Runtime Characterization of Triple Stores

Long Cheng*†, Spyros Kotoulas†, Tomas Ward* and Georgios Theodoropoulos†
*Department of Electronic Engineering, National University of Ireland, Maynooth
Maynooth, Co. Kildare, Ireland
Email: {lcheng, tward}@eeng.nuim.ie
†IBM Research, Ireland
Mulhuddart, Dublin 15, Ireland
Email: {spyros.kotoulas, georgheo}@ie.ibm.com

***Abstract*—As the Semantic Web becomes mainstream, the performance of triple stores becomes increasingly important. Up until now, there have been various benchmarks and experiments that have attempted to evaluate the response time and query throughput of individual stores to show the weaknesses and strengths of triple store implementation. However, these evaluations have primarily focused on the application level and have not sufficiently investigated system-level aspects to discover performance inhibitors and bottlenecks. In this paper, we are proposing metrics based on a systematic study of the impact of triple store implementation on the underlying platform. We choose some popular triple stores as use cases, and perform our experiments on a standard (128GB RAM, 12 cores) and an enterprise platform (768GB RAM, 40cores). Through detailed time cost and system consumption measures of queries derived from the Berlin SPARQL Benchmark (BSBM), we describe the dynamics and behaviors of query execution across these systems. The collected data provides insight into different triple store implementation as well as an understanding of performance differences between the two platforms. The results obtained help in the identification of performance bottlenecks in existing triple stores implementations which may be useful in future design efforts for Linked Data processing.**

***Keywords*-runtime characterization; triple store;**

I. Introduction

The Semantic Web is becoming mainstream. It is increasingly prevalent particularly among governments and enterprises that see RDF as a more flexible way to represent their data. The availability of several datasets from multiple domains as Linked Data, such as general knowledge (DBpedia [1]), bioinformatics (Uniprot [2]), GIS (geonames [3], linkedgeodata [4]), and web-page annotations (schema.org [5], RDFa [6], microformats [7]), has taken Semantic Web corpora from a lab setting to a terabyte-scale. In tandem with the increasing availability of such data, and corresponding technologies, an increasing number of software platforms now use RDF (e.g. the BBC website [8]).

RDF stores are the backbone of this Web of Data, allowing storage and retrieval of semi-structured information. The engineering of RDF Stores is an active area, represented by roughly four types of approaches [9]: native stores, which have a database engine optimized for RDF processing (Jena-TDB [10], Sesame [11], RDF-3X [12], YARS2 [13], OWLIM [14]), DBMS-backed stores (Virtuoso [15], Allegrograph [16]), representing the RDF Model in relational schema backed by a relational DBMS and hybrid stores, which support both architectures. A fourth approach is to tap on existing platforms for general-purposed distributed processing [17], [18], [19].

Along with the growth in new RDF Store implementations, there has been a corresponding increase in interest for relevant performance evaluations. Liu et al. [20] evaluated 7 RDF stores by comparing data loading and query response time over different size datasets generated using the Lehigh University Benchmark (LUBM). Rohloff et al. [21] implemented the queries and datasets from LUBM to compare the performance of triple stores with different storage backends (such as MySQL etc.) by applying metrics like load time, query response time, query completeness and soundness, and disk-space requirements. Schmidt et al. [22] compared the performance of a single triple store and the vertically partitioned scheme for storing RDF data in DBMS using the SP2Bench SPARQL benchmark. Bizer et al. [23] worked out the Berlin SPARQL Benchmark (BSBM), on the basis of which they performed an evaluation comparing loading time, overall run time and average run time per query. Furthermore, Bröcheler et al. [24] presented an experimental assessment of their DOGMA system by comparing the performance with other RDF database systems in many cases, like query time and index size. And recently, Morsey et al. [25] compared four popular triple stores through measuring QpS (Queries per Second) and QMpH (Query Mixes per Hour) over different size datasets with their DBpedia SPARQL Benchmark (DBPSB). All these reports have provided valuable insight on the performance of RDF stores. However, all these evaluation experiments stay only on an application level but have not gone into the system-level to discover performance inhibitors and bottlenecks.

In this paper, we focus on a more detailed analysis of four of the most popular and mature triple stores, available as open-source software: Jena and Sesame, both written in Java, RDF-3X, a state-of-the-art store for scalable SPARQL processing and, Virtuoso, a commercial RDBMS-based store. We construct suitable metrics and implement our experi-

ments on the basis of BSBM [23] over a standard (128GB RAM, 12cores, standard HDD) and an enterprise platform (768GB RAM, 40 cores, enterprise SAN). The analysis and results allow the dynamics and behaviors of query execution for these particular stores to be better understood and could help in the design of efficient distributed stores, optimized for parallel RDF processing.

Rather than a benchmarking effort, this paper should be read as an analysis of the runtime characteristics of queries for a representative set of RDF stores. We only consider loading times with regard to the feasibility of our experiments and we do not focus on the compliance to the SPARQL specification or the feature-set of each store. In this light, our measurements and analysis aim at guiding development of RDF stores, rather than evaluating existing ones.

Our main findings are that: (1) SPARQL query optimization pays off but introduces risk of performance failures, (2) Current RDF stores do not exploit the parallel nature of modern architectures for single queries, and (3) RDF stores are susceptible to serious performance failures.

The rest of this paper is structured as follows: In Section 2, we provide a short technical background for RDF query processing and detail the metrics used in this paper. The detailed methods we use to collect the data for our proposed metrics are shown in section 3. In Section 4, we describe the experimental environments. We present the test results and discussion in Section 5 and conclude in Section 6.

II. RDF Store Querying

The general query process for most RDF stores is illustrated in Figure 1. It comprises parsing, planning and execution phases. The execution phase comprises processes (joins, data accesses) that are critical to system performance. We examine these phases in turn.

A. Query Processing

Jena, Sesame, RDF-3X and Virtuoso are similar to traditional database systems in terms of system architecture. The main components include a query engine, a storage subsystem and a database. The query engine is used to parse the query from a user or an application program and produce an execution plan, represented as a tree of relational operations [26]. The storage subsystem includes a buffer or even its own file cache manager, which, as the name suggests, manages the buffering of data and reduces the number of disk accesses. Unlike performance evaluations done previously, which have only focused on the time cost of entire query process, here, we divide the data retrieval process into three phases - parsing, planning and execution. We will measure the time cost of each phase to track performance more precisely.

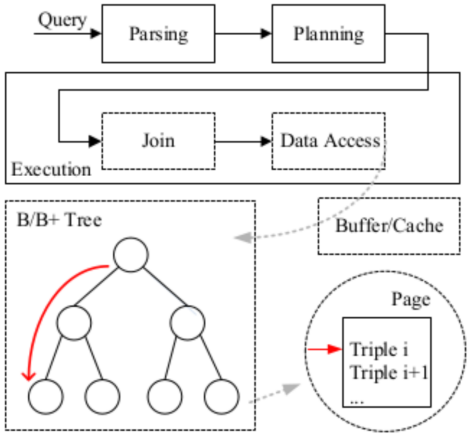

Figure 1. Genral Query Process.

B. Query Planning

Query planning is critical for performance [27]. SPARQL queries typically generate deep query plans, and the query optimizer can only collect limited statistics. In particular,

RDF lacks information about access patterns available in relational databases (e.g. foreign keys). This makes query planning, and in particular join order optimization, challenging and resource-consuming. Some systems (for example, Virtuoso), cache query plans for later use.

C. Join Execution

The execution time of SPARQL queries is dominated by join operations. There are three main join types widely used in triple stores: nested-loop joins, merge joins and hash joins. Jena and Sesame use nested-loop joins, Virtuoso uses all three types of joins and RDF-3X uses merge joins and hash joins.

D. Data Access

If a join operation organizes the general operation of all the triple patterns in a query, then the data access process can be considered as the detailed implementation of retrieving bindings for single triple patterns. This process is always costly and an efficient indexing structure enables fast data retrieval. Jena, Sesame and RDF-3X use B/B+ Tree indexes, suitable for range queries, and the index scheme of Virtuoso contains primary key and bitmp index. In the meantime, Jena provides three triple indexes on spo, pos and osp to fit for different triple patterns, while Sesame offers two indexes spoc and posc by defaut, and RDF-3X maintains 12 indexes (6 indexes and 6 aggregated indices) for covering all the possible join patterns. The redundancy is offset by index compression methods. Virtuoso provides two full indexes posg and pogs and three partial indexes sp, op and gs as default. All systems use a dictionary, mapping values to numeric identifiers. The triple indexes, and most operations, operate on these numeric identifiers.

E. Data Caches

Practically all RDF stores (and all databases) employ caching mechanisms for triple indexes and dictionaries. Data caches are implementation-specific. For example, Sesame employs a caching and buffering approach using the Java heap. During data retrieval, it will access the buffer or cache to check whether the required data is there and start an index scan. If there is no matched data, the needed B-tree node will be read into the buffer first before seeking to the exact data position. Depending on the location of the requested data, some B-tree nodes will be processed directly, some will be read from the disk cache and some will be read directly from disk. Caches influence data access operations like index scans, page reads and triple lookups. To obtain a more precise description of the performance of such operations, we record the number of index scans and their timing, the number of the pages read and the number of the triple lookups for a single query. All these data is useful for describing the dynamics of data searching, which is directly associated with query performance.

Our exercise aims to measure the processing cost of different phases of query execution, and we propose parsing time, planning time, execution time, scan time, number of scan, number of lookups and page read as metrics for this part. In addition, from the operating system level, we aim to measure the CPU usage to evaluate the computation efficiency during the query. In the next section, we go into more detail with regard to the metrics used.

III. Methodology

The previous section gave insight to the workings of RDF stores in general. In this section, we describe in detail the methodology and the metrics used in our experiments. We have instrumented Jena, Sesame and RDF-3X by modifying their source code. Virtuoso already provides some metrics[1] itself, which we retrieve using the Virtuoso JDBC driver.

We measure the time cost for the parsing phase starting from the time we get a query string (in-process), to the time we get the query tree. At this point, the planning phase starts. We consider that the planning phase is finished when we get an execution plan. Note that for the purposes of this paper, any runtime decisions (for example, sideways information passing techniques used in RDF-3X) are counted as part of the execution phase and not as part of the planning. The execution phase is finished when the last result has been received.

For Virtuoso, we retrieve other relevant metrics using the corresponding SQL statements after each query. For the other three systems, all other metrics are collected through inserting counters in the program code.

Four counters are assigned for a scan as pseudo codes below demonstrates. We define the start of an index scan with the reading of the root node of the index and the end with release of the root node. A separate counter for the number of pages accessed is used in this process (Counter 2 in the code), here it indicates only part of pages read, while there would be no scan when stores read the consequent pages. We also insert counters for that.

```
# Index and Search Range [a,b] have been confirmed.
Counter1: scan_start_time;
start
 read (tree.root())
 #binary search to get child node
  repeat
   read (child.node())
   Counter2: page_read_1++;
  until triple_id = a ;
 release(tree.root())
end.
Counter3: scan_number++;
Connter4: scan_end_time;
```

Figure 2. Pseudo codes of four counters in a scan implementation.

Table I
METRICS LIST

Metrics	General	Detailed
Loading Time	✓	
Disk Consumption	✓	
QMpH	✓	
Parsing Time		✓
Planning Time		✓
Execution Time		✓
Number of Index Scans		✓
Scan Time		✓
Number of Lookups		✓
Read in Pages		✓

A triple is located in a slot of data page and an extra ID is used to indicate the slot. Our lookup counters increment every time when system lookup this id to check whether it meet the searching range. Lookup happens during the scan for RDF-3X and accompany with results retrieve process for Jena and Sesame.

Lastly, we monitor the CPU usage during all the tests with SYSTEMTAP [28] , a tool for gathering information about system utilization in the Linux operating system. The CPU was sampled every second. For the purposes of this paper, we consider 100% CPU usage when a single logical processing unit is fully utilized (i.e. a dual-core machine with two threads per core can have up to 400% CPU usage).

We will present results for some metrics across all queries and for some others, we will focus on specific queries. The metrics used in this paper are summarized in Table I.

[1] http://docs.openlinksw.com/virtuoso/ptune.html

IV. Experimental Settings

A. Benchmark

For our experiments, we have used the Berlin SPARQL Benchmark (BSBM) [23]. BSBM generated synthetic datasets of arbitrary size, representing an e-commerce use-case in which a set of products is provided by various vendors and consumers post reviews around those products. We created a series of datasets, the largest of which is composed of 5 billion triples, occupying around 1.2 TB in N-Triples format.

We have concentrated on the explored use-case of BSBM. Although the corresponding query set is suitable for use with Jena, Sesame and Virtuoso, several query features such as DESCRIBE and OPTIONAL are not supported by RDF-3X. Consequently, we rewrote the queries[2] to cater to RDF-3X.

B. Platform

All the experiments were conducted on two platforms, a Standard Platform (SP) and Enterprise Platform (EP). Their configurations are shown in Table II.

The standard platform we have used is an iDataPlex node with 2 Intel Xeon X5679 processors, 128GB RAM with a single 1TB SATA HDD. The enterprise platform consisted of a high-memory server IBM x3850 X5, an enterprise-grade IBM XIV SAN and two IBM System Storage SAN48B-5 fiber channel switches. The server was equipped with 4 Intel Xeon E7-8850 processors, 768GB RAM and two Emulex 8Gb FC Single-port HBAs, each connected to one switch. The XIV SAN used 156 HDDs and was connected to each switch on six fiber channel ports. Both platforms use hardware multithreading (namely, each core can run two separate threads).

C. Setup

We have experimented on Jena v2.6.4, Sesame v2.6.5, RDF-3X v0.3.7 and Virtuoso Open Source v6.1.5. We set the Java heap size for Jena and Sesame to 40GB on SP and 240GB on EP. For Virtuoso, the NumberOfBuffers and MaxDirtyBuffers were set to 5242880 and 3932160 on SP and 31457280 and 23592960 on EP. For Jena, we have configured the optimizer with the statistic optimization strategy. For Sesame we have set the index configuration to spoc, posc and opsc. The rest of the parameters were left to the default values. We chose 150 query mixes for our experiments, of which 50 query mixes were used in the warm-up phase and the other 100 were in the hot-run. To minimize the caching effects of previous queries, we empty the file system cache before running the query mixes of each test. The rest of the parameters were left to the default values.

We chose 150 query mixes for our experiments, of which 50 query mixes were used in the warm-up phase and the other 100 were in the hot-run. To minimize the caching effects of previous queries, we empty the file system cache before each test. The rest of the parameters were left to the default values.

Our experiments have been limited by the following conditions: (1) hard disk space, (2) loading time (with a cut-off at 100 hours) and (3) query execution time (with a cut-off at 24 hours).

[2]http://code.google.com/p/para-computing-long/downloads/list

Table II
THE CONFIGURATIONS OF TEST PLATFORMS

Machine	Standard Platform	Enterprise Platform
CPU	2*6 Cores, 2.93GHz	4*10 Cores, 2.00GHz
RAM	128GB	768GB
Disk	1TB	XIV SAN
Linux Kernel	rhel-2.6.32-220	rhel-2.6.18-308
Java Version	1.6.0_25	1.6.0_25

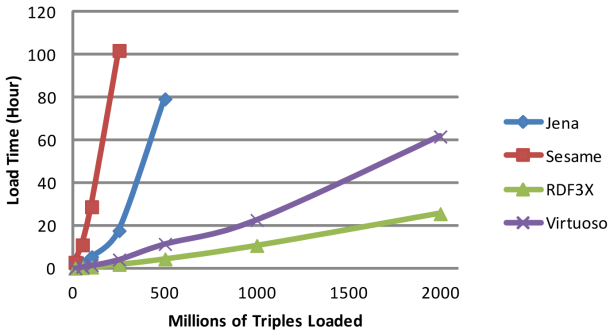

Figure 3. Load time on Standard Platform.

V. Results and Discussion

In this section we analyze the results derived from the metrics described previously and provide insight on the runtime characteristics of the aforementioned RDF stores for the platforms used.

A. Loading

The loading times for SP and disk consumptions are shown in Figure 3 and Figure 4. Sesame performs poorest in terms of loading capability - 250M triples take more than 100 hours. A possible explanation for this lies in the relatively small page size used by Sesame, which is only 2KB, that leads to very frequent index updates. On EP the situation is improved as 500M is loaded in about 34 hours. For Jena, 500M was loaded in 80 hours for SP while the performance on EP was dramatically superior - 5B loaded in 70 hours. In comparison, RDF-3X took 75hours. Virtuoso and RDF-3X achieved much faster loading speeds than Jena and Sesame on SP, but their loading time on EP is at the same levels as in SP, indicating that they did not exploit the hardware.

For disk space requirement, we observe that the index compression methods of RDF-3X pay off, resulting in the

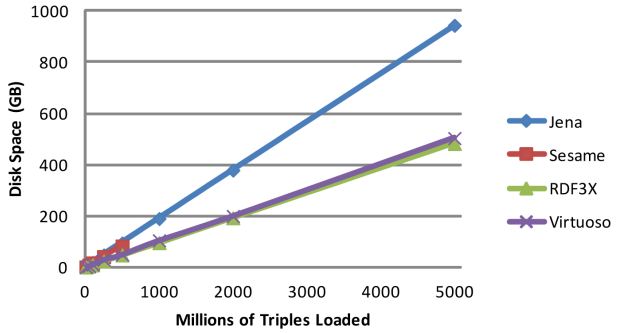

Figure 4. Disk space required for various datasets.

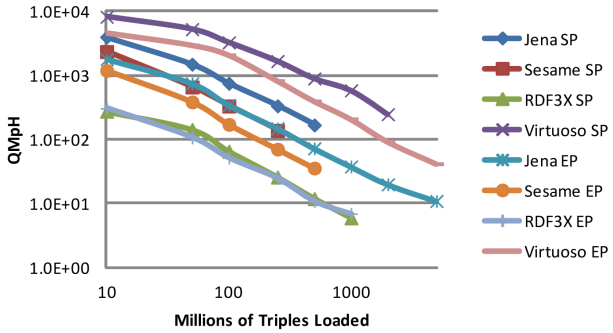

Figure 5. QMpH for various datasets.

Table III
SPECIAL QUERIES FOR RDF STORES WITH 250M TRIPLES ON STANDARD PLATFORM

RDF Store	Best Queries	Worst Queries
Jena	Q2, Q9, Q12	Q3, Q4, Q5
Sesame	Q2, Q9, Q12	Q5, Q10, Q11
RDF-3X	Q2, Q11,Q12	Q5, Q7, Q8
Virtuoso	Q2, Q9, Q12	Q3, Q5, Q8

smallest index size. Virtuoso (also using index compression) uses nearly 10% more space. Jena and Sesame generate much larger indexes about 2 times larger than those of RDF-3X.

B. QMpH

There are 25 queries in a Query Mix, which is the same as the BSBM configuration, and Figure 5 shows the QMpH result on the basis of our rewritten queries. As expected, performance decreases with an increasing dataset size for all stores in both platforms. This change is essentially linear as demonstrated in the figure. We also note that RDF-3X performed the worst of all stores, which comes in contradiction with [12]. We will explain this further in the following part C.

Comparing the two platforms, EP appears to have worse QMpH than SP, which is surprising. In terms of hardware configuration, SP has only one advantage, namely CPU clock speed. We draw the conservative conclusion that computation is CPU-bound and explain this further in part G of this section.

With the results mentioned above, regarding to our test strategy, the maximum number of triples for each store in our experiments is also expressed in Figure 5 according to the terminal points of different curves. We measured the QpS of 250M triples on SP for all stores, which we believe that it could indicate a general performance for our test. Based on that, we also listed three queries with the best QpS and three with the worst for each store as shown in Table III. Since that Q5, Q8 and Q12 appears frequency in that list, with the interest in outstanding queries, we chose these three queries as main analysis objects for our proposed metrics in the following.

C. Cost Breakdown

Figure 6 shows the cost breakdown between query parsing, planning and execution, across all stores and queries for 250M triples. For most stores, the runtime is dominated by execution time. Query parsing represents a small fraction of the cost, so we will exclude it from further discussion. Planning cost differs significantly per store, with Virtuoso spending significantly more time than the other stores. And for Q5, Q7 and Q8 we see that the execution time of RDF-3X is nearly 100%. And especially for Q7, which appears four times in the query mix, and its QpS for RDF-3X is only 0.03, which is extreme low while other stores is in the order of ten, that leads RDF-3X a worse QMpH as mentioned above.

D. Planning and Execution

Query12 is chosen as being representative for further analyzing planning costs. Our results in Figure 7 show that: (1) Planning costs for Virtuoso and Jena are fairly constant and are not significantly influenced by dataset size. We attribute this to the plan caching and the statistical approach taken in these systems respectively. (2) Sesame and RDF-3X clearly demonstrate an increase in planning costs as the dataset size increases.

The Virtuoso query planner dominates runtime, especially for Q5 (not shown). In this case, taking 808.3ms for an average query runtime of 808.4ms. This illustrates that performance failures in the query plans can be lethal in RDF stores. On the other hand, the significant effort for optimization pays off, as shown in the execution times in Figure 6, where Virtuoso significantly outperforms other stores. We should nevertheless note that this optimization cost is not amortized over the (lower) execution time, as we describe next.

Query 5 is a bad query for all four stores, as evident in the execution times presented in Figure 8. We can see that the execution time is basically linear with the data size for Jena,

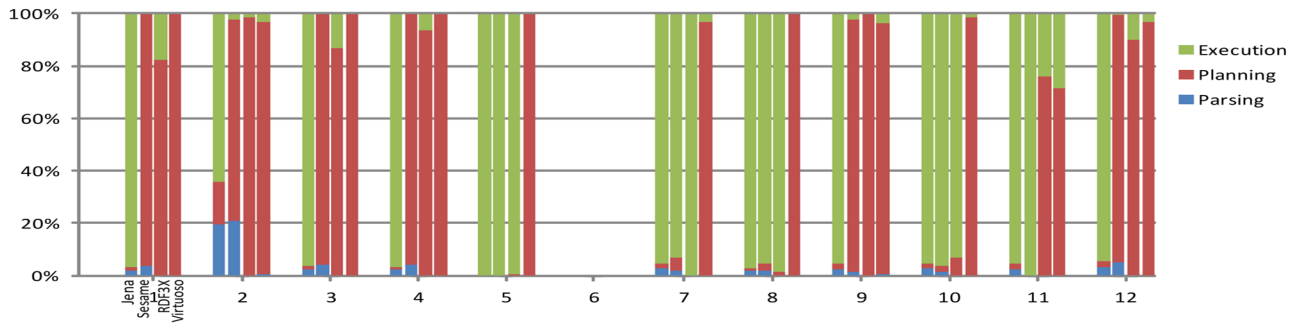

Figure 6. Breakdown of different Queries for 250M Triples on Standard Platform.

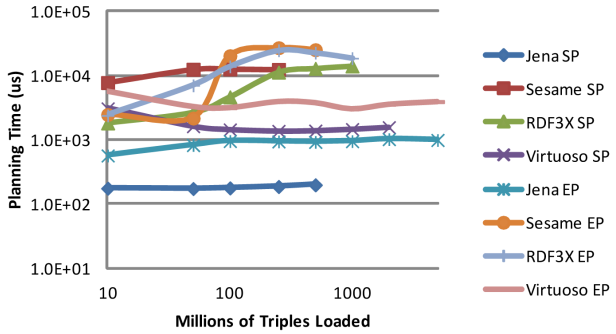

Figure 7. Planning Time of Query 12.

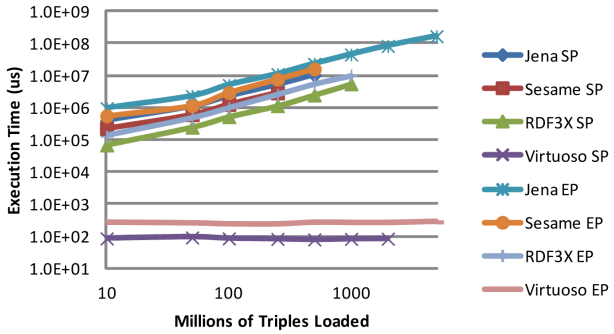

Figure 8. Execution Time of Query 5.

Sesame and RDF-3X, with the latter performing better. The time cost of Virtuoso is practically constant with the dataset size, indicating that a large portion of the computation for this query is done during the planning phase.

E. Number of Scans and Scan Time

In terms of number of scans and scan time, the Virtuoso provided metric Locks is always 0 for all the queries in our experiments, which indicates no index is locked during the query implementation, we assume the reason is that perhaps the results are stored in the store cache and there is no need to search the triples through an index scan. And it is also possible that the lock refers to the number of locks needed for synchronization, and the implementation contains only read operations. We report then Number of Scans and Scan Time only for Jena, Sesame and RDF-3X, since the instrumentation in Virtuoso does not support these metrics. In Figure 9, we show results for Query 8. The curves do not change after 250M triples. The number of scans for RDF-3X is smaller than Jena and Sesame. We attribute this to the fact that RDF-3X maintains indexes for all term permutations. On the right part of Figure 9, we also see that the time spent scanning is significantly higher for RDF-3X (again, this is due to its architecture). In the same Figure, Sesame spent much less time scanning, since it is using its proprietary cache, compared the file cache used by Jena.

F. Number of Lookups and Read in Pages

Figure 10 shows the number of triples retrieved (triple lookups) and pages read for Query 5. These both grow linearly with dataset size. Given the fact that it heavily relies on sequential data access, RDF-3X retrieves much more data (on both metrics). The difference in page reads between Sesame and Jena is attributed to the proprietary cache of the former. Comparing these two results with Figure 7, we see that even though RDF-3X accesses more data, it strongly outperforms Sesame and Jena.

G. CPU Usage

For all systems, we observed very low CPU usage and although Jena, Sesame and Virtuoso support concurrent evaluation of multiple queries, no system parallelizes the execution of single queries. The CPU usage of Jena and Sesame was almost identical (70% $\sim$ 80%) on both platforms. RDF-3X and Virtuoso reached 100% CPU on the standard platform, and Virtuoso reached 200% on the enterprise platform. Given that SP and EP have 24 and 80 logical processing units respectively, none of the systems exploit the parallel nature of modern architecture for the evaluation of single queries.

VI. Conclusions and Future Work

We have conducted a comparative analysis of the runtime characteristics of a representative set of RDF stores, namely,

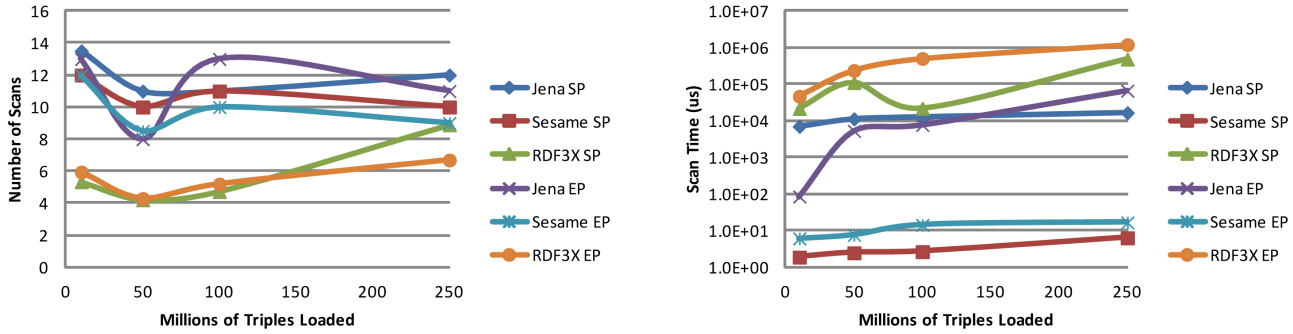

Figure 9. Number of Scans and Scan Time of Query 8. The curves follow the same pattern for datasets larger than 250M triples.

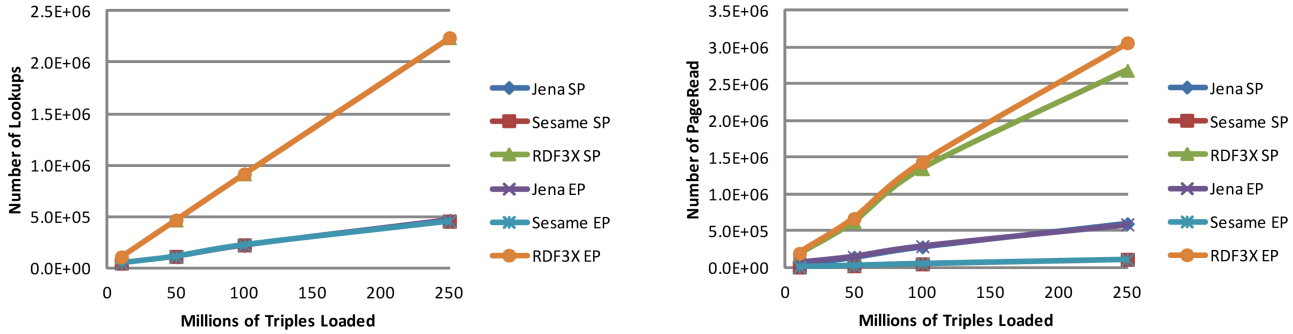

Figure 10. Number of LookUps and PageRead of Query 5. The curves follow the same pattern for datasets larger than 250M triples.

Jena, Sesame, RDF-3X and Virtuoso. We have described the dynamics and behaviors of the query execution on the basis of experimental data and queries derived from the BSBM benchmark. For the first time in the literature, we report on the performance and characteristics of triple stores on an enterprise platform.

Our main findings were the following: (1) Investing in query optimization pays off in general, but, in SPARQL, it is easy to arrive at a situation in which the runtime performance is dominated by optimization with a deleterious effect on the runtime. (2) Planning failures are potentially catastrophic. In our experiments, although RDF-3X was the fastest system in most queries, failure in a single query resulted in it having the worst overall performance. (3) None of the RDF stores examined can exploit modern parallel architectures for single queries. This is expected to have a very negative effect on analytical workloads. (4) Using very fast storage, in most cases, did not have the expected impact on performance. This indicates that either the datasets used were completely served by data in memory and caching techniques performed adequately, or that query processing in RDF stores is actually CPU-bound.

We are currently planning experiments using Solid State Storage (SSD) attached directly to the PCIe bus. The bandwidth of this setup would be comparable to the one obtained by an enterprise SAN and the SSD would be superior in terms of response times. Our long-term goal is the development of highly parallel and distributed architectures that cater specifically to processing of RDF data.

Acknowledgment

This work is supported by Irish Research Council and IBM Research, Ireland.

References

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: a nucleus for a web of open data," in *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, ser. ISWC'07/ASWC'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 722–735.

[2] R. Apweiler, A. Bairoch, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, D. A. Natale, C. Oonovan, N. Redaschi, and L. L. Yeh, "Uniprot: the universal protein knowledgebase," *Nucleic Acids Research*, vol. 32, no. suppl 1, pp. D115–D119, 2004.

[3] Geonames.org, http://www.geonames.org.

[4] C. Stadler, J. Lehmann, K. Höffner, and S. Auer, "Linkedgeodata: A core for a web of spatial open data," *Semantic Web Journal*, 2011.

[5] Schema.org, http://schema.org.

[6] B. Adida and M. Birbeck, "Rdfa primer. bridging the human and data webs. w3c working group note 14," http://www.w3.org/TR/xhtml-rdfa-primer/, October 2008.

[7] J. Allsopp, *Microformats: Empowering Your Markup for Web 2.0*. Berkeley, CA: Friends of Ed, 2007.

[8] Y. Raimond, T. Scott, P. Sinclair, L. Miller, S. Betts, and F. McNamara. (2012) Case study: Use of semantic web technologies on the bbc web sites. [Online]. Available: http://www.w3.org/2001/sw/sweo/public/UseCases/BBC/

[9] B. Haslhofer, E. M. Roochi, B. Schandl, and S. Zander, "Europeana rdf store report," University of Vienna, Vienna, Technical Report, March 2011.

[10] A. Owens, A. Seaborne, N. Gibbins, and mc schraefel, "Clustered tdb: A clustered triple store for jena," November 2008.

[11] J. Broekstra, A. Kampman, and F. van Harmelen, "Sesame: A generic architecture for storing and querying rdf and rdf schema," in *Proceedings of the First Internation Semantic Web Conference*, ser. Lecture Notes in Computer Science, I. Horrocks and J. Hendler, Eds., no. 2342. Springer Verlag, July 2002, pp. 54–68.

[12] T. Neumann and G. Weikum, "Rdf-3x: a risc-style engine for rdf," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 647–659, Aug. 2008.

[13] A. Harth, J. Umbrich, A. Hogan, and S. Decker, "Yars2: a federated repository for querying graph structured data from the web," in *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, ser. ISWC'07/ASWC'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 211–224.

[14] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov, "Owlim: A family of scalable semantic repositories," *Semant. web*, vol. 2, no. 1, pp. 33–42, Jan. 2011.

[15] O. Erling and I. Mikhailov, "RDF Support in the Virtuoso DBMS," in *Proceedings of the 1st Conference on Social Semantic Web CSSW*, vol. 221. Springer, 2007.

[16] W3C, "Allegrograph rdfstore web 3.0's database," Online http://www.franz.com/agraph/allegrograph/, September 2009.

[17] "Rapid: Enabling scalable ad-hoc analytics on the semantic web," in *8th International Semantic Web Conference (ISWC2009)*, October 2009.

[18] Z. Kaoudi, K. Kyzirakos, and M. Koubarakis, "Sparql query optimization on top of dhts," in *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I*, ser. ISWC'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 418–435.

[19] M. Karnstedt, K.-U. Sattler, and M. Hauswirth, "Scalable distributed indexing and query processing over linked data," *Web Semant.*, vol. 10, pp. 3–32, Jan. 2012.

[20] B. Liu and B. Hu, "An evaluation of rdf storage systems for large data applications." in *SKG*. IEEE Computer Society, 2005, p. 59.

[21] K. Rohloff, M. Dean, I. Emmons, D. Ryder, and J. Sumner, "An evaluation of triple-store technologies for large data stores," in *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems - Volume Part II*, ser. OTM'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 1105–1114.

[22] M. Schmidt, T. Hornung, N. Küchlin, G. Lausen, and C. Pinkel, "An experimental comparison of rdf data management approaches in a sparql benchmark scenario," in *Proceedings of the 7th International Conference on The Semantic Web*, ser. ISWC '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 82–97.

[23] C. Bizer and A. Schultz, "The berlin sparql benchmark," *Int. J. Semantic Web Inf. Syst*, vol. 5, no. 2, pp. 1–24, 2009.

[24] M. Bröcheler, A. Pugliese, and V. S. Subrahmanian, "Dogma: A disk-oriented graph matching algorithm for rdf databases," in *Proceedings of the 8th International Semantic Web Conference*, ser. ISWC '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 97–113.

[25] M. Morsey, J. Lehmann, S. Auer, and A.-C. N. Ngomo, "Dbpedia sparql benchmark: performance assessment with real queries on real data," in *Proceedings of the 10th international conference on The semantic web - Volume Part I*, ser. ISWC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 454–469.

[26] M. S. J. Hammer, *Data Structures for Databases*, 2001, vol. 60.

[27] T. Neumann and G. Weikum, "Scalable join processing on very large rdf graphs," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 627–640.

[28] Systemtap, http://sourceware.org/systemtap/.