with each other in the RDF model according to the query structure.

There have been a number of approaches proposed to query the web of linked data. One direction has been to crawl the web by following RDF links and build an index of discovered data. The queries are then executed against these indexes. This approach is followed by Sindice[5], Swoogle[6]. Another approach has been to follow the federated query processing concepts [7], as in DARQ[8], which decomposes a SPARQL query in subqueries, forwards these subqueries to multiple, distributed query services, and, finally, integrates the results of the subqueries. Another execution approach for evaluating SPARQL queries on linked data is proposed in [9]. It is basically a run-time approach which executes the query by asynchronously traversing RDF links to discover data sources at run-time. SPARQL query execution takes place by iteratively dereferencing URIs to fetch their RDF descriptions from the web and building solutions from the retrieved data. The SPARQL query execution according to [9] is explained with an example below.

```
SELECT ?prof ?publ WHERE
{
 <http://site//univ> univ:hasPublications ?publ
      ?publ  univ:authoredBy  ?prof
      ?prof  rdf:type  Professor
}
```

Fig. 1. Example SPARQL query

Example. The SPARQL query shown in Figure 1 searches for Professors employed by the university who have authored a publication. The query execution begins by fetching the RDF description of the university by dereferencing its URI. The fetched RDF description is then parsed to gather a list of all of its pub-

lications. Parsing is done by looking for triples that match the first pattern in the query. The object URIs in the matched triples form the list of publications in the university. Lets say `<http://site1/publ1.rdf>`, `<http://site2/publ2.rdf>`, `<http://publ3/Mary.rdf>` were found to be the papers. The query execution proceeds by fetching the RDF descriptions corresponding to the three publications. Lets say first publ1's graph is retrieved. It is parsed to check for triples matching the second query pattern and it is found that publ1 was authored by John `<http://site4/John.rdf>`. John's details are again fetched and the third triple pattern in the query is searched in the graph to see whether he is of type Professor and if he is, the result of query is formed and displayed as output. Publ1's and Publ2's graphs and their author details would also be retrieved and the query execution proceeded in a way similar to Publ1's.

Consider the situation where the retrieved list publications authored by the professors may not meet the requirements of the user in which case query conditions can be relaxed to produce more results. For example, instead of looking for only Professors, the query can be generalized by searching for all types of people including lectures,graduate students etc. The algorithm presented in [10] generates relaxed SPARQL queries and executes them sequentially one after another to generate approximate answers. The algorithm was designed to work on centralized RDF repositories and the approach is extended to the web of linked data in this paper. The relaxed SPARQL queries formed share many query conditions in common, which are not utilized to optimize the queries. Especially in a distributed environment, like the web of linked data, avoiding repeated fetching of data shared across the queries results in significant performance benefits.

```
SELECT ?prof ?publ WHERE
{
 <http://site//univ> univ:hasPublications ?publ
      ?publ  univ:authoredBy  ?prof
      ?prof  rdf:type  Faculty
}
```

```
SELECT ?prof ?publ WHERE
{
 <http://site//univ> univ:hasPublications ?publ
      ?publ  univ:authoredBy  ?prof
      ?prof  rdf:type  Person
}
```

Fig. 2. Relaxed SPARQL query

Example. Figure 2 gives the two similar queries formed after the query term Professor is replaced by Faculty and Person terms using RDFS ontology, which are then executed sequentially. However, the first two predicates are common between the two queries, therefore to achieve efficiency the information corresponding to them can be fetched once. Hence, instead of generating the two queries the execution of the original query can continue by dereferencing the URIs corresponding to Publ1 and its author and retrieving their RDF descriptions, and the check performed by the third predicate to see whether the author is a Professor or not is only replaced by Faculty and Person at the last step.

This allows the retrieval of the shared data only once instead of twice had the existing approach been followed.

The goal of this paper is to perform approximate SPARQL querying of the web of linked data. This paper extends the approach presented in [10] for relaxed querying of centralized RDF repositories to the context of web of linked data and along with the execution approach presented in [9] takes into account different namespaces being used. The idea of delaying query relaxation to run-time is introduced in order to optimize the query performance and the various other optimization opportunities present are also recognized and used.

2 Similarity Measures

In [10] the similarity measures were defined to allow ranked approximate answers. However the measures were designed for centralized RDF repositories and considered only one ontology. In the context of linked data, each user publishing data has the freedom to define his own ontology, but according to the principles of linked data it has to be mapped to existing ontologies, therefore we assume such mappings exist for the purposes of this paper.

A triple pattern can be replaced by terms in the ontology in a number of ways. Therefore, there is a need to attach a score to each relaxation which can be used to rank them to ensure the quality of results. The score given to each relaxation measures the similarity of it to the original triple pattern. Highest scoring relaxation are executed first followed by others in the decreasing order of the similarity score. For example, we would rank the relaxation from (?X,type,professor) to (?X,type,faculty) higher than (?X,type,professor) to (?X,type,person) as the former is more similar to the original triple pattern. A SPARQL query consists of a basic graph pattern which in turn consists of triple patterns. Therefore, the score associated with an answer to a SPARQL query is computed by aggregating the scores of relaxed triple patterns. Each triple pattern consists of a subject, predicate and object parts, and each of them can be potentially relaxed. Their aggregated score gives the score of the triple pattern.

Similarity between nodes In a triple pattern t_1, if the subject/object node belongs to class c_1 in the RDFS ontology and is relaxated to class c_2 using the ontology we use the idea of Least Common Ancestor to compute the similarity of the two triple patterns. The Least Common Ancestor denotes the depth of the common ancestor superclass of the two classes from the root in the RDFS ontology.

$$score(c_1, c_2) = \frac{2 * Depth(LCA(c_1, c_2))}{depth(c_1) + depth(c_2)}$$

Similarity between predicates In a triple pattern t_1, if the predicate belongs to class p_1 in the RDFS ontology and is relaxed to class p_2 using the ontology we use the idea of Least Common Ancestor to compute the similarity of the two triple patterns similar to that done for subject/object nodes. The Least Common Ancestor denotes the depth of the common ancestor superproperty of

the two classes from the root in the RDFS ontology.

$$score(p_1, p_2) = \frac{2 * Depth(LCA(p_1, p_2))}{depth(p_1) + depth(p_2)}$$

Similarity between triple patterns If the triple pattern t_1-$(s_1, p1, o1)$ is relaxed to t_2-(s_2, p_2, o_2) we aggregate the similarity scores of the triple pattern constituents to compute the overall similarity score of relaxed triple pattern.

$$similarity(t_1, t_2) = score(s_1, s_2) + score(p_1, p_2) + score(o_1, o_2)$$

Score of an answer The bindings of the relaxed SPARQL queries form the answers to the original SPARQL query. Since the original query is relaxed in a number of ways we need a measure to rank the relevant answers to ensure the quality of results. Thus, we define the score of each relevant answer as the similarity of its corresponding relaxed SPARQL query from which it is produced to the original SPARQL query. The similarity between the two queries is obtained by combining the similarities of the triple patterns in them. Suppose the answer A is obtained from query $Q^{'}(t_1^{'}, t_2^{'}, t_3^{'}...t_n^{'})$ which was formed after the original query $Q(t_1, t_2, t_3....t_n)$ was relaxed.

$$score(A) = \sum_{i=1}^{n} similarity(t_i, t_i^{'})$$

3 Query Processing Algorithms

[10] presents an approach to generate relaxed SPARQL queries from the original SPARQL query using RDFS ontology. It produces many relaxed versions and assigns scores to them based on the similarity to the original query. The relaxed queries are then executed one by one sequentially in the descending order of their scores to get ranked approximate answers. However, the SPARQL queries generated have many query conditions in common. Therefore, the sequential execution approach of all the queries involves needlessly fetching the same data repeatedly. In this section we present an optimized query processing algorithm where relaxed queries are generated and answered on-the-fly during query execution resulting in significant performance benefits.

Algorithm 1 describes the approach presented in [10] that can be extended to produce approximate answers in the web of linked data. Lines 2-7 denote the steps taken to generate multiple relaxed queries. The relaxation procedure is described as a graph, called a relaxation graph here. First the given query is put as a root in the relaxation graph. Then each triple is relaxed one-by-one and the new query produced as a result is inserted as a child node of the query node in the relaxation graph that led to it being produced. Each triple relaxation is accompanied by computing its relaxation score and this score is attached to its corresponding relaxed query. This process is repeated till all possible relaxed queries are generated. Lines 11-18 execute the relaxed queries produced earlier sequentially one by one. To generate ranked approximate results and ensure

the quality of answers the relaxed queries are executed in the descending order of their similarity scores with the original query. The relaxed query with the maximum score is executed first following which the next query to be executed is chosen with the highest score amongst its children and so on.

Algorithm 1: Existing Approach

Input : :Query Q
Output: :Approximate answers

1 $relaxationGraph = \phi$
2 Insert Q as root in $relaxationGraph$
3 **while** $Q \neq \phi$ **do**
4 **foreach** *Triple* t_i *in* Q **do**
5 Relax t_i to $t_i^{'}$
6 compute the score of approximation
7 Insert $Q_i^{'}$ as a succeeding node of Q in $relaxationGraph$
8 $Q \Leftarrow Q_{siblingNode}$ or $Q_{succeedingNode}$
9 $Result = \phi$
10 $Candidates = \phi$
11 Insert Q's succeeding nodes from $relaxationGraph$ into $Candidates$.
12 **while** $Candidates > 0$ **do**
13 Select Q_i with maximum score from $Candidates$
14 Insert Q_i succeeding nodes $relaxationGraph$ into $Candidates$
15 $R \Leftarrow Execute(Q_i)$
16 $Result = Result \cup R$
17 Add Q_i to processed
18 Remove Q_i from $Candidates$
19 Return $Result$

Figure 3 describes the execution of two queries of figure 2. The two queries are generated from the query of figure 1 as described by algorithm 1. The query in figure 1 finds the professors in the university who have authored a publication. To get approximate answers, the query is relaxed by producing two queries in which the query condition professor is replaced by faculty and persons. The left box in figure 4 shows the execution of left query in figure 2 and similarly for the box on the right. As we can see, many of the URIs dereferenced are the same in both the cases. For both of them, the query execution takes place by first dereferencing the university's URI to retrieve its RDF graph. Then the details of its publications publ1,publ2 and publ3 followed by its authors, John,Peter and Mary, are fetched. The existing approach repeats this process twice for each of the relaxed query when instead we can fetch the shared information once and then perform the relaxation. This motivates us to integrate the approximation process with the execution of the query and is described in algorithm 2.

Algorithm 2 describes the proposed approach in this paper for efficient approximate answering. Lines 3-16 repeat for each query predicate in the given

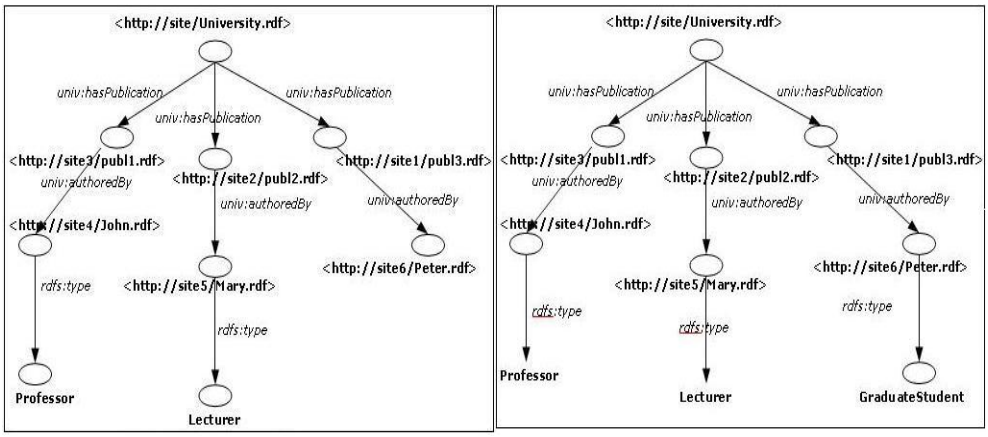

Fig. 3. Execution with existing approach

query. It begins with the seed, fetching its RDF graph. Then presence of the query predicate is checked for in the fetched RDF graph. If it is present the relaxation score for the predicate in the graph is given the maximum value of 1.0. Predicates belonging to different namespaces are assumed to be mapped in accordance with the linked data principles. Otherwise, using the metrics described in the earlier section the similarity score for each predicate in the RDF graph is computed. The predicates are then sorted in the descending order of their scores. The query execution proceeds by updating the seed with the object URIs of the predicates, which are then dereferenced to retrieve their graphs. Further similarities are computed and this process is repeated till a set of leaf values are produced. The path from the root to the leaf values in lines 17-19 along the relaxed predicates gives the approximate answers.

Figure 4 shows the query execution with the proposed approach for the query in figure 1. The query execution takes place by fetching the university's details, the details its publications and their authors just once. Once the publication's authors details have been retrieved the third predicate checking whether the person is of type professor can be relaxed to check for all people in the university like lecturers and graduate students. Thus in effect the relaxation mechanism has been delayed to be performed on-the-fly at run-time and by doing so the shared data is not fetched repeatedly which results in significant performance benefits.

4 Optimizations

The query processing described in the last section works by relaxing the query on-the-fly during query execution. This approach serves well to optimize the query but there are further opportunities that arise during query execution that can be exploited to optimize the query. To do so the vocabulary(RDFS/OWL) describing the resources which gives the domains and ranges of various predicates

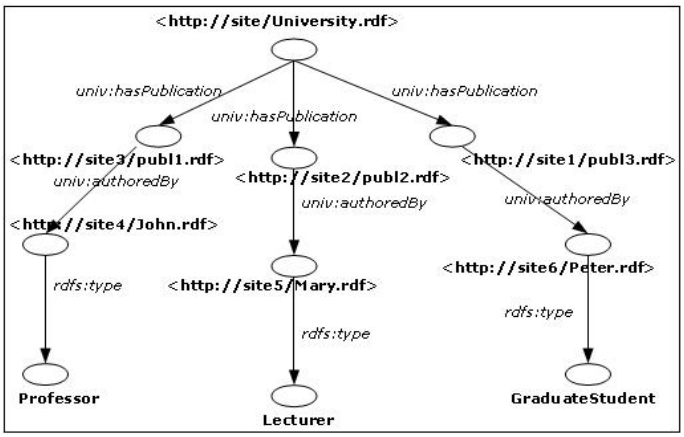

Fig. 4. Execution with proposed approach

as well as the subclass/superclass hierarchy details of all classes is considered. There are two cases that arise.

Case1: If a predicate p is replaced by $p^{'}$ with the subsequent predicate q, and that $range(p^{'}) \cap domain(q)$ is NULL the current relaxation of p is pruned as it will not produce results. There may be a situation when the subsequent predicate q is relaxed to $q^{'}$ and $range(p^{'}) \cap domain(q^{'})$ is not NULL in which case some results are missed. Therefore, a minimum threshold for the score of relaxation is maintained, and if the intersection of $range(p^{'}) \cap domain(q)$ is NULL the relaxation is pruned only if the score is below the threshold.

Case2: If an object o is replaced by $o^{'}$ with the subsequent predicate q, and that $o^{'} \cap domain(q)$ is NULL the current relaxation can be pruned as it will not produce results. There may be a situation when the subsequent predicate q is relaxed to $q^{'}$ and $o^{'} \cap domain(q^{'})$ is not NULL in which some results are missed. Therefore, a minimum threshold is maintained for the score of relaxation, and if the intersection of $o^{'} \cap domain(q)$ is NULL the relaxation is pruned only if the score is below the threshold.

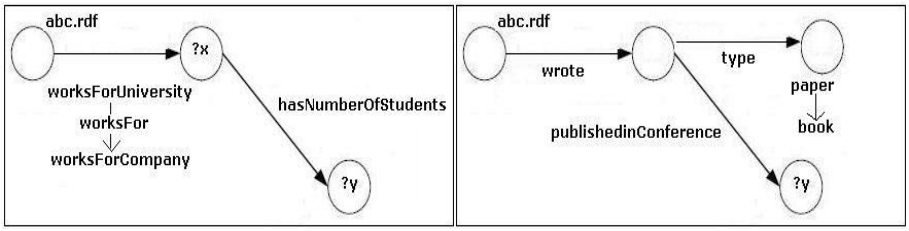

Fig. 5. Examples

Algorithm 2: Proposed Approach

Input : :Query Q
Output: :Approximate answers

let γ be the threshold
seed = intial set of uris
foreach $queryPredicate_k$ *in* Q **do**
  while *seed* $\neq \phi$**do**
    foreach $seed_i$ **do**
      Dereference $seed_i$ *and retrieve its RDF graph* R
      Remove $seed_i$ *from seed*
      foreach *predicate* p_j *in* R **do**
        if p_j *matches the corresponding query predicate* $queryPredicate_k$ **then**
          $relaxScore(p_j) = 1$
          if $p_{j_{object}} isbound$ **then**
            compute $relaxScore(p_{j_{object}})$ *with* $queryPredicate_{k_{object}}$
          else
            compute the $relaxScore(p_j)$ *with* $queryPredicate_k$
    Sort all p_j *in the descending order of their relaxScores.*
    foreach p_j **do**
      if $relaxScore(p_j) > \gamma$ **then**
        if $p_{j_{object}} isnotbound$ **then**
          $seed \Leftarrow seed \cup p_{j_{object}}$
  foreach $seed_i$ *in seed* **do**
    Retrieve the path p from $seed_i$ to root
    Return p as the approximate answer

Figures 5 illustrates the two cases. The figure on the left shows the query during whose execution the predicate "worksForUniversity" is relaxed to "worksFor". If there is a predicate "worksForCompany" in the retrieved RDF graph of the entity and as it is a subproperty of "worksFor" the query condition is relaxed to "worksForCompany". But the domain of the predicate succeeding it, that is "hasNumberOfStudents", is the class of universities whereas the range of the predicate "worksinCompany" is the class of Companies whose intersection is NULL. Thus this relaxation is pruned. But there is a possibility that the relaxation of the next predicate is from "hasNumberofStudents" to "hasnumberofEmployees". In which case the domain of the new relaxed predicate is the class of companies whose intersection with the range of earlier relaxed predicate is again the class of companies. Hence, if the first relaxation had not been discarded results could have been produced. To handle this situation, the score of relaxation is taken into account. If the score is above a certain predefined threshold, the relaxation is allowed and the query execution proceeds as usual.

The figure on the right shows the query during whose execution the object node "paper" is relaxed to the class of "books". However, the next predicate "publishedinConference" has the class of papers as it domain. Hence, the relaxation to class of books produces a NULL set and can be pruned.

Algorithm 3: Optimizations

Input : :Query Q
Output: :Decision on whether to continue with current approximation

```
1  let t denote the triple being handled, which is approximated to t'
2  let q be the predicate succeeding t
3  let γ be the threshold on the score of approximation
4  if predicate p is relaxed to p' then
5      if range(p') ∩ domain(q) == NULL then
6          if score(t) < γ then
7              try different relaxation of p
8  if object node o is relaxed to o' then
9      if o' ∩ domain(q) == NULL then
10         if score(t) < γ then
11             try different relaxation of o
```

5 Experiments

The experiments were conducted on a Pentium 4 machine running windows XP with 1 GB main memory. All the programs were written in Java. The synthetic data used for the simulations was generated with the LUBM benchmark data generator [11]. The LUBM benchmark is basically an university ontology that describes all the constituents of a university like its faculty,courses,students etc. The synthetic data is represented as a web of linked data with 200,890 nodes denoting entities and 500,595 edges denoting the relationships between them. The efficacy of the proposed idea was demonstrated by executing a set of queries in figure 6 used in [10] on the simulated web of linked data of a university and comparing the results with the existing approach. Each of the query below can be relaxed in a number of ways and the existing approach generates relaxed queries and executes them sequentially one by one whereas in contrast the proposed approach integrates the process of relaxation with the query execution to produce approximate answers. The time taken to execute the query is proportional to the number of URIs resolved to fetch their RDF descriptions during the course of query execution. Therefore, this paper uses the reduction in the number of URIs fetched as a metric to judge the results as the web of linked data was simulated on a single machine.

Q_1:(?x,type,TeachingAssitant)(?x,teachingAssistantOf,http://www.Department0.University0/Course3)(?x,mastersDegreeFrom,http://www.Department0.University0.edu)
Q_2:(?x,teacherOf,?z)(?x,ub:worksFor,University0)(?x,type,AssistantProfessor)
Q_3:(?x,advisor,?y)(?y,type,AssistantProfessor)(?y,researchInterest',Research12')(?y,worksFor,http://www.University0.edu)
Q_4:(?x,advisor,?y)(?y,type,Professor)(?y,worksFor, http://www.University0.edu)
Q_5:(?x,type,JournalAticle)(?y,publicationAuthor,?x) (?y,type,Professor)

Fig. 6. Queries

Query 1 searches for the teaching assistants of a particular course who have a masters degree from a particular university. Approximate answers are generated by relaxing the constraints step by step on the teaching assistant that is the teaching assistant can handle any course and have a master's degree from any university. Query 2 searches for assistant professors who teach a graduate course. Approximate answers are produced by relaxing the conditions in steps to look for all faculty who teach any course. Query 3 looks for assistant professor advisors who have a particular research interest. The query is again relaxed in steps by searching for all the people in the university who have any research interest. Query 4 searches for advisors who are professors and work for a particular university. Approximate answers are produced by looking for advisors who can be any type of faculty and who work for any university. Query 5 searches for professors who have authored a journal article. Approximate answers are produced step by step by looking for all persons including graduate students who have authored any type of paper. Figure 7 shows the number of URIs of entities dereferenced with the existing and the proposed approaches. The query performance improves by 75% for query 1, 80% for query 2, 83% for query 3, 78% for query 4 and 67% for query 5.

6 Conclusions And Future Work

The paper presented an approach towards allowing approximate querying of the web of linked data. The proposed idea produces approximate answers by relaxing the query conditions on-the-fly during query execution using the ontologies available on the web of linked data, in contrast with existing approach which generates multiple relaxed queries and executes them sequentially. The advantage of proposed approach is that it is able to avoid fetching the shared data between the relaxed queries repeatedly, which results in significant performance benefits as shown in the experiments. Future work includes investigation of other schemes, like top-k systems, towards producing approximate answers.

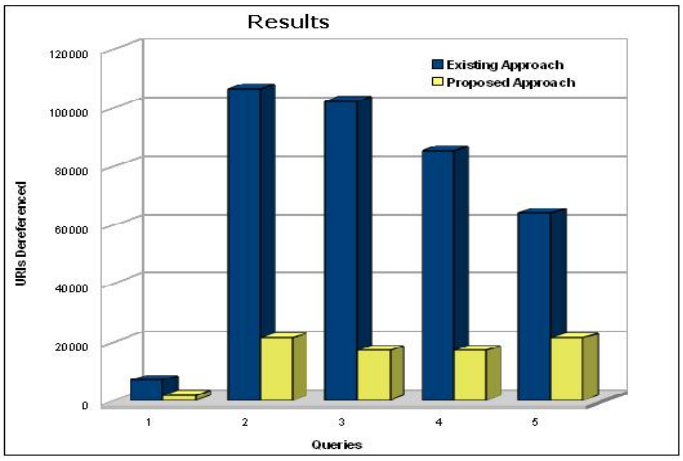

Fig. 7. Results

References

1. Franklin, M.: From databases to dataspaces: A new abstraction for information management. SIGMOD Record **34** (2005) 27–33
2. Bizer, C., Heath, T., Berners-Lee, T.: Linked data – the story so far. International Journal on Semantic Web and Information Systems **5**(3) (2009) 1–22
3. Berners-Lee, T.: Linked data - design issues. web page (2006)
4. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C recommendation, World Wide Web Consortium (2008)
5. Tummarello, G., Delbru, R., Oren, E.: Sindice.com: Weaving the open linked data. (2008) 552–565
6. Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V.C., Sachs, J.: Swoogle a semantic web search and metadata engine. In: Proc. 13th ACM Conf. on Information and Knowledge Management. (Nov. 2004)
7. Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv. **22**(3) (1990) 183–236
8. Quilitz, B., Leser, U.: Querying distributed rdf data sources with sparql. In Hauswirth, M., Koubarakis, M., Bechhofer, S., eds.: Proceedings of the 5th European Semantic Web Conference. LNCS, Berlin, Heidelberg, Springer Verlag (June 2008)
9. Hartig, O., Bizer, C., Freytag, J.C.: Executing SPARQL queries over the web of linked data. In: ISWC 2009: Proceedings of the 8th International Semantic Web Conference, Chantilly, VA, USA. (2009) 293–309
10. Huang, H., Liu, C., Zhou, X.: Computing relaxed answers on rdf databases. In Bailey, J., Maier, D., Schewe, K.D., Thalheim, B., Wang, X.S., eds.: WISE. Volume 5175 of Lecture Notes in Computer Science., Springer (2008) 163–175
11. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for owl knowledge base systems. J. Web Sem. **3**(2-3) (2005) 158–182