Improving the Real-time Performance of Heterogeneous Extremely Large Datasets

Stasinos Konstantopoulos
NCSR “Demokritos”
Patr. Grigoriou & Neapoleos
15341 Agia Paraskevi, Greece
+302106503162
konstant@iit.demokritos.gr

Antonis Koukourikos
NCSR “Demokritos”
Patr. Grigoriou & Neapoleos
15341 Agia Paraskevi, Greece
+302106503162
kukurik@iit.demokritos.gr

Pythagoras Karampiperis
NCSR “Demokritos”
Patr. Grigoriou & Neapoleos
15341 Agia Paraskevi, Greece
+302106503162
pythk@iit.demokritos.gr

ABSTRACT
The POWDER protocol is a Semantic Web technology –and W3C Recommendation- that takes advantage of natural groupings of URIs, as identifiers as well as navigational paths, to annotate all the resources in a regular expression-delineated sub-space of the URI space. POWDER was designed as a mechanism for accreditation, trustmarking and resource discovery, emphasizing the publishing of attributed metadata by third parties and trusted authorities. However, its versatility allows the application of POWDER in different use cases such as repository compression. In this paper, we present the POWDER protocol, briefly discuss current implementations and use cases and present how POWDER can be implemented over existing well-tested and robust semantic storage systems. Furthermore, we discuss a novel solution for the scalable storing data summaries in the form of metadata for the purposes of source selection and source schema coordination in large-scale, heterogeneous federations of semantic querying end-points. Our solution takes advantage of POWDER’s ability to exploit naming conventions and other natural groupings of URIs in order to compress instance-level metadata about the nodes of a data service federation, especially in situations where URI hashing cannot be used to efficiently resolve the sources that hold statements regarding a given URI resource.

Categories and Subject Descriptors
H.2.5 [**Heterogeneous Databases**]

General Terms
Algorithms, Experimentation.

Keywords
Metadata Publishing, Semantic Web, Triple Store Compression, Resource Discovery.

1. INTRODUCTION
The volume and quality of data made available freely on the Internet has significantly increased during the last years. This trend provided the opportunity for the Linked Data community to combine, cross-reference, and analyze unprecedented volumes of high-quality data and to build innovative applications. This effort has caused a tremendous network effect, adding value and creating new opportunities for everybody, including the original data providers.

But most of the low-hanging fruit has been picked and it is time to move on to the next step, combining, cross-indexing and, in general, making the best out of all public data, regardless of their schema, size, and update rate; accepting that some schemas might be better suited to a given dataset and application and that there is no consensus about a "universal" schema or vocabulary for any given application, let alone for the Semantic Web and related initiatives such as the LOD cloud. In other words, we need infrastructure that besides being efficient, real-time responsive and scalable is also flexible and robust enough to allow data providers to publish in the manner and form that best suits their processes and purposes and data consumers to query in the manner and form that best suits theirs.

This will be a decisive factor in maintaining the momentum of the Linked Open Data movement by including in the cloud large, live, constantly updated datasets and streams that are published in formats that were not designed with linking across sources in mind. This will not only increase the value of all public data, but can also provide both the incentive and the opportunity to follow Semantic Web standards and Linked Data best practices for publishers that will not or cannot directly and immediately make this transition.

In order to achieve this ambitious vision and solve a difficult data management problem, the following key challenges should be addressed:

- Develop novel algorithms and methods for querying distributed triple stores that can overcome the problems stemming from heterogeneity and from the fact that the distribution of data over nodes is not determined by the needs of better load balancing and more efficient resource discovery, but by the providers of the data instead.
- Develop scalable and robust semantic indexing algorithms that can serve detailed and accurate data summaries and other data source annotations about extremely large datasets. Such annotations are crucial for distributed querying, as they support the decomposition of queries and the selection of the data sources which each query component will be directed to.

- Since it is, in the general case, not possible to align schemas and vocabularies so perfectly that there is no loss of information, investigate how to minimize losses and how to not accumulate them over successive schema translations.

To move towards these challenges, we propose to exploit the W3C POWDER protocol in order to maintain more detailed data summaries and data source annotations, and do at a larger scale than is possible today. In this paper, we focus on the implementation of a POWDER triple store over widely used semantic storage systems and examine the benefits that may be achieved with respect to the performance of querying mechanisms over such stores.

The paper is structured as follows: First, we briefly present the POWDER protocol and some of its current implementations (Section 2). We proceed to discuss issues related with the implementation of a POWDER triple store (Section 3) and the issues that must be addressed for using it in the context of large-scale, heterogeneous federations of semantic querying end-points (Section 4). Afterwards, we present the proposed overall architectural solution for such a federation (Section 5), and discuss experimental results measuring query performance over an experimental infrastructure (Section 6). Finally, we discuss our findings and the conclusions that can be offered.

2. THE POWDER PROTOCOL

2.1 THE POWDER W3C Recommendation

POWDER is a Semantic Web technology that takes advantage of natural groupings of URIs to annotate all the resources in a regular expression-delineated sub-space of the URI space. It can be summed up most succinctly as a solution to the rdf:aboutEach problem, as exposed by, among others, Gibbins and Shadbolt [2]. POWDER allows one or more properties and their values to be associated with an arbitrary number of subjects within a fully-defined semantic framework. The fundamental information unit in POWDER is the Description Resource (DR) [1]. For example, the following code example contains repetitive URLs.

```
@prefix space:
<http://purl.org/net/schemas/space/> .
@prefix foaf:
<http://xmlns.com/foaf/0.1/> .
<http://nasa.dataincubator.org/spacecraft/1969-
059A>
a space:Spacecraft;
foaf:name “Apollo 11 Command and Service Module
(CSM)”.
<http://nasa.dataincubator.org/spacecraft/1969-
059B>
a space:Spacecraft;
foaf:name “Apollo 11 SIVB” .
<http://nasa.dataincubator.org/spacecraft/1969-
099A>
a space:Spacecraft;
foaf:name “Apollo 12 Command and Service Module
(CSM)”.
<http://nasa.dataincubator.org/spacecraft/1969-
099B>
a space:Spacecraft;
foaf:name “Apollo 12 SIVB” .
```

Using POWDER, we can recast the code these repetitive URIs as shown below.

```
<dr>
  <iriset>
    <includehosts>
      nasa.dataincubator.org
    </includehosts>
    <includepathstartswith>
      Spacecraft
    </includepathstartswith>
  </iriset>
  <descriptorset>
    <typeof
src=”http://purl.org/net/schemas/space/Spacecraft”
/>
  </descriptorset>
</dr>
```

POWDER allows making explicit the intention behind URI structure, which goes beyond exhaustively annotating all resources in a domain as it also represents the knowledge that currently unknown and future resources within the domain will have the properties implied by their position in the URI structure. For example, POWDER allows us to express the knowledge that any resource, currently existing or added in the future, under http://red.com has value "red" for the ex:colour property.

The logical foundations of POWDER, as laid out in the Formal Semantics document of the recommendation [10], are based on an extension of RDF semantics that assigns the wdrs:matchesregex datatype property to resources, based on their URI’s matching the regular expression that fills the property. Under this extension, the semantics of POWDER documents can be formulated in OWL, allowing POWDER statements to be involved in further inference about the properties of given resources known by URI. The POWDER Recommendation safeguards its semantics against involving OWL inference results in the process of calculating the extension of the wdrs:matchesregex property. Hence, the POWDER extension can be implemented independently of any inference based on POWDER assertions.

POWDER offers the advantage that, unlike various ad-hoc formalisms and other standards for providing such information - like robots.txt files, RFC 5785 and the PICS protocol - it is designed with extensibility and interoperability in mind, having a semantics that is compatible with RDF and OWL tools and a mechanism for developing and for assigning formal semantics to, extensions to the basic POWDER constructs primarily targeting URLs and the HTTP Web. Furthermore, the protocol makes provisions for attributing POWDER documents with provenance data and validity period restrictions. These features allow for a variety of possible applications to be explored besides the fundamental functionality of annotating resources.. In general, the use cases for POWDER are centered on machine-readable trustmarks, disambiguation of subject matter, declarations of accessibility compliance, mobile-friendliness, online safety and so on. These all share the need to make statements about websites or parts of websites, which is a more generally recurring need. However, its versatility at mass-annotating resources has given rise to an unexpected, at the time of developing the protocol, application as a technology for repository compression. Empirical experimentation on the space compression and the computation efficiency of current POWDER implementations has been carried out in [11] and has shown POWDER to achieve moderate compression rates (at the order of 20%) at no computational expense.

2.2 POWDER Implementations

While POWDER can be processed entirely as XML, its strict definition allows its transformation into an OWL ontology via a two-stage process. First, POWDER reduces the syntactic sugar of the URI set into just two primitive properties: include-regex and excluderegex, producing a POWDER-BASE document. The second stage of the transformation brings us fully into semantic technologies. POWDER-BASE documents are transformed to the equivalent OWL/RDF knowledge bases, following the POWDER-S specification.

This rest of this section describes three alternative possible approaches to implementing an OWL/RDF POWDER Processor that can be integrated with Semantic Web tools like triple stores and reasoners.

Model-theoretic semantics: This is the most straightforward implementation of a POWDER processor. It relies on the transformation of POWDER documents into their POWDER-S equivalent, make all applicable assertions for all URI resource × RegEx pairs in the repository and then use OWL inference. An example of this approach is the SemPP reference implementation, produced by the POWDER Working Group.

Proof-theoretic semantics: The extension in RDF semantics defined by POWDER can be applied to the proof-theoretic RDF and RDFS semantics [5], rather than the aforementioned model-theoretic semantics. The extension amounts to augmenting the rules of RDFS inference in a way that triples are inferred from no premises other than the information in the DR, using a function that matches a resource's URI against the regular expressions in the DR's <iriset> to decide if the resource falls within the scope of the DR.

This extension was implemented by modifying the forward-chaining RDFS inference engine bundled with the Sesame 2 framework.

Query rewriting: We have explored the possibility of implementing POWDER in a way that does not require any other form of inference but can be integrated into a stack that performs any (or none) semantic inference. To achieve this, we have implemented a POWDER layer over a standard triple store. The added layer is a full-fledged POWDER processor, with no introduced dependencies on OWL or RDFS reasoning; i.e. the POWDER layer does not implement the wdrs:matchesregex extension but instead it directly annotates resources in the triple store based on the DRs upon which it operates. Additionally, this implementation adopted a layered inference approach, taking advantage of the SAILs architecture of the Sesame 2 framework. Sesame SAILs are 'stackable' components that infer implicit RDF triples from the (explicit or implicit) data they receive from the SAIL immediately below in the stack. The POWDER processor has been implemented as a SAIL stacked between the RDF store and the Semantic Inference SAIL [11].

All these implementations share the property of having very limited interaction with non-POWDER semantic data as POWDER results are combined with other data following a layered interface approach where POWDER statements contribute to semantic inference but not vice versa.

3. IMPLEMENTING A POWDER TRIPLE STORE

Although implementing POWDER as a distinct layer is sanctioned by the formal semantics of POWDER statements, this direct implementation of POWDER semantics forces a choice between efficiency and compression. On the one hand, forward-chaining approaches either generate all POWDER-inferred triples or push those to the semantic inference layer above or implement a combined RDFS/POWDER inference engine [12] that, again, generates all POWDER and RDFS-inferred triples. Either way, any benefits POWDER can have to repository compression are lost since the POWDER-inferred triples are made explicit. On the other hand, backward-chaining approaches such as query rewriting achieve compression at the expense of efficiency. Query rewriting operates by transforming triple patterns involving POWDER-inferred predicates into equivalent FILTER clauses which apply the regular expression that the resource must match for the predication to hold. As a result even the most restrictive POWDER statements can never be used to guard a query but can only be applied as tests over resources selected by previous triple patterns.

To fully realize the potential of POWDER we need to examine the assumptions made when designing triple stores and databases in general. The BTree is the data structure that underlies modern triple stores and relational databases, coupled with hash tables that index the externally visible values against the internal node IDs. These data structures support the following strategy when searching for a triple pat-tern (or, in general, a tuple):

- The bound values in the triple are looked up in the hash table and replaced with internal node IDs
- The node IDs in the tuple are looked up in the BTree
- The unbound variables are bound by the values retrieved from the BTree

This strategy is very efficient when the variables in query patterns are bound to full URIs, but breaks down when retrieving POWDER annotations, as explained in the below sections.

3.1 Retrieving Resources by Property

Let us assume that we are looking for all resources that have a given property. Current approaches maintain two or more BTrees for looking up the properties of a given resource (spo indexing) or the resources that have a given property (pos indexing) using the strategy outlined above. But no index of nodes that correspond to specific resources or values can efficiently retrieve all resources that have a given POWDER-inferred property, without testing every single URI in the knowledge base against the regular expression.

One possibility would be to maintain in the structure all (relevant to POWDER statements) regular expressions a node's URI matches. Besides the large space usage (effectively annulling any compression benefits from POWDER) this would require prohibitive population costs for adding a POWDER statement to a large repository, since every single URI would have to be visited and checked against the new statement's regular expression. It becomes obvious that to efficiently implement POWDER processing we need to also be able to efficiently retrieve all URIs that match a regular expression.

3.2 Retrieving Properties of Resources

Let us now assume that we are looking for the filler of a POWDER-assigned property when the subject is bound to a URI. We look up the URI in the hash table, but we find no explicit triples in the tree. A possible solution would be to have regular expressions themselves be the subject in the tree, and have the hash table return not only the internal ID of the URI, but also the internal IDs of all matching regular expressions. This is sub-optimal because it (a) dramatically increases population costs (every single entry in the hash table needs to be revisited and updated when a new POWDER statement is added) and (b) increases querying time by causing unnecessary regular expression tests.

4. SOURCE SELECTION AND DISTRIBUTED QUERYING

Repository federation and distributed querying are key technologies for efficient and scalable large-scale semantic repositories. The support for federated querying in most major repository implementations will be formalized in the upcoming SPARQL 1.1 specification. The new specification does not target federation that is transparent to the user, which is an open research question, but will include the SERVICE keyword through which query authors can specify which repositories to query about each triple pattern in the query.

Among the various approaches to optimizing distributed repositories, many target homogeneous databases where the same kind of information is stored across all nodes of the system using the same (or compatible) schema. Furthermore, such approaches typically require control over the way triples are hashed over the cluster, in order to optimize the distribution of triples so as inter-node communication is minimized [8]. Although this allows considerable optimizations, more dynamic solutions are needed as well as many real-world use cases: the ability to formally describe and take into account a prior and externally imposed data partitioning that the system does not control.

In the literature, several systems maintain indexes of the (kinds of) information stored at each source. Schema-level indexes are light-weight indexes of the properties and types (classes) that occur at each source [13]. Although efficient to maintain, such indexes are too coarse as they are missing instance-level information. So, for example, looking up repositories based on classes and properties from commonly used vocabularies (e.g. FOAF in general or AGROVOC in the agricultural domain) will return many false positives and thwart meaningful optimization.

Data summaries, on the other hand, combine schema and instance-level indexing to perform source selection. These stem from database concepts such as histograms, capturing statistics on selectivity and cardinality for the purpose of query optimization [9]. In order to index statistics about RDF triples, Q-Trees [4] combine the notion of histograms with that of R-Trees [3], multi-dimensional trees typically used to index spatial data. The core idea is that a hash function maps URIs and values to numerical "coordinates" that are used to find the node(s) in the Q-Tree corresponding to a (possibly only partially instantiated) triple pattern. Each such node aggregates statistics about all triples within its "bounding box"; this statistics is a ranked list of sources where data should be looked up.

The geometric metaphor employed by Q-Trees works well with heterogeneous data and is also robust to slightly outdated indexes [7]. However (a) does not allow for explicit declarations by repository maintainers regarding the kinds of data their repositories contain, but only relies on indexes incrementally built and maintained by successive queries; (b) fails take into account the semantic similarity between re-sources and relies instead on a purely syntactic mapping from URIs and value strings to numerical coordinates. In fact, the method is known to be very sensitive to the hashing function, with no universally good function found; (c) fails to intelligently break up buckets when capacity is exceeded, relying on simple area-minimizing strategies.

Revisiting the stored schema under which the original query was formed, query results from the end-points involved in a complex query are transformed back into that schema and are joined. As interdependences between the sub-queries can degenerate this process into a situation where massive data volumes need to be copied to and processed by the results collector, the proposed solution builds upon methods for approximately joining distributed query results and for distributed and approximate inference.

5. PROPOSED ARCHITECTURAL SOLUTION

The proposed solution is to offer SPARQL endpoints that federate SPARQL end-points over heterogeneous and diverse data sources, as depicted in Figure 1, which presents the proposed overall architecture.

The resource discovery and query decomposition component analyses SPARQL queries and uses information about the schema used and the instances stored in the various federated data stores in order to break up the original query into the optimal query fragments and decide where to forward each such fragment for execution. "Optimal" in this context involves a multitude of considerations, including minimizing the number of fragments (since joining the results carries considerable computational costs), schema proximity (minimizing the schema translation needed) and load balancing (preferring less used repositories).

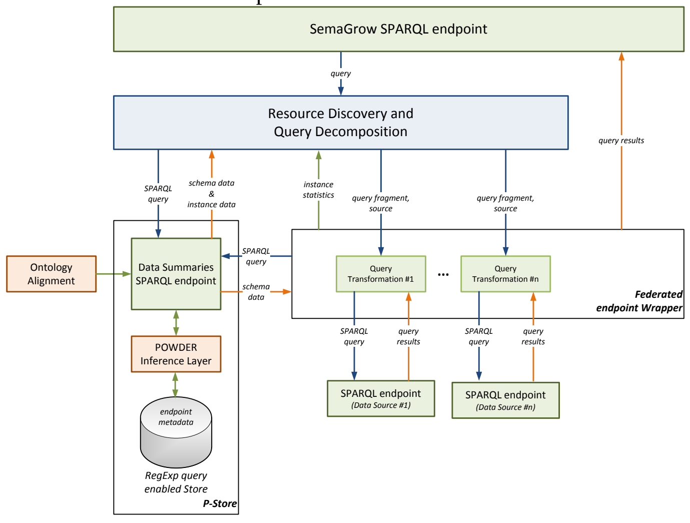

Figure 1. Architecture of the proposed solution

The resource discovery and query decomposition mechanism relies on using POWDER to mass-annotate large sub-spaces of the URI space, allowing the system to take advantage of naming convention regularities to compress its indexes. POWDER can concisely annotate regular-expression delineated URI spaces with a single statement, affording efficiency and scalability in storing and maintaining data summaries. The decomposition rules of the system are refined during system operation by comparing the results that are returned against the system's expectations. The system also foresees update and maintenance cycles, where new end-points are added to the federation or update the schema they employ or have accumulated considerable changes in the instances they hold. Schema metadata must be provided by the data provider when joining the federation, using authoring tools and tutorials produced by the project. Instance metadata may also be provided, but are also automatically maintained by the resource discovery module based on statistics extracted from query results. The rules and metadata used to achieve query transformation are derived by ontology alignment methods as well as human supervision.

6. EXPERIMENTS ON QUERY PERFORMANCE

6.1 Experimental Setup

To better understand the strengths and weaknesses of current POWDER implementations we have conducted experiments with a 500 Megatriple repository generated using the Lehigh University Benchmark (LUBM). Two variations of this dataset were prepared, one containing all RDF data and one where rdf:type information that can be inferred from the URI was removed; for example:

```
data:UnderGraduateStudent0          rdf:type
lubm:UnderGraduateStudent .
```

```
data:Department0.University0.edu/AssociatePr
ofessor0
```

```
rdf:type lubm: AssociateProfessor
```

To estimate the amount of triples saved, we used the generated data to instantiate two Sesame 2 Native Store instances, one full and one where POWDER-inferable triples were excluded. In the same fashion, we used the data to instantiate two Virtuoso instances, one full and one where POWDER inferable triples were excluded.

Queries on these repositories were executed via a POWDER implementation that applies backward-chaining inference to rewrite POWDER-inferable triple patterns in queries into equivalent FILTER clauses applying the regular expressions specified by the POWDER document that describes the data.

6.2 Experimental Results

The results over the Sesame Store instances for two queries on Table 1 are shown in Figures 2 and 3.

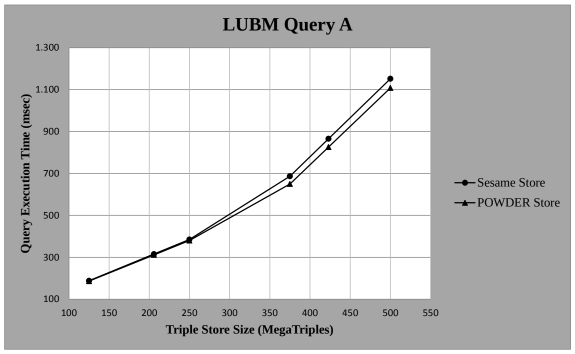

Figure 2. Performance for Query Example A (Native Sesame vs. POWDER on Sesame).

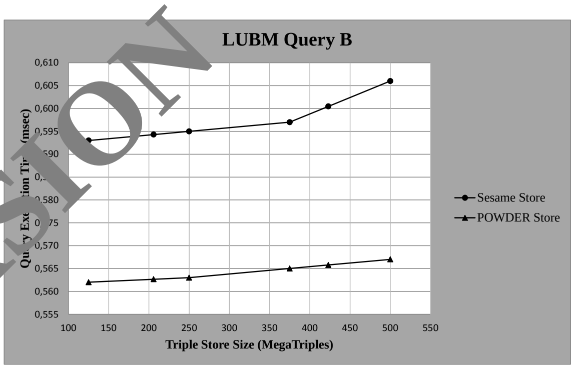

Figure 3. Performance for Query example B (Native Sesame vs. POWDER on Sesame).

Similarly, the results over the Virtuoso Store instances are depicted in Figures 4 and 5.

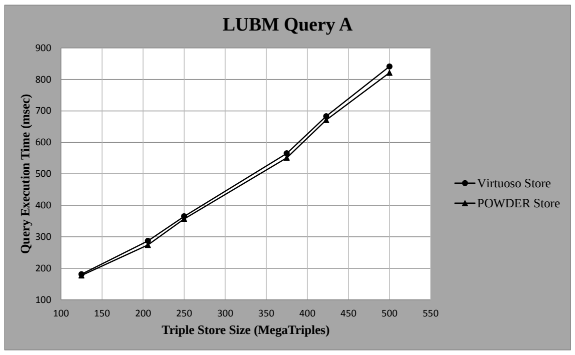

Figure 4. Performance for Query Example A (Native Virtuoso vs. POWDER on Virtuoso).

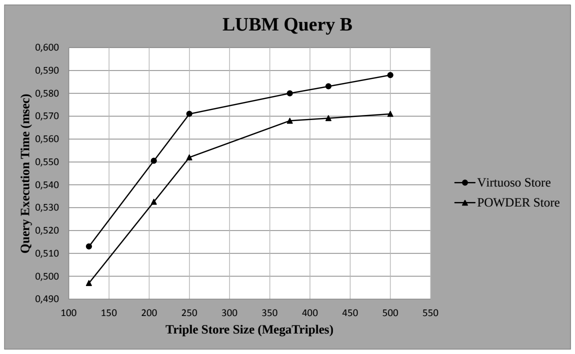

Figure 5. Performance for Query Example B (Native Virtuoso vs. POWDER on Virtuoso).

Query A is a high-throughput query that yields a number of tuples directly proportional to repository size; POWDER has no practical effect on these types of queries, especially since there is no non-POWDER pattern to guard the query. Query B is a query about a specific resource with a fixed throughput regardless of repository size; POWDER has a more profound effect since two out of the three triple patterns in the query (the two rdf:type patterns) do not result in a lookup in the index but are satisfied by matching the bindings of the ?X and ?Y variables against regular expressions; the existence of very restrictive guards (patterns with grounded resources) maximizes the benefit from POWDER.

```
SELECT ?X WHERE {
?X rdf:type
 lubm:UndergraduateStudent .
}
```

Figure 6. Query A

```
SELECT ?X ?Y WHERE {
  ?X rdf:type lubm:Student .
  ?Y rdf:type lubm:Course .

data:Department0.University0.edu/As
sociateProfessor0
    lubm:teacherOf ?Y .
  ?X lubm:takesCourse> ?Y .
   }
```

Figure 7. Query B

Naturally, since the computation needed to match the regular expression does not depend on the size of the repository but only on the number and complexity of POWDER descriptions, POWDER vs. vanilla performance over Query B diverges as the repository grows bigger. In other words, POWDER achieves scalability by loosening the association between the time it takes to annotate resources ?X and ?Y from the size of the repository.

Two factors should be noted when projecting the efficiency of a naïve POWDER store:

- The naïve POWDER store will dramatically improve performance on Query A-type queries.
- The LUBM schema is not well-suited for drastic compression under the currently implemented expressivity.

Especially with respect to this latter point, the naïve POWDER store will allow more extensive use of regular expressions by using them to compute property values instead of only permitting regular expression recognizers. So, for example, the LUBM URI convention would be better exploited inferring further properties besides the type, such as the university and department where they are employed, from resources such as:

data:Department0.University0.edu/AssociateProfessor0

Such a mapping formalism will be particularly useful in source selection and ontology alignment, and especially for instance-based meta-information. Consider, for example, the URIs:

`http://www.geonames.org/264371/athens.html`

`http://dbpedia.org/page/Athens,`

and the opportunity an extended POWDER implementation offers to succinctly de-scribe the first guess one would make to generate the latter from the former. Such a description can reduce great volumes of triples of mapping knowledge to a single statement and a handful of exceptions, such as:

`http://www.geonames.org/4180386/athens.html`

`http://dbpedia.org/page/Athens,_Georgia`

It is, thus, expected that the compression achieved can reach the order of 90% with querying time growing at a sub-linear rate against total data volume, resembling Figures 2 and 4 rather than Figures 3 and 5.

7. CONCLUSIONS AND FUTURE WORK

The present paper discusses (a) the main characteristics of the POWDER W3C Recommendation, its current application and the way that can be used for producing data summaries (b) how a scalable and robust semantic storage can be developed, using indexing algorithms that can take advantage of resource naming conventions and other natural groupings of URIs to compress data source annotations about extremely large datasets; and (c) how query decomposition, source selection, and distributed querying methods can be designed, that take advantage of such algorithms to implement a scalable and robust infrastructure for data service federation. Future work includes the development of a distributed infrastructure layer on top of existing data repositories and networks that will support the interoperable and transparent application of data-intensive techniques over heterogeneous data sources. This infrastructure will integrate:

- Novel indexing algorithms that support the efficient storage and retrieval of data summaries that concisely describe instance-level metadata about the different sources federated. These will support advanced source selection methods over distributed databases, affording scalability by disassociating or sub-linearly associating the time it takes to decide which repository to ask from the number and size of the repositories that the distributed system comprises.
- An extension of state-of-the-art query decomposition and rewriting methods that will enable complex queries in one schema to be broken down into sub-queries, each in a (possibly) different schema. In synergy with the distributed source selection mechanisms, this will allow queries in any schema to be executed at all and only those repositories that might hold relevant information..

The integration of state-of-the-art schema alignment methods under a novel architecture for the prior selection of the most appropriate method(s) for a given schema pair, the synthesis of multiple methods into a unified alignment, and the posterior evaluation of alignment quality. The results will be used at querying time to rewrite queries (or query fragments) from the query schema into the source schema and results from the source schema back into the query schema.

8. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n° 318497 (SemaGrow - Data Intensive Techniques to Boost the Real – Time Performance of Global Agricultural Data Infrastructures).

9. REFERENCES

[1] Archer, P., Smith, K., Perego, A, 2009. *Protocol for Web Description Resources (POWDER): Description Resources*, W3C Recommendation, 1 September 2009, http://www.w3.org/TR/powder-dr.

[2] Gibbins, N., Shadbolt, N., 2010. *Resource Description Framework (RDF)*, Encyclopedia of Library and Information Sciences, 3rd Edition, CRC Press.

[3] Guttman, A., 1984. *R-Trees, A dynamic index structure for spatial indexing*, in Proceedings of the Annual Meeting of ACM SIG on the Management of Data (SIGMOD '84), Boston, MA, USA.

[4] Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.-U., Umbrich, J., 2010. *Data summaries for on-demand queries over Linked Data*, in Proceedings of the 19th International World Wide Web Conference (WWW 2010), Raleigh, NC, USA.

[5] Hayes, P., 2004. *RDF Semantics*, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/rdf-mt

[6] Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.* 3 (Mar. 2003), 1289-1305.

[7] Hose, K., Klan, D., Sattler, K.-U., 2006. *Distributed data summaries for approximate query processing in PDMS*, in Proceedings of the 10th International Database Engineering and Applications Symposium (IDEAS '06), Delhi, India.

[8] Huang, J., D.J. Abadi, D.J., Ren, K., 2011. *Scalable SPARQL querying of large RDF graphs*, in Proceedings of the VLDB Endowment, Vol. 4(11)

[9] Ioannidis, Y, 2003. *The history of histograms (abridged)*, in Proceedings of the 29th International Conference on Very Large Databases (VLDB 2003), Berlin, Germany, Ten-Year Best Paper Award.

[10] Konstantopoulos, S., Archer, P., 2009. *Protocol for Web Description Resources (POWDER): Formal Semantics*, W3C Recommendation, 1 September 2009, http://www.w3.org/TR/powder-formal.

[11] Konstantopoulos, S., Archer, P., 2011. POWDER and the multi-million triple store, in Proceedings of the 3rd International Workshop on Semantic Web Information Management, ACM SIGMOD/PODS, Athens, Greece.

[12] Konstantopoulos, S, Archer, P., Karampiperis, P., Karkaletsis, V., 2012. *The POWDER protocol as infrastructure for serving and compressing semantic data*, International Journal of Metadata, Semantics, and Ontologies. Accepted to appear.

[13] Stuckenschmidt, H., Vdovjak, R., Houben, G.-J., Broekstra, J., 2004. Index structures and algorithms for querying distributed RDF repositories, in Proceedings of the 13th International World Wide Web Conference (WWW 2004), New York, USA (2004