Observing the Web of Data through Efficient Distributed SPARQL Queries

Xin Wang [a], Thanassis Tiropanis [a] and Wendy Hall [a]
[a] *Web and Internet Science Group, Electronics and Computer Science, University of Southampton, Southampton, UK, SO17 1BJ*
E-mail: {xw4g08,tt2,wh}@ecs.soton.ac.uk

Abstract. Dealing with heterogeneity is one of the key challenges of Big Data analytics. The emergence of Linked Data provides better interoperability and thus further enhances potential of Big Data analytics. The use of Linked Data for analytics raises performance challenges when considering the distribution of data sources and the performance of Linked Data stores in comparison to other storage technologies. This paper investigates the performance of distributed SPARQL approaches to gain an understanding of how well the distribution challenge can be addressed in such an environment. We describe the distributed SPARQL query infrastructure that has been deployed in the Southampton University Web Observatory (SUWO), to support analytics that requires prompt access to a large number of Linked Data datasets. This distributed SPARQL approach adopts dynamic optimisation techniques to take advantage of runtime statistics, and exploits parallelism. The infrastructure is evaluated on Twitter data hosted in the SUWO and popular queries on these data. It demonstrates that the infrastructure described in this paper has a performance advantage over other existing distributed SPARQL engines.

Keywords: Linked Data, Big Data, SPARQL, distributed query, Web Observatory

1. Introduction

In this era data are produced at an astonishing rate [16]. They grow beyond the ability of typical databases in terms of storage and analytics, and are usually referred to as the Big Data. Big Data analytics are reported to have brought significant amount of profits for many well known companies [18], among which are Amazon[1], Google[2], Twitter[3] etc. Meanwhile, Big Data analytics has also drawn much attention from academic [1].

Big Data have been characterised by a number of V's, including volume, velocity and variety [11]. In the meantime, effective Big Data analytics requires machine-understandable, interoperable data and coordinated datasets [13], which are contradictory to the properties of Big Data. The contradiction makes Linked Data [3], which was created to provide interoperable and machine-understandable semantics, a natural choice for improving Big Data analytics.

The representation of Linked Data is based on the Resource Description Framework (RDF) [9], which can be queried using the SPARQL Protocol and RDF Query Language (SPARQL) [21]. Both RDF and SPARQL are W3C standards and have been used to establish a Web-scale interoperable data space [10]. Many LD can be accessed via SPARQL endpoints, which are services conforming to SPARQL protocol and accept SPARQL queries over HTTP connections.

Querying LD, or distributed SPARQL query processing, confronts many challenges that exist in distributed database management systems (DBMSs). For example, a key challenge of distributed DBMS query processing is the latency of data transfer [19], and it becomes more challenging in the case of distributed SPARQL queries. In addition, advantages of dis-

[1] www.amazon.com
[2] www.google.com
[3] twitter.com

tributed DBMS, such as detailed statistics of datasets, are likely missing in many LD datasets. Such statistics are critical for query optimisation, and lacking them makes efficient distributed SPARQL queries more challenging.

The research regarding efficient LD queries has a good number of efforts [22,4,23,26,27,17], and will attract an increasing attention to provide effective support for analytics on LD in the Big Data context. In this paper we describe a distributed SPARQL query engine that adopts novel techniques to further improve LD query efficiency, and has been deployed in the Southampton University Web Science Observatory (SUWO) [5] at the University of Southampton. This engines exploits parallelism using a novel algorithm named Ψ and takes advantage of runtime statistics to optimise queries dynamically. It has been used in some initial analytical research over Twitter data within the SUWO. The Twitter data and relevant queries are used to evaluate the engine described in this paper. Its advantage in terms of efficiency is demonstrated by comparing to representatives of existing distributed SPARQL engines.

The remaining part of this paper is organised as follows. In section 2 existing approaches of distributed SPARQL are reviewed. After that in section 3 we provide details of the query infrastructure of the SUWO, including the Ψ algorithm and its integration with dynamic optimisation. To evaluate this approach we firstly describe the Twitter data and related queries in section 4. The evaluation settings and result are discussed in section 5. Future plans are given in section 6.

2. Related Work

Approaches for querying LD can be roughly divided into two categories. One is called link-traversal-based query execution [8], the other resembles query processing in distribute DBMSs. Link-traversal approaches assume that URIs are resolvable and RDF data can be retrieved by traversing URIs. A link-traversal approach resolves URIs in given queries to gather initial sets of RDF data. Queries are evaluated against these data and intermediate results are returned, which contain new URIs. The final results are achieved by applying the two steps iteratively. One representative link-traversal-based approach is [7], whose performance is improved in [6] by applying heuristics for query optimisation. Similar work have been described in [14,15], which propose a non-block iterator to improve query performance. Link-traversal approaches do not require extra knowledge about potentially relevant datasets. However the link-traverse-query loop can be time consuming.

The other category contains approaches having a similar behaviour of distribute DBMSs. These approaches require certain knowledge of target datasets, such as the addresses and statistical information. With such knowledge these DBMS-alike approaches have a better chance to optimise queries and thus gain better performance. In the context of SUWO, all datasets as well as certain statistics can be known in advance, and in this paper we focus on the DBMS-alike approaches and refer to them as distributed SPARQL engines.

In distributed DBMSs, effective query optimisation is closely related to the accuracy of dataset statistics. This also applies to distributed SPARQL optimisation. Accurate statistics allow thorough search for the optimal query execution plans (QEPs). Many distributed SPARQL engines follow this approach. For example, DARQ [22] maintains statistics containing the frequency of each predicate in an RDF dataset. The predicate frequency along with some selectivity[4] heuristics [25] are used in DARQ's cost model, which estimates the result size of a triple pattern as the product of the frequency of predicate and the selectivity of subject and object. Based the estimations DARQ adopts an (iterative) dynamic programming (IDP) [12] algorithm that exhaustively search for the optimal QEP.

SPLENDID [4] follows a similar strategy, however, exploits statistics provided by the Vocabulary of Interlinked Datasets (VoID) [2]. Beside predicate frequency, VoID further provides the number of distinct subjects and objects associated with each predicate. The extra statistics replace selectivity heuristics used in DARQ, and further improve the effectiveness of query optimisation, which is also an IDP algorithm, of SPLENDID.

Exhaustive search algorithms, such as IDP[5], usually have a high time complexity. To this end, LHD [27] takes advantage of certain patterns of SPARQL queries, and assists IDP with heuristics to improve optimisation efficiency without loosing the quality of QEP.

Despite of all the virtues of the combination of accurate statistics and exhaustive search algorithms, they

[4] The proportion of triples that meet a certain condition [24,20]

[5] Strictly speaking IDP approximates exhaustive search algorithms and allows trade-off between efficiency and effectiveness.

are costly in terms of maintenance and processing time. Furthermore, statistics that are accurate enough to guarantee the effectiveness of exhaustive search are unlikely to be available on a large scale. Alternative approaches, that require less statistics, or adopt less complex optimisation algorithms, have been investigated as well. FedX [23], for instance, completely utilises heuristics (and thus requires few statistics) for query optimisation and shows quite promising performance. DSP [26] adopts a minimum spanning tree (MST) algorithm and also has an advantage over DARQ in terms of efficiency.

Beside query optimisation, execution of QEPs are also crucial for efficient LD queries. It has been demonstrated that parallelisation significantly improves query efficiency [23]. There can be further performance gain if the concurrency of query execution is tuned with respect to the capacity of target endpoints [27].

3. Linked Data Query Infrastructure of the SUWO

In this section we describe an approach that further exploits parallelisation, and optimises queries during execution time using runtime statistics.

3.1. Ψ: Parallel Sub-Query Identification

SPARQL queries are composed by Basic Graph Patterns (BGPs), which are a set of conjunctive triple patterns. A BGP can be regarded as a connected graph whose nodes (or vertices) are subjects and objects and whose triple patterns are edges. During query execution, the number of bindings, or cardinality, of a concrete node is not changed by edges connecting to it. Given two edges (triple patterns) sharing a concrete node (e.g. {s p_1 ?x. s p_2 ?y}), they can be processed in parallel without introducing extra network traffic comparing to being executed sequentially. This observation also holds for shared variables whose cardinality does not change during execution.

For convenience we introduce the notion of *fixed cardinality node* and generalise the above observation as follows. A fixed cardinality node has the property that its number of bindings does not change more than a certain percentage, during the execution of all connected edges. If disconnected sub-graphs can be obtained by "removing" all fixed-cardinality nodes[6], these sub-graphs can be processed in parallel. For example, in the graph shown in Fig. 1, if both node B and C are fixed-cardinality nodes, then we have three independent sub-graphs $\{AC, AB\}, \{BC\}, \{CD, BD\}$. If only B has fixed cardinality, then the given graph cannot be further broken down. A more subtle case is that cardinality of both B and C are only changed by AB and AC respectively, while BC and BD have comparable cardinality at B, and BC and CD have comparable cardinality at C. That is, B and C are not fixed-cardinality nodes w.r.t all connecting edges, but they are w.r.t some edges. In this case $\{CB\}, \{CD, BD\}$ can still be executed in parallel, and we say this two components form a partial parallel group. However, identifying all partial parallel group can be costly and not worthy in practice.

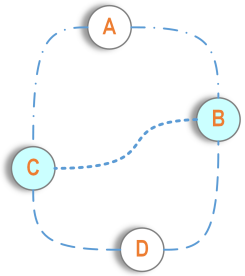

Fig. 1. If B and C are fixed-cardinality nodes, there are three independent components shown by three different types of dash lines.

Exploiting this idea, we propose the algorithm Ψ[7] (shown in algorithm 1) that efficiently identifies independent components. The algorithm firstly creates a sub-graph for each edge (the loop at line 1). Then it scans through all nodes and merges sub-graphs that share a none-fixed-cardinality node (the loop at line 4). Finally, all remaining sub-graphs can be processed in parallel using the dynamic optimisation described in the following section (section 3.2). The time complexity of the first loop is linear to the number of edges $|E|$. The merge operation in the second loop can be done in constant time by maintaining a hash table that maps a node to the set of its connected edges. Therefore, the complexity of the second loop is linear to the number

[6] Since removing a node produces broken edges that have only one node, a more precise description here would be "regarding all edges that connect to a fixed-cardinality node as disconnected at this node".

[7] Ψ=PSI=**P**arallel **S**ub-query **I**dentification

of vertices $|V|$. The complexity of Ψ (upper bound) is $O(\max(|E|, |V|))$.

Algorithm 1: Ψ(V,E)

input : A connected graph (V, E)
output: Independent sub-graphs

foreach $e \in E$ **do**
    `sub(`e`)` $\leftarrow e$;
end
foreach $v \in V \wedge \neg$ `fixCard(`v`)` **do**
    merge sub-graphs containing v;
end

Not only concrete nodes have fixed-cardinality. With certain knowledge of datasets, we can determine in advance whether the cardinality of a variable node will remain the same. In general, we need to know property schemas and relevant statistics. For example, in {?person foaf:firstName ?frstN. ?person foaf:familyName ?fmName}, since a dataset usually contains both the first name and family name of a person, the cardinality of *?person* probably keeps unchanged during execution. This conclusion is made based on knowing that that both properties have the same domain, have close numbers of distinct subjects, and are closely relevant. Heuristics can also be used to identify fixed-cardinality nodes. For example, a variable $?v$ is considered as a fixed-cardinality node if it satisfies:

$$\forall\, T_i, T_j \in conn(?v): \quad \frac{card(T_i, ?v)}{card(T_j, ?v)} < 110\% \quad (1)$$

or

$$|?v| < \min_{T \in conn(?v)} (card(T, ?v))/10 \quad (2)$$

where $conn(?v)$ gives all triple patterns that are connected to $?v$ and $|?v|$ is the number of existing bindings of v. The above two equations states that if the estimations of the cardinality of a variable $?v$ w.r.t all its connected triple patterns are close (i.e. for each triple pattern T_i having v, $card(T_i, ?v)$ is close to the same number) the number of bindings of $?v$ probably will not change. Also, if the number of existing bindings of $?v$ is very small, it probably will not change. The effectiveness of the above two heuristics depends on the accuracy of cardinality estimation. We enable these heuristics in the SUWO engine since runtime statistics are used (described as below).

3.2. Dynamic Optimisation Using Runtime Statistics

The effectiveness of query optimisation is closely related to the accuracy of cost estimation [19]. Many RDF datasets provide schemas and statistics using VoID. However, VoID can only provide limited statistics. Furthermore, since data publishers are independent, the accuracy of VoID statistics varies. To this end, the SUWO engine exploits statistics that become available during query execution, and interleaves query optimisation and execution. Once a triple pattern is executed, its number of bindings is used in the cost estimation of remaining triple patterns. This dynamic optimisation approach requires QEPs to be built in an incremental fashion. The estimated costs keep changing during query execution, which makes exhaustive search algorithms, such as IDP, less appropriate. Therefore, we choose a concise cost model and a MST-based algorithm (which is a greedy algorithm) in our approach.

Cost estimation

Since cost estimation is required at each time a triple pattern being executed, we use a concise cost model to save computing costs. We denote by $sel(t, n)$ the selectivity of a node (either a subject or an object) w.r.t a triple pattern t, and by $|p|$ the number of triples having p as predicate in relevant datasets. $sel(t, n)$ and $|p|$ are obtained from available statistics such as VoID files, or empirical values. For more details of these values please refer to SPLENDID [4] and LHD [27]. The cardinality $card(t)$ of a triple pattern $t : \{s\ p\ o\}$ is estimated as:

$$card(t) = |s| \cdot sel(t, s) \cdot |p| \cdot |o| \cdot sel(t, o) \quad (3)$$

where $|s| = 1/sel(t, s)$ if s is a variable having no bindings (i.e. an unbound variable does affect the cardinality), otherwise $|s|$ is the number of values of s. $|o|$ is determined in the same manner. During query execution, $|s|$ and $|o|$ are updated as new bindings becoming available.

The cost of a triple pattern depends on the execution method. If it is evaluated over relevant datasets without attaching existing bindings, the cost is estimated as $card(t) \cdot c$, where c is a constant. If existing bindings,

presumably from s, are attached, the cost is estimated as $|s| \cdot c + 1 \cdot sel(t, s) \cdot |p| \cdot |o| \cdot sel(t, o) \cdot c$.

Query optimisation

The MST-based optimisation algorithm is shown in algorithm 2. It picks the triple pattern having the lowest cost based on existing statistics, and builds QEPs incrementally. Each time the algorithm is called, it maintains a list of remaining edges ordered by their costs from low to high. If an edge has two possible costs, the smaller one is chosen. Then the algorithm returns and removes the minimum edge (i.e. an edge belongs to the MST), which is going to be executed. It also returns edges whose subjects and objects are all bound (i.e. edges that do not belong to the MST), which are used to prune existing bindings.

Algorithm 2: NextEdges(V,E)

input : A connected (sub-)graph (V, E)
output: $next$ a set of edges to be executed

1 $edges \leftarrow sort(E)$;
2 $next \leftarrow edges[0]$;
3 $next \leftarrow next \cup findBoundEdges(edges)$;
4 $E \leftarrow edges - next$;

The overview of query execution of the SUWO engine is shown as algorithm 3. Firstly a given query is broken into sub-graphs. For each sub-graph a new thread is created. At each step, minimum-cost triple patterns are selected (lines 6) and executed (line 7 to 8). Then cost of remaining edges (executed edges are removed at the end of algorithm $NextEdges(V, E)$) are updated using runtime statistics and $Execute(V, E)$, which is described in the following section, is called recursively. It should be noticed that a sub-graph can be further divided in future call of $Execute(V, E)$ w.r.t updated edge cost. Details of how

3.3. Parallel Query Plan Execution

The SUWO engine uses a parallel QEP execution manager that is similar to the one used by LHD. For integrity of this paper we briefly describe this parallel execution manager again.

For each dataset the QEP execution manager maintains a first-in-first-out task queue and a worker thread pool. The size of the thread pool is determined with respect to the capacity of an individual dataset. A QEP produced by query optimisation contains operations to

Algorithm 3: Execute(V,E)

input : A connected (sub-)graph (V, E)

1 **if** E *is empty* **then**
2 return;
3 **end**
4 $components \leftarrow \Psi(V, E)$;
5 **foreach** *sub-graph* $(V', E') \in components$ *create a new thread* **do**
6 $next \leftarrow NextEdges(V', E')$;
7 evaluate $next[0]$;
8 use remaining edges of $next$ to prune bindings;
9 update costs of edges in E';
10 $Execute(V', E')$;
11 **end**

execute triple patterns against relevant datasets. Once an operation is executed, it does not directly contact remote datasets, but raises a task to each task queue of relevant datasets. The task is executed immediately if there is an idle worker thread, waits in the queue otherwise. Once a worker thread finishes a task, it checks the task queue. If the task queue is not empty, the worker thread picks the task comes first. Otherwise the thread halt to save system resource.

4. Twitter Datasets and Analytic Queries

In this section we describe the datasets and queries that were used to evaluate the distributed SPARQL engine described in this paper. The datasets contain Twitter posts and are hosted in the SUWO. They have been used by a joint analytical research program among Tsinghua University[8], KAIST[9] and the University of Southampton[10]. Queries used in the analysis are recorded, and we select those that are used the most and have representative structures (e.g. chain or sparse queries) for query optimisation.

4.1. Schema and Statistics of Twitter Datasets

The Twitter data are composed of 48 million triples having 8 properties (9 including *rdf:type*), 44 million distinct subjects and 21 million distinct objects. These

[8] www.tsinghua.edu.cn
[9] www.kaist.edu
[10] http://www.southampton.ac.uk

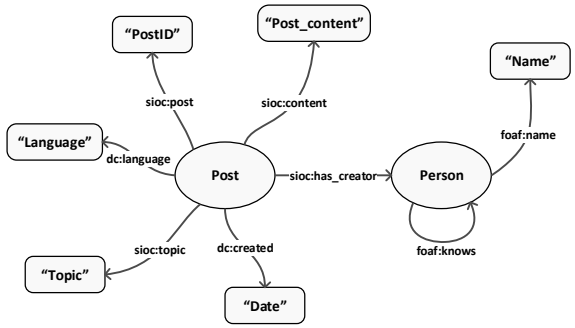

Fig. 2. Ellipses represent resources (i.e. URIs) and rectangles represent literals. Arrows represent predicates pointing from subjects to objects. Name spaces are omitted for simplicity.

triples are evenly distributed in 20 SPARQL endpoints (2.4 million triples each) hosted in the SUWO.

The schema of the Twitter is shown in Fig. 2. Three prefixes are used in the Twitter data:

```
PREFIX sioc:<http://rdfs.org/sioc/ns#>
PREFIX dc:<http://purl.org/dc/elements/1.1/>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
```

There are two classes in the Twitter data: *Posts* and *Persons*. Each *Post* instance has 6 properties:

- An id which is the address of this post, represented by *sioc:post*.
- A language tag which is "en" at this moment, represented by *dc:language*.
- A topic tag, represented by *sioc:topic*.
- A date on which the post is created, represented by *dc:created*.
- A creator which is an instance of *Person*, represented by *sioc:had_creator*.
- Content of this post, represent by *sioc:content*.

For each instance of *Person* there are 2 properties (excluding *sioc:creator*):

- The name of this person, represented by *foaf:name*.
- The acquaintance of this person, represented by *foaf:knows*.

4.2. Analytical Queries

Queries used in evaluation are selected based on two criteria. One is popularity, that represents the number of usage in real analytics, and is obtained from logs of SUWO. Many popular queries are very simple (e.g. contain only one or two triple patterns) and can reveal little of the performance of query optimisation approaches. Therefore we have the second criterion which is query structure. Typical structures such as queries having long chains or star-shapes are considered. Furthermore we select queries having large intermediate results which are most challenging for query optimisation.

Although many different queries have been used to query the Twitter data, most of them share similar patterns. Based on the aforementioned criteria we select 4 queries in total, as shown in table 1.

Table 1
Queries for Twitter analysis

Query 1 Retrieve details of a post by id

```
SELECT DISTINCT *
WHERE { ?p sioc:post        ?pid;
           sioc:content     ?content;
           sioc:topic       ?topic;
           dc:language      ?lang;
           dc:created       ?date;
           sioc:has_creator ?person;
           sioc:post        "postId". }
```

Query 2 Retrieve posts of extended circle (i.e. friends of friends)

```
SELECT DISTINCT ?name ?postId
WHERE
{ <Person> foaf:knows       ?personB;
           foaf:name        ?name.
  ?personB foaf:knows       ?personC.
     ?post sioc:has_creator ?personC;
           sioc:post        ?postId. }
```

Query 3 Find the 6-degree separation of a person

```
SELECT DISTINCT ?personB ?personC
       ?personD ?personE
WHERE
{ <PersonA> foaf:knows ?personB.
   ?personB foaf:knows ?personC.
   ?personC foaf:knows ?personD.
   ?personD foaf:knows ?personE.
   ?personE foaf:knows <PersonF>. }
```

Query 4 Find the extended circle linked following posts

```
SELECT DISTINCT ?personB ?personC
WHERE
{ ?post1 sioc:has_creator <Person>.
  ?post1 sioc:has_creator ?personB.
  ?post2 sioc:has_creator ?personB.
  ?post2 sioc:has_creator ?personC. }
```

5. Evaluation and Discussion

This sections give details of the evaluation of the distributed SPARQL query engine used in SUWO. We give details of the experiment environment and evaluation result. We demonstrate that our approach outperform existing ones in terms of query response time and network overhead.

5.1. Experiment Settings

As described in section 4.2, there are 20 SPARQL endpoints hosted in a cluster of SUWO, each of which has 2.4 million Twitter data. All endpoints are served by Sesame 2.4.0 on Apache Tomcat 6 with default settings.

The query engine used in SUWO is compared to FedX and LHD which are reviewed in section 2. As suggested by previous evaluation result [23,4,27], these two represent the latest achievement in distributed SPARQL queries. All engines under testing are run at a machine having Intel Xeon W3520 2.67 GHz processor and 12 GB memory.

The following metrics are examined in the evaluation.

- Query per second (QPS), that measures the overall performance of query engines.
- Incoming network traffic (INT), that measures the amount of data received from SPARQL endpoints. This metric is the primary indication of the amount of intermediate results.
- Outgoing network traffic (ONT), that measures the amount of queries sent to SPARQL endpoints. Depending on execution techniques, this metrics can be effected by the amount of intermediate results.
- Average transmission rate (ATR), that measures the average bandwidth during query execution, calculated as (ONT+INT)*QPS. This metric is effected by query execution techniques (e.g. parallel execution), but also related to the amount of intermediate results. Usually high ATR does not show with low INT and ONT.

5.2. Evaluation Result and Analysis

The QPS, INT, ONT and ATR of all three engines under testing are shown in Fig. 3, 4, 5 and 6 respectively.

On Q1 both the SUWO engine and LHD have a QPS that is significantly higher (roughly twice faster) than

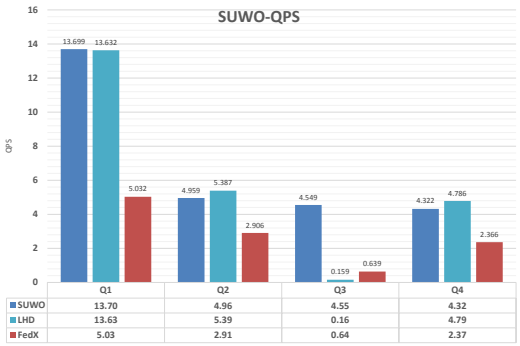

Fig. 3. QPS of SUWO query engine

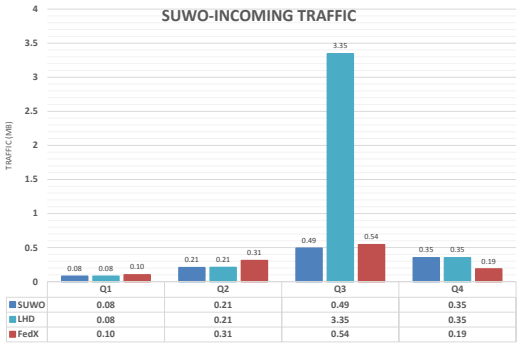

Fig. 4. INT of SUWO query engine

FedX. In the meantime, all three engines have roughly the same amount of INT and ONT, and the advantage of the SUWO engine and LHD is due to higher data transmission rate. This result is because Q1 is a start-shaped query, and both the SUWO engine and LHD execute it in a high parallel fashion. In this case, after executing the triple pattern *?p sioc:post "postId"*, all remain triple patterns are executed simultaneously by the two engines. Although they have the same behaviour on Q1, but is due to different decisions. LHD chooses this execution strategy because this strategy has the shortest response time according to its cost model. On the other hand the SUWO engine choose this strategy by the Ψ algorithm in a way that increases parallelisation without generating extra network traffic. Currently there's no query in SUWO's log that can clearly demonstrate the difference. In future this difference will be shown when more complex queries occur.

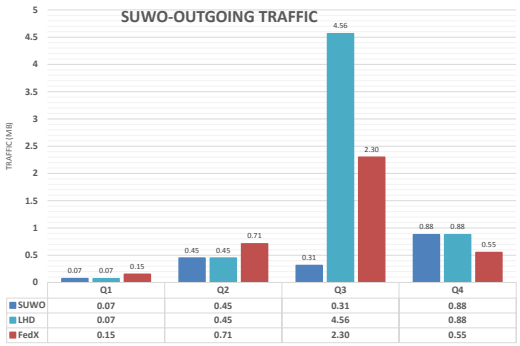

Fig. 5. ONT of SUWO query engine

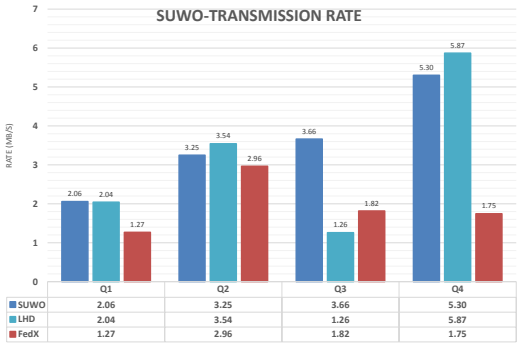

Fig. 6. ATR of SUWO query engine

Q2 mixes a short chain with a few branches. Since each triple pattern has either a different predicate or a different subject or object, selectivity-based cost models such as the one used in LHD are able to make fair estimations. However, since FedX adopts heuristics that counts the number and position of variables in triple patterns, it cannot clearly distinguish every triple pattern. The SUWO engine uses runtime statistics that is able to make the most accurate estimation of triple patterns. However, these statistics are only available after query execution starts and therefore limits the chance of finding an optimal QEP. The QPS on Q2 confirms the analysis above, that LHD is slightly higher than the SUWO engine and both are much faster than FedX.

Q3 is challenging for both LHD and FedX, since none of their cost models can distinguish the triple patterns. Meanwhile, as a result of using runtime statistics, the SUWO engine can still produce a good QEP, and gives a high QPS while the other two are slow. It is worth noticing that LHD shows largest INT and ONT yet a lowest NTR. This is probably due to bad QEP which makes LHD the slowest engine on Q3.

Q4 is also a chain and all three engine have the same order to execute the triple patterns. Therefore this query demonstrate the efficiency of QEP execution of each engine. The SUWO engine and LHD have the same amount of INT and ONT, which confirms that they use the same QEP. However, LHD does not optimise queries at runtime and is slightly more efficient than the SUWO engine. FedX produces least amount of INT and ONT, while having the lowest NTR. This is because FedX executes triple patterns sequentially, and parallelisation is only applied to executed each triple pattern. In the SUWO engine and LHD, however, parallelisation is not only used to executed individual triple pattern, triple patterns are executed in parallel as well.

Overall the SUWO engine shows a fair performance. Using dynamic optimisation can slightly limit the chance of finding the optimal QEP, in the meantime, it helps avoid bad QEP when pre-computed statistics are not sufficient. Due to the limit of evaluation queries only initial evidence of the effectiveness of the Ψ algorithm is shown. A more thorough evaluation will be performed when more queries become available.

6. Conclusions and Future Plans

In this paper we explore techniques for improving query efficiency on LD, and describe the distributed SPARQL engine deployed in the SUWO. This engine adopts an algorithm named Ψ to increase parallelisation without introduce extra network traffic, and optimises queries at query execution time based on runtime statistics. Optimised queries are executed by a efficient concurrent query execution that controls the degree of parallelism with respect to the capacity of each individual dataset.

The aforementioned is evaluated using real Twitter data and queries that have been used in SUWO. By comparing to existing distributed engines we demonstrate the advantage of our approach in terms of query efficiency.

The future plan focuses on three parts. The first is to perform a further evaluation when more queries are logged in the SUWO. With more complex queries it

will be possible to further examine the advantages of the Ψ algorithm.

The second is to further exploit query logs of SUWO. SUWO logs not only frequency of each query, but also the number of results of each triple pattern. As more queries being logged, it is possible to extract statistics more accurate than VoID. To some point these statistics will be accurate enough to support exhaustive algorithms to produce better QEPs. Therefore either the approach described in this paper and LHD will be used depending on the accuracy of available statistics.

The third part is about semantic consolidation, or vocabulary mapping. Currently to query multiple RDF datasets users have to take care of the probability that different vocabularies are used by different datasets. It would require users to have an insight of every dataset and could be a serious issue. In the future we aim to integrate automate mapping in SUWO for popular vocabularies to enable users to query datasets using less vocabularies.

References

[1] D Agrawal, P Bernstein, and E Bertino. Challenges and Opportunities with Big Data. A community white paper developed by leading researchers across the United States. 2012.

[2] Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. Describing linked datasets on the design and usage of VoID , the " Vocabulary of Interlinked Datasets ". In *proceedings of the Linked Data on the Web Workshop (LDOW) , at the International World Wide Web Conference (WWW)*, 2009.

[3] Christian Bizer, T. Heath, and Tim Berners-Lee. Linked Data-The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, January 2009.

[4] Olaf Görlitz and Steffen Staab. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In *proceedings of the Consuming Linked Data Workshop(COLD)*, 2011.

[5] Wendy Hall, Thanassis Tiropanis, Ramine Tinati, Xin Wang, Markus Luczak-Rösch, and Elena Simperl. The Web Science Observatory-The Challenges of Analytics over Distributed Linked Data Infrastructures. *ERCIM Special Theme on Linked Open Data*, 2014.

[6] Olaf Hartig. Zero-Knowledge query planning for an iterator implementation of link traversal based query execution. In Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Pan, editors, *proceedings of the European Semantic Web Conference (ESWC)*, volume 6643 of *ESWC'11*, pages 154–169. Springer, 2011.

[7] Olaf Hartig and Christian Bizer. Executing SPARQL Queries over the Web of Linked Data. *The Semantic Web-ISWC 2009*, 5823:293–309, 2009.

[8] Olaf Hartig and Johann-Christoph Freytag. Foundations of traversal based query execution over linked data. In *Proceedings of the 23rd ACM conference on Hypertext and social media - HT '12*, page 43, New York, New York, USA, June 2012. ACM Press.

[9] Patrick Hayes and Brian McBride. RDF semantics, 2004.

[10] Tom Heath and Christian Bizer. Linked Data: Evolving the Web into a global data space. *Synthesis Lectures on the Semantic Web: Theory and Technology*, 1(1):1–136, February 2011.

[11] P Hitzler and K Janowicz. Linked Data, Big Data, and the 4th Paradigm. *Semantic Web Journal*, pages 233–235, 2013.

[12] Donald Kossmann and Konrad Stocker. Iterative dynamic programming: a new class of query optimization algorithms. *ACM Transactions on Database Systems*, 25(1):43–82, March 2000.

[13] A Labrinidis and HV Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, pages 2032–2033, 2012.

[14] G. Ladwig and Thanh Tran. Linked Data Query Processing Strategies. *The Semantic WebâĂŞISWC 2010*, pages 453–469, 2010.

[15] Günter Ladwig and Thanh Tran. SIHJoin: Querying remote and local Linked Data. *The Semantic Web: Research and Applications*, 6643:139–153, 2011.

[16] P Lyman and H Varian. How much information 2003? *UCBerkeley School of Information Management &*, 2003.

[17] Steven J Lynden, Isao Kojima, Akiyoshi Matono, Akihito Nakamura, and Makoto Yui. A Hybrid Approach to Linked Data Query Processing with Time Constraints. In Christian Bizer, Tom Heath, Tim Berners-Lee, Michael Hausenblas, and Sören Auer, editors, *LDOW*, volume 996 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[18] J Manyika, M Chui, B Brown, and J Bughin. Big data: The next frontier for innovation, competition, and productivity. *McKinsey Global Institute*, page 156, 2011.

[19] MT Özsu and P. Valduriez. *Principles of distributed database systems*. Prentice Hall, 1999.

[20] Viswanath Poosala and Yannis E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *proceedings of the 23rd International Conference on Very Large Data Bases (VLDB)*, pages 486–495, August 1997.

[21] Eric Prud'hommeaux and C Buil-Aranda. SPARQL 1.1 federated query, 2013.

[22] Bastian Quilitz. Querying distributed RDF data sources with SPARQL. *The Semantic Web: Research and Applications*, pages 524–538, 2008.

[23] A. Schwarte, P. Haase, K. Hose, R. Schenkel, and M. Schmidt. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *proceedings of the International Semantic Web Conference (ISWC)*, 2011.

[24] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, page 23, New York, New York, USA, May 1979. ACM Press.

[25] Markus Stocker, Andy Seaborne, Abraham Bernstein, C. Kiefer, and D. Reynolds. SPARQL basic graph pattern optimization using selectivity estimation. In *proceeding of the International Conference on World Wide Web (WWW)*, pages 595–604. ACM, 2008.

[26] Xin Wang, Thanassis Tiropanis, and Hugh C. Davis. Evaluating graph traversal algorithms for distributed SPARQL query optimization. In *proceedings of the Joint International Seman-*

tic Technology Conference (JIST), 2011.

[27] Xin Wang, Thanassis Tiropanis, and Hugh C. Davis. LHD: Optimising Linked Data query processing using parallelisation. In *proceedings of the Linked Data on the Web Workshop (LDOW) , at the International World Wide Web Conference (WWW)*, 2013.