WP2: Triple Store Benchmarking

Dieter De Witte - iMinds Data Science Lab
Laurens De Vocht - iMinds Data Science Lab

1 Introduction

In this project iMinds will help Europeana with the selection of a triple store to host their semantic data. Prior to this project Ontotext GraphDB was used for this purpose but the choice was made to experiment an open source license solution.

The number of RDF solutions to choose from is vast and the optimal choice heavily depends on the properties of the data and the type of queries which will make up the workload.

Based on the data of Europeana and a set of queries iMinds will run a benchmark which will motivate the selection of a triple store which will be used to support the future Semantic Web activities of Europeana.

2 Technical Requirements

2.1 Requirements for both parties

Requirements for iMinds:

- Infrastructure for benchmarking:
  - 1 server per triple store
  - multiple client for executing benchmarks
  - administrator rights on all nodes

- Europeana dataset and queries
  - dataset available on all server nodes
  - dataset available as RDF in turtle format or N-triples format

Requirements for Europeana's triple store selection:

- Triple stores should be Open Source (community edition is not sufficient)
- Triple store must handle dataset of 2+ billion triples
- Support for reasoning capabilities is not a requirement
- Support for horizontal scalability is not a requirement (single server)

- Dataset dynamics: update frequency is currently on a monthly basis and need not be higher than once per day in the future. Higher update frequencies are not a requirement.
- The triple store should be well maintained and sufficiently mature to run in a production environment.

2.2 Meeting the requirements

- Infrastructure requirements:
    - June 17: Servers for triple stores and benchmark clients were made available by Christos.

- Dataset availability:
    - June 17: First version of the data was in GraphDB proprietary data format which cannot be loaded in the suggested triple formats
    - June 21: Data uploaded to all servers, format is RDF/XML. Issue raised to convert data to turtle or n-triples as agreed.
    - June 23: Data converted to turtle, uploaded to all servers until the next day.

- June 14: Triple store selection was approved at the initial meeting

2.3 Tasks and planning

Worktype	**Person**	**MD**	**Description**	**Status**
Meeting	Dieter	0.75	Initial meeting & preparation, hardware suggestions	Done.
Meeting	Dieter	0.25	Follow up	Done.
Development	Dieter	0.5	Query Validation	Done.
Development	Dieter	2.5	Virtuoso Setup & Benchmarking	Done.
Development	Laurens	3	Blazegraph Setup & Benchmarking	Done. Some files dropped due to parse exceptions.

Development	Dieter	2.5	Fuseki Setup, benchmarking and HDT creation	HDT from XML used, HDT from Turtle too many parse errors
Development	Dieter	/	Neo4J Setup	Canceled due to re-scoping for handling data quality issues.
Development	Dieter	/	Neo4J Benchmarking	Canceled due to re-scoping for handling data quality issues.
Development	Dieter	1	Benchmark clients	Done.
Analysis	Dieter	1	A jupyter notebook where results can be plugged in and visualized	Done.
Project Management	Dieter	0.5		Done.
Reporting	Dieter	1		Done.
Total		**13**	**Triple store benchmarking**	**Done.**

3 Triple Store selection procedure

3.1 Triple stores under consideration

Triple stores selected for this benchmark study must be:

- Well maintained to support Enterprise operation
- Have a high adoption to ascertain support over a long period of time
- Have an open source license

One website which measures the adoption/popularity of databases is DB-engines. It has a section on RDF stores. The most popular open source RDF database in this list is Virtuoso (#2), followed by Jena (#3) and Sesame (#4). GraphDB (#12) and BlazeGraph (#14) have a lower rank which might be partially attributed to their rebranding. Neo4J is the most popular Graph Database but uses Cypher and not SPARQL as its native query language.

Triple Store	Details	Comments

S2RDF	Research by Freiburg on translating SPARQL engines to Apache Spark for easier integration and horizontal scalability Paper link	Europeana: Request not to use Apache Spark to avoid introducing a new platform to Europeana's existing Cloud infrastructure. (requires Cloudera Hadoop CDH)
Virtuoso Open Source	Most popular entreprise triple store, generally performs best in benchmarks. Open Link	Both Open Source as a closed source version with enterprise support available. Very scalable: both scale out and high availability
Blazegraph	Graph Database with RDF/SPARQL support. Blazegraph	Open Source, no license fees. Used by Wikidata. Both scale out and high availability. For parallel setup enterprise support is recommended.
Jena Fuseki with HDT	Fuseki is a Jena-based SPARQL-over-HTTP solution. Can run with TDB as a backend or on top of HDT files. Fuseki website HDT website	Not a scalable solution but due to its ability to run on compressed HDT format a good candidate for big datasets and interesting point of comparison with LDF.
GraphDB	Entreprise RDF Graph Database with many features. GraphDB	Not chosen since it's proprietary and will definitely not be chosen.
Neo4J with RDF plug-in	Graph database excelling in path queries. Neo4J Unmanaged extension	Potential issue: plugin is not maintained by Neo4J
RDF4J	Formerly Sesame. Currently in Beta	In literature Sesame never shows up as the most performant store.

Table 1: Triple stores considered for Europeana. Stores marked in green were selected for benchmarking. Neo4J was selected as the 4th store with a lower priority.

3.2 Benchmark Results in Scientific Literature

In scientific literature the use of artificial benchmarks is very common. The most cited ones are:

- LUBM: Lehigh University Benchmark[1]
- BSBM: the Berlin SPARQL Benchmark[2]
- DBPBM: DBPedia Benchmark[3]
- SP2Bench: SPARQL performance Benchmark[4]

In a more recent study about the diversity of SPARQL benchmark suites by University of Waterloo[5] it was shown that previous benchmarks are not sufficiently diverse in terms of query properties nor the selectivity properties of the data. They provide a suite - Watdiv - which offers both queries and data which addresses these issues. Results from Watdiv will be used to further motivate the choice of Sparql endpoint for Europeana.

Currently 2 European projects are focusing on trying to provide industry-level benchmarks for testing triple stores:

- LDBC: Linked Data Benchmark Council
- HOBBIT: Holistic Benchmarking of Big Linked Data

In a recent benchmark study (Benchmark Toyama 2012[6]) a set of triple stores was compared with Virtuoso, GraphDB (OWLIM) and Blazegraph (Bigdata) being the most promising ones. For the biggest datasets - Uniprot and DDBJ - both in the order of 1-10 billion triples Virtuoso showed the best performance.

Performance benchmarks only evaluate stores along one dimension. Therefore it is equally interesting to evaluate stores based on the features they support. This exercise was done before by Wikimedia. They added weights to a set of features which allowed them to create a score and a ranking of different RDF management solutions.

- Blazegraph comes out on top of this evaluation with a score of 1597.
- Virtuoso has 159 points, but a lot of values are missing and therefore counted as zero. Apart from that we are somewhat sceptical about some of the scores. For example ‘maturity of distributed version’ is much higher in Virtuoso but both Virtuoso and Blazegraph get a 1, ability to maintain uptime of Virtuoso gets a zero since it’s an enterprise feature. Blazegraph on the other hand also recommends enterprise support for certain features although in theory they can be used without a license, in practice it’s very hard to do so. As a conclusion we postulate that after filling in the missing values the score for Virtuoso will be of the same order as Blazegraph, so definitely > 1000!
- GraphX is one of the possible Spark solution which gets a very high ranking. Again here we expect some bias, Virtuoso is definitely more mature and suitable as an RDF store.

- Neo4J and Titan are the highest scoring native databases. The fact that they have no native support for RDF and SPARQL would make it definitely a less viable option than Virtuoso and Blazegraph.

Also this wiki provides a lot of details of the features of many different stores allowing Europeana to take these into consideration.

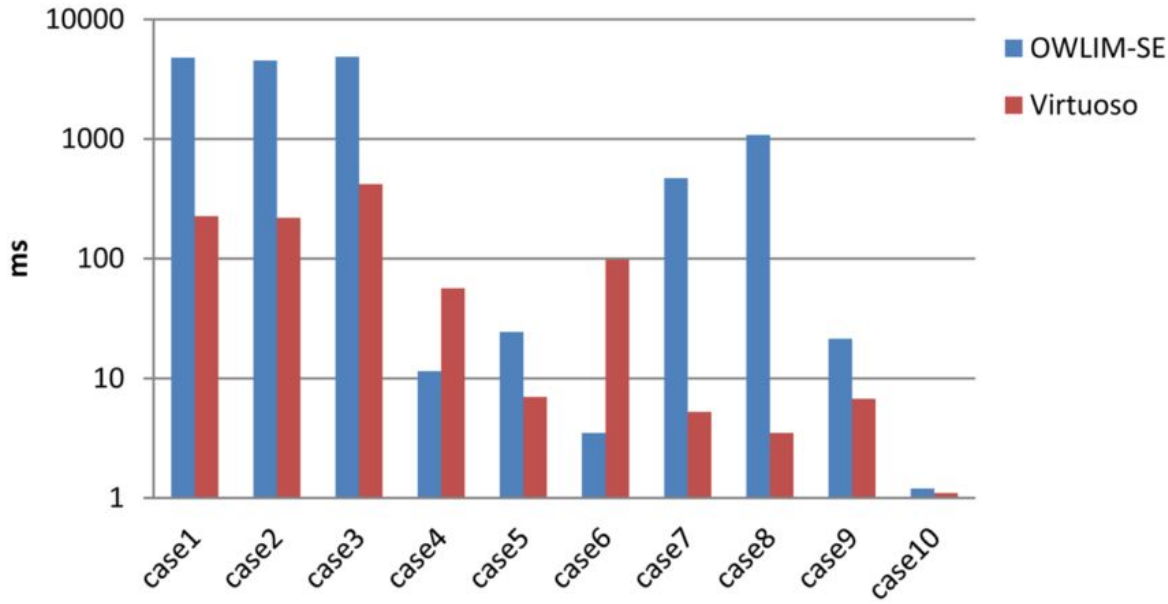


Figure 1. Virtuoso shows the best performance in a study on life science datasets.

[1] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. Web Semantics: Science, Services and Agents on the World Wide Web, 3(2):158–182, 2005.
[2] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. International Journal on Semantic Web and Information Systems (IJSWIS), 5(2):1–24, 2009.
[3] M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. DBpedia SPARQL benchmark–performance assessment with real queries on real data. The Semantic Web–ISWC 2011, pages 454–469, 2011.
[4] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. SP2Bench: a SPARQL performance benchmark. In Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on, pages 222–233. IEEE, 2009.
[5] G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee. Diversified stress testing of RDF data management systems. In The Semantic Web–ISWC 2014, pages 197–212. Springer, 2014.
[6] H. Wu, T. Fujiwara, Y. Yamamoto, J. Bolleman, and A. Yamaguchi. BioBenchmark Toyama 2012: an evaluation of the performance of triple stores on biological data. Journal of biomedical semantics, 5(1):1, 2014.

4 Infrastructure configuration

To set up the stores two approaches can be followed:

- Installation based on the online documentation
- Installation based on dockerfiles

The docker file has the following advantages/disadvantages:

+ Maintained by an external partners + automatically stays up to date
+ Used and tested by a larger audience
+ Single command setup
+ A docker file is in fact an installation recipe

- No enterprise guarantees

Docker images can be found on Docker Hub. Docker is a virtualization approach so the containers can be deployed independent of the OS. Details on the installation of docker are on the wiki page, the scripted version is found under `/prereq`.

Some containers require additional software during the setup process. Scripts will be provided via a **github repository**. The github repository contains a very broad set of **wikipedia pages**. The starting location of which can be found **here.**

4.1 Preparation

4.1.1 Triples dataset

The initial data offered by Europeana consisted of a dump of their current Sparql endpoint. This endpoint was using Ontotext GraphDB as a data management system. Since this format cannot be used by other triple stores Europeana provided a dataset in RDF/XML format and in RDF/Turtle format. The latter is format is preferred since it is less verbose and more easy to parse and apply postprocessing. The RDF/XML dataset was used in WP1 to create a set of HDT files and was also used further on in the project due to the fact that the RDF/Turtle dataset still has a number of data quality issues (which are being noted for discussion with Europeana's group working on data quality). Due to the aforementioned advantages of RDF/Turtle we recommend that Europeana tries to correct these issues for future use.

4.1.2 HDT compression

In WP1 the RDF/XML dataset was converted to HDT. The same process was repeated starting from the RDF/Turtle dump. As a preprocessing step the turtle files in directory `dump_30_6_2016` where grouped per prefix and concatenated. (scripts available, though trivial). This preprocessing phase revealed a first issue with the RDF/Turtle dataset: files with prefix 3 to 9 and 45 are not present in the RDF/Turtle dump!

To create HDT files from both RDF/XML or RDF/Turtle we recommend using the rfdhdt docker container. Creating an HDT from a turtle file is as simple as:

sudo docker run -it --rm -v $(pwd):/data rfdhdt/hdt-cpp rdf2hdt -f turtle /data/_2.ttl /data/_2.hdt

The HDT compression algorithm is very sensitive to anomalies in the dataset. Unfortunately the software doesn't just skip malformed triples, it terminates the conversion upon error. In the file HDTCompression.xlsx in the tab `sidetoside` the number of triples per file are compared between the conversion of RDF/XML to HDT and the conversion starting from the Turtle dump. The regular conversion from RDF/XML has 1.93 billion triples while the conversion from RDF/Turtle only has 0.75 billion triples. Therefore we chose to work with the HDT files of WP1. (actual errors can be found in HDTCompression.xlsx, first tab)
Note that the number of triples in an HDT file can be seen by simply opening the HDT file in a text editor, the number of triples is one line 3. Differences in the number of successfully generated triples can be seen in Figure 2.

An important reason we recommend the cleanup of the RDF/Turtle dump is the big performance difference for handling these two data types. The conversion to HDT from **RDF/XML** takes approximately **18 hours** while the conversion from **RDF/Turtle** is expected to take 217 minutes or **3 hours 47 minutes** (calculation in HDTCompression.xlsx, first tab).

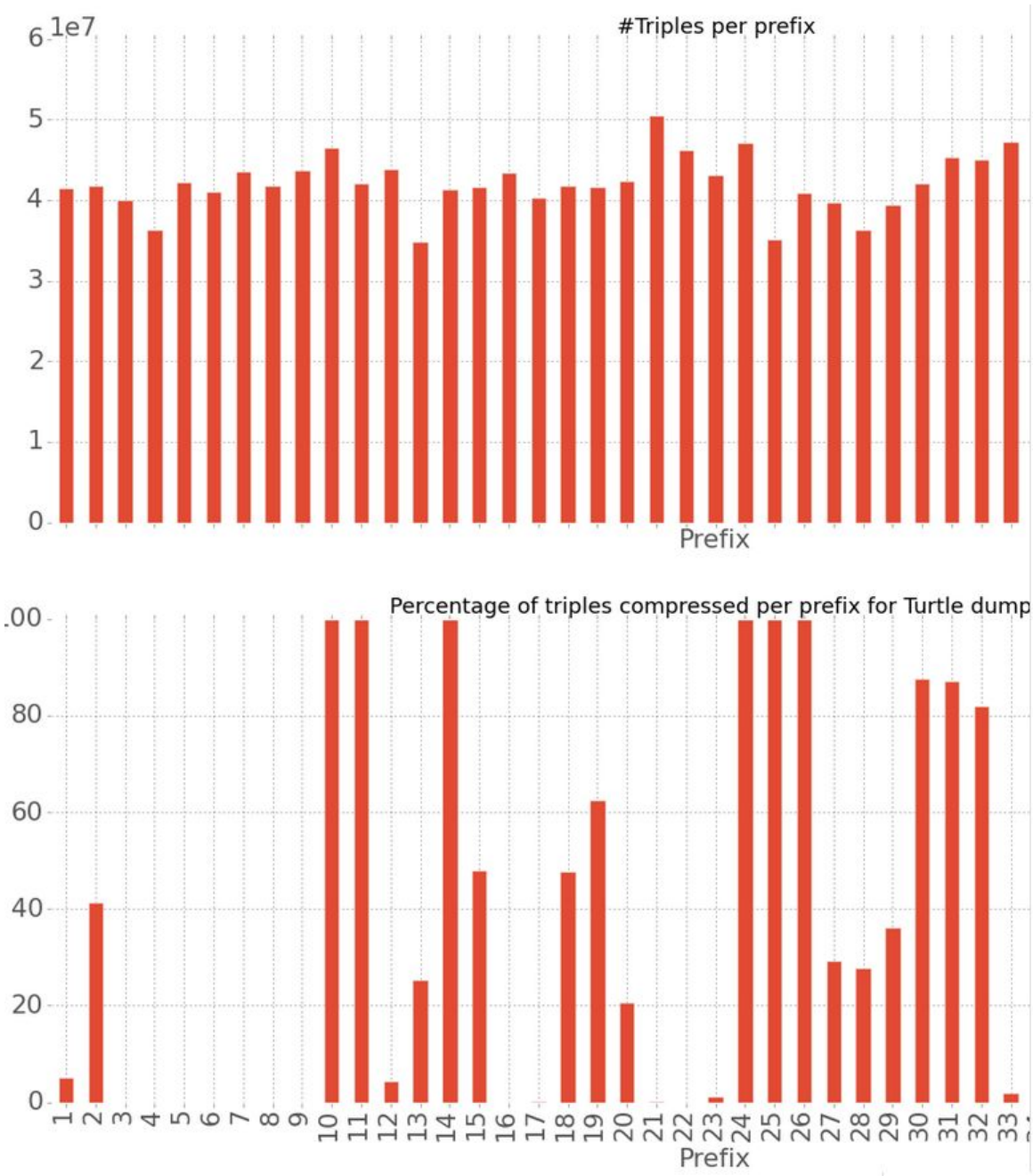


Figure 2: Number of triples per prefix starting from RDF/XML (top) and RDF/Turtle (bottom). RDF/Turtle has too many errors to be an acceptable source for HDT generation.

4.1.3 Query Validation

The Europeana specific queries where provided via a google document. A simple copy-paste of the queries resulted in a set of errors thrown by the benchmarker client. (table is in the wiki page, screenshot below). Therefore we chose to use SPARQLer Query Validator to validate and correct the queries.

Table of errors:

Sparql file	Error	Fix
Query324.sparql	Line 17, column 10: Unresolved prefixed name: dcterms:spatial	replace dcterms: by dct: (wrong prefix)
Query3211.sparql	Encountered " "?Proxy "" at line 17, column 3.	Added a '.' after first triple in BGP
Qiery331b.sparql	Encountered or signs at line 20, column 46	Probably invalid syntax with or sign, rewrote query with UNION clauses
Query42.sparql	Encountered " "?property "" at line 18, column 10.	() for FILTER
Query51.sparql	Line 16, column 18: Unresolved prefixed name: http:	<> around url
Query53.sparql	Line 16, column 21: Unresolved prefixed name: dc-term:issued	prefix dc-term: => dct:

After validation and correction the queries are stored in benchmarker/europeana_queries_DDWParsed/.

4.2 Virtuoso Open Source Edition

4.2.1 Installation & Configuration

Docker image at https://hub.docker.com/r/tenforce/virtuoso/
Details on installation can also be found in the virtuoso wiki page we created.

Container can be installed with the following command:

- **sudo docker run --name virtuoso -p 8890:8890 -p 1111:1111 -e DBA_PASSWORD=dba -e SPARQL_UPDATE=true -e DEFAULT_GRAPH=http://www.example.com/my-graph -v /home/ddewitte/data:/data -d tenforce/virtuoso**
- **sudo docker start/stop virtuoso** can be used to start or shutdown the machine while preserving state.

This creates a shared folder between the docker container and the host. In the host system the directory /home/ddewitte/data corresponds to the internal /data directory. Ports 8890 and 1111 are the SPARQL port and the ISQL port respectively. The other parameters are self explanatory. Note that modifying the port mappings is a simple as modifying the left hand side of the expression: **-p outer_port:container_port**

Alternatively Virtuoso can be build from source as described in the documentation: http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSBuild

The Virtuoso container automatically starts Virtuoso. The modify its configuration the container must be stopped. In the ~/data folder a config file named *virtuoso.ini* can be modified to tune Virtuoso (the file used in the benchmarks can be found in /virtuoso/setup/virtuoso.ini). The following set of parameters should be modified:

- Add the /data folder to the list of allowed dirs, to allow data import from this directory
    - DirsAllowed = ., /usr/local/virtuoso-opensource/share/virtuoso/vad, /data

- Adjust the Virtuoso parameters to take advantage of your server hardware (values extrapolated for 128 GB system, for smaller system just multiply/divide):
    - NumberOfBuffers = 10900000
    - MaxDirtyBuffers = 8000000
- Virtuoso can be optimized to automatically limit the number of results per query or the (estimated) query execution time. These are standard settings which should be adjusted for benchmarking:
    - ResultSetMaxRows = 1000000000
    - MaxQueryCostEstimationTime = 3600
    - MaxQueryExecutionTime = 3600

Note: The virtuoso binaries can be found at **/usr/local/virtuoso-opensource/bin,** the database files, logs,... are in **/usr/local/virtuoso-opensource/var/lib/virtuoso/db**

- **virtuoso.log:** is the starting point for troubleshooting

More information on performance tuning can be found in the Virtuoso documentation: http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtRDFPerformanceTuning

Note after benchmark evaluation: The number of results per query seems to experience an upper threshold in the benchmark. After consulting with OpenLink they recommend commenting out ResultSetMaxRows, they also advise against setting the maximal timings.

4.2.3 Virtuoso Bulk loader

For large datasets such as the Europeana dataset triples are imported in bulk to avoid transactional overhead. Virtuoso has a bulk loader which can be run as follows (from /usr/local/virtuoso-opensource/)

- **sudo bin/isql-v 1111 dba dba exec="ld_dir('/data', '*.ttl', 'http://europeana.eu/');"**
- **sudo bin/isql-v 1111 dba dba exec="rdf_loader_run();"**

To investigate the status of the bulk loading process you can check the state:

- **sudo bin/isql-v 1111 dba dba exec="select * from DB.DBA.load_list;"**

ll_state 1 means ‘in progress’ , 0 is not started, 2 is finished. ll_start and ll_stop indicate the start and end times of the bulk loader ll_error contains potential error messages.

More information on the Virtuoso Bulk loading process can be found in the documentation: http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtBulkRDFLoader

In /virtuoso/setup/ there’s a script `loadTriplesPer1000.sh` which was used to load the RDF/Turtle dump. The bulk loader didn’t have any problems loading the data and didn’t report any issues, this makes the Virtuoso Bulk loader the most robust of the systems tested.

4.2.4 Virtuoso Sparql Endpoint

The Virtuoso SPARQL endpoint can be reached via http://benchmark3.eanadev.org:8890/sparql,

4.3 Blazegraph

For Blazegraph no docker images were available for the newer versions of the software. Therefore a docker file was created and committed to the original docker blazegraph repository on github (version 2.1.2)

The wiki page contains details on how to modify the server configuration properties, run the bulk loader and start Blazegraph.

4.4 Fuseki with HDT back-end

The rdf-hdt homepage describes the integration of jena and fuseki, unfortunately this document is not up to date (broken links). The best source is the github repo, more specifically the hdt-fuseki page: https://github.com/rdfhdt/hdt-java/tree/master/hdt-fuseki

The Fuseki wiki page we provide contains all commands to setup Fuseki. These commands are grouped in the following files:

- installFuseki.sh: install prerequisites (java, maven), download latest github source and build using maven
- ./bin/hdtEndpoint.sh: script to actually start the server with modified memory settings: INDEX_MEM=32g and SERVER_MEM=64g
- loadDataAndStartFuseki.sh: Script used to launch fuseki while logging to fuseki.log. This script uses a custom config file. Note that when the .index files are not present this script will take some time to perform the mapping, this is considered part of the bulk loading process. With these files present starting fuseki only takes about a minute.

- Fuseki_europeana_config.ttl: custom config file to load a set of HDT files into fuseki

Some details about the config file. There is a standard file in the hdt-java repository which we modified:

- Main modification is to use a UnionModel which contains all hdt files and use this model as the default graph (only one default graph can be chosen and dropping the FROM clause in the SPARQL queries has the effect of only query the default graph)
- Using the UnionModel introduced an exception in the loading phase: org.apache.jena.assembler.exceptions.AmbiguousSpecificTypeException: This can be resolved by specifying that a HDT graph is in fact a model by adding the line:

  ```
  hdt:HDTGraph rdfs:subClassOf ja:Model .
  ```

  to the configuration file.

4.5 Neo4J with Sparql plugin

The Neo4J SPARQL plugin was considered an additional store for benchmarking but was left out due to time limitations. It's also the least likely to be adopted due to the fact that the plugin is not frequently updated.

4.6 Sparql Query Benchmarker

On the client[1-5] nodes SPARQL query benchmarker was installed, this is an open source project developed by Rob Vesse. The prerequisites and run details can be found in the Benchmarker wiki page.

Under input specifics the actual commands issued are shown. As can be found in the wiki the benchmark client can be run as follows:

```
./sparql-query-bm-2.1.0/cmd/benchmark
```

- `-m ./queryListEuropeana.txt`
- `-r 10`
- `-o 2`
- `-w 3`
- `-q <http://sparqlendpoint:port>`
- `-p 8`
- `-c benchmarkerOutputVirtuoso.csv`
- `--log-file runnerVirtuoso.log`
- >> outputVirtuosoBenchmark.log

Meaning of parameters:

- m: query list file contains the paths to the files with the queries (one query per file!)
- r: number of query mix runs
- o: number of outliers (omitted in the calculation of averages)
- w: number of warmup runs
- q: url of sparql endpoint
- p: parallel threads
- c: resultsfile (csv)

Login credentials are sometimes required and can be supplied via the following extra flags:

- --username dba
- --password dba

4.7 Issue Tracker

4.7.1 Networking issues

Docker containers did refuse to restart. The error was the following:

- iptables failed: iptables --wait -t nat -A DOCKER -p tcp -d 0/0 --dport 8890 -j DNAT --to-destination 172.17.0.2:8890 ! -i docker0: iptables: No chain/target/match by that name. (exit status 1).

According to Christos the iptables service is governed by service *ferm*. Restarting *ferm* did not result in the docker container to successfully restart but restarting the docker service (after stopping all containers) turned out to be the way to go:

- sudo systemctl restart docker

4.7.2 URI exceptions

```
ERROR: Banner.java:160: Uncaught exception in thread
java.lang.RuntimeException: Could not parse file: /data_mount/data/dump.ttl
        at com.bigdata.rdf.store.DataLoader.loadFiles(DataLoader.java:1378)
        at com.bigdata.rdf.store.DataLoader.main(DataLoader.java:2085)
Caused by: org.openrdf.rio.RDFParseException: IRI included an unencoded space: '32' [line 2228046]
        at org.openrdf.rio.helpers.RDFParserHelper.reportFatalError(RDFParserHelper.java:441)
        at org.openrdf.rio.helpers.RDFParserBase.reportFatalError(RDFParserBase.java:671)
        at org.openrdf.rio.turtle.TurtleParser.reportFatalError(TurtleParser.java:1306)
        at org.openrdf.rio.turtle.TurtleParser.parseURI(TurtleParser.java:947)
        at org.openrdf.rio.turtle.TurtleParser.parseValue(TurtleParser.java:614)
        at org.openrdf.rio.turtle.TurtleParser.parseObject(TurtleParser.java:502)
        at org.openrdf.rio.turtle.TurtleParser.parseObjectList(TurtleParser.java:428)
        at org.openrdf.rio.turtle.TurtleParser.parsePredicateObjectList(TurtleParser.java:421)
        at org.openrdf.rio.turtle.TurtleParser.parseTriples(TurtleParser.java:385)
        at org.openrdf.rio.turtle.TurtleParser.parseStatement(TurtleParser.java:261)
        at org.openrdf.rio.turtle.TurtleParser.parse(TurtleParser.java:216)
        at com.bigdata.rdf.rio.BasicRioLoader.loadRdf2(BasicRioLoader.java:236)
        at com.bigdata.rdf.rio.BasicRioLoader.loadRdf(BasicRioLoader.java:176)
        at com.bigdata.rdf.store.DataLoader.loadData4_ParserErrors_Not_Trapped(DataLoader.java:1595)
        at com.bigdata.rdf.store.DataLoader.loadFiles(DataLoader.java:1359)
```

Figure 2: Triple stores are experiencing problems in loading the turtle dump generated by Europeana. Stacktrace of Blazegraph.

Both Blazegraph and Virtuoso failed to complete the bulk loading process so far. Blazegraph is more explicit in highlighting the cause of the failure as can be seen in figure 2. Doing a status check in Virtuoso revealed that also there the bulk loading process fails due to a uri problem, although the actual error is not very comprehensive, from the virtuoso.log:

- 13:25:22 PL LOG: File /data//dump.ttl error 37000 SP029: TURTLE RDF loader, line 101334038: syntax error processed pending to here.

Fuseki can work with HDT files. The HDT compression software did manage to create the HDT files from the RDF/XML dataset, creating HDT files from the updated RDF/Turtle although recommended didn't work very well due to data quality issues, this will be discussed later on. One of the issues that Europeana tried to tackle was to replace all spaces in URIs by %20. Unfortunately this in turn introduced unexpected replacements which had to be corrected.

5. Benchmark Results

In a research paper by our lab a number of closed source solutions were compared on an artificial benchmark (Watdiv). We'll add these results here to give additional perspective. On top of that we'll provide some results on Europeana specific queries and run on Europeana hardware.

5.1 Results on artificial datasets (outside project)

In a recent paper published by our lab and presented at SIGMOD 2016 a number of closed source triple stores were tested on Cloud commodity hardware. Since Virtuoso Open Source edition and Blazegraph are part of this research effort we also show this results here.

To compare the different storage solution we compare the runtime distributions.

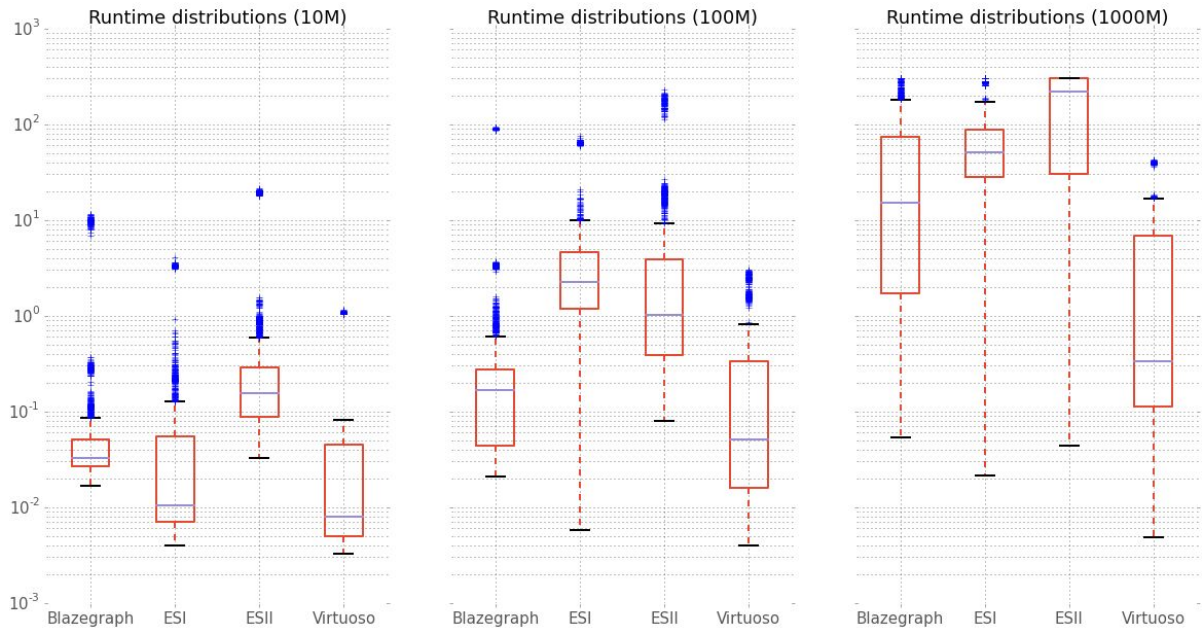


Figure 4: Runtime distribution for different triple stores

These results are obtained by running the triple stores with default configuration on 32 GB servers. Especially this amount of RAM proves problematic for most triple stores, except for Virtuoso. Important nonetheless is the fact that Blazegraph is definitely one of the best triple stores out there. In later benchmarking efforts the gap was slightly narrowed in setups with more RAM. On the other hand performance difference for a full query mix is mainly determined by the tails of the distribution making Virtuoso excel by an order of magnitude on the Watdiv benchmarks.

In order to better predict the performance giving a specific query mix it is instructive to classify the queries according to their structural properties. In figure 4 results are shown for 4 query template types: linear queries, star-shaped queries, snowflakes and more complex queries.

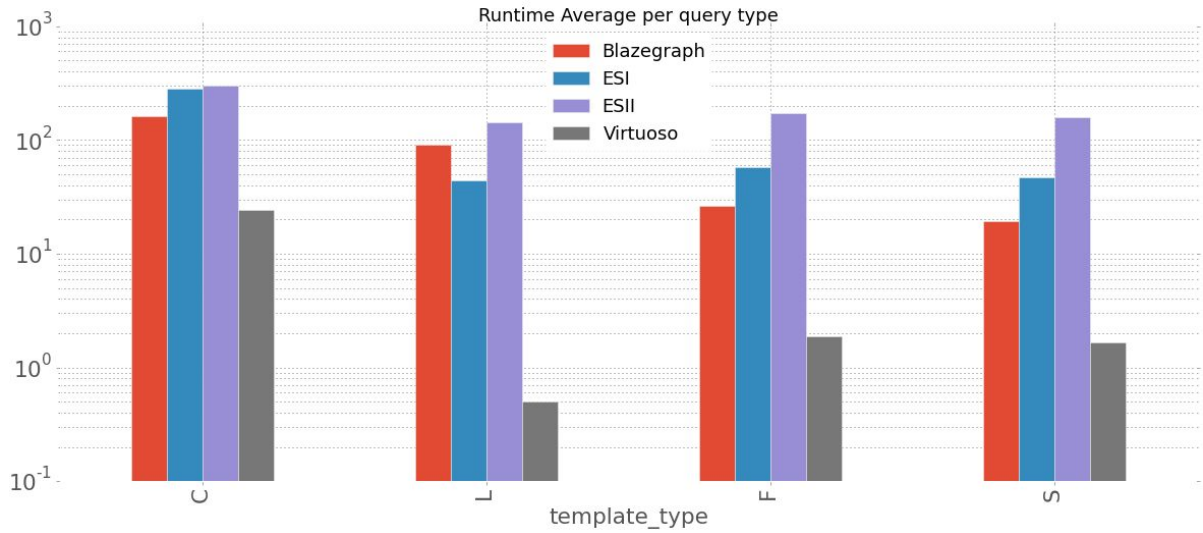


Figure 5: Runtime for different query templates for the different triple stores. L = Linear query, F = Snowflake query, S = star shaped query, C = complex queries. More details on [Watdiv site]

This exercise can be repeated with different classification schemes for example taking into account the different SPARQL keywords used in the queries.

5.2 Results on Europeana triples and queries

5.2.1 Time measurements

In the postprocessing directory of the github directory a number of csv files is collected with analysis on the benchmark data. The content of these csv files is visualized in EuropeanaPostprocessing.x:

- .py: python script
- .ipynb: ipython notebook file (open using ipython notebook filename)
- .html: an export of the notebook file

Using this notebook Europeana can very easily reproduce the results of the benchmark project but on top of that also run additional tests with new queries and even new triple stores.

Below we'll give a summary of the results of the notebook:

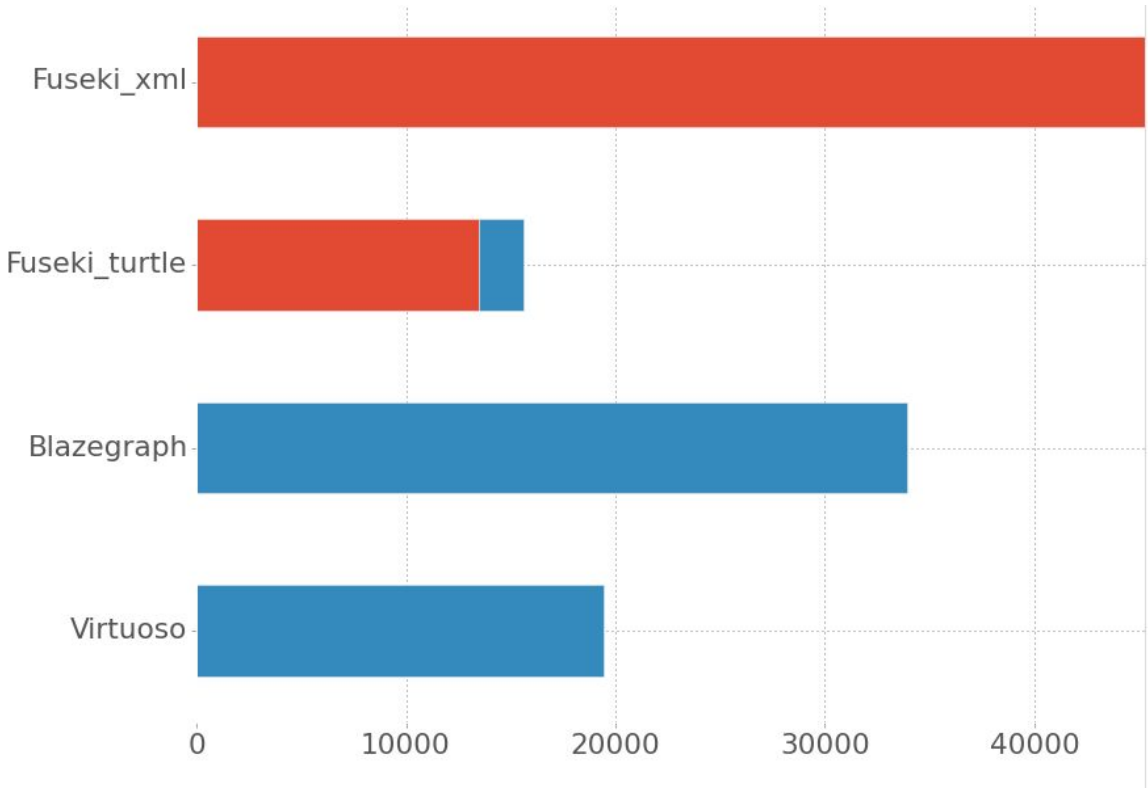


Figure 6: Load times for the different triple stores

In figure 5 we have a look at the load times for the different triple stores. Since Fuseki starts (just as the LDF system in WP1) from a set of HDT files, the creation of these has to be taken into account. Creating HDT files from RDF/Turtle makes a huge difference in the overall load process. The X-axis is in seconds, with Virtuoso loading approximately twice as fast as Blazegraph. Fuseki load time also consists of building index files of the HDT data, this takes approximately one hour and is shown in blue.

As the number of queries is rather low (50) we chose to visualize their runtimes in a line plot instead of using box plots. Figure 7 shows the runtime for the 3 different solutions on a log scaled plot. Note that the timeout of SPARQL benchmarker was set on 300 seconds, it might make sense to rerun the experiments with higher timeouts. On the other hand, as the plot shows Virtuoso doesn't experience any timeouts:

Timeouts:

- Virtuoso: no timeouts
- Blazegraph: 26 timeouts
- Fuseki: 45 timeouts

The actual runtimes of Fuseki are somewhat reflected in fuseki.log but are often in the order of +1000 seconds. This results is probably caused by the fact that the timeout only occurs at the client side while the server keeps processing the queries. We assume that this causes the results to be worse for Fuseki, on the other hand it is a shortcoming of the Fuseki server that it doesn't respond to the client canceling the query.

The good results of Virtuoso will be somewhat put in perspective if we analyze the correctness of the queries later on!

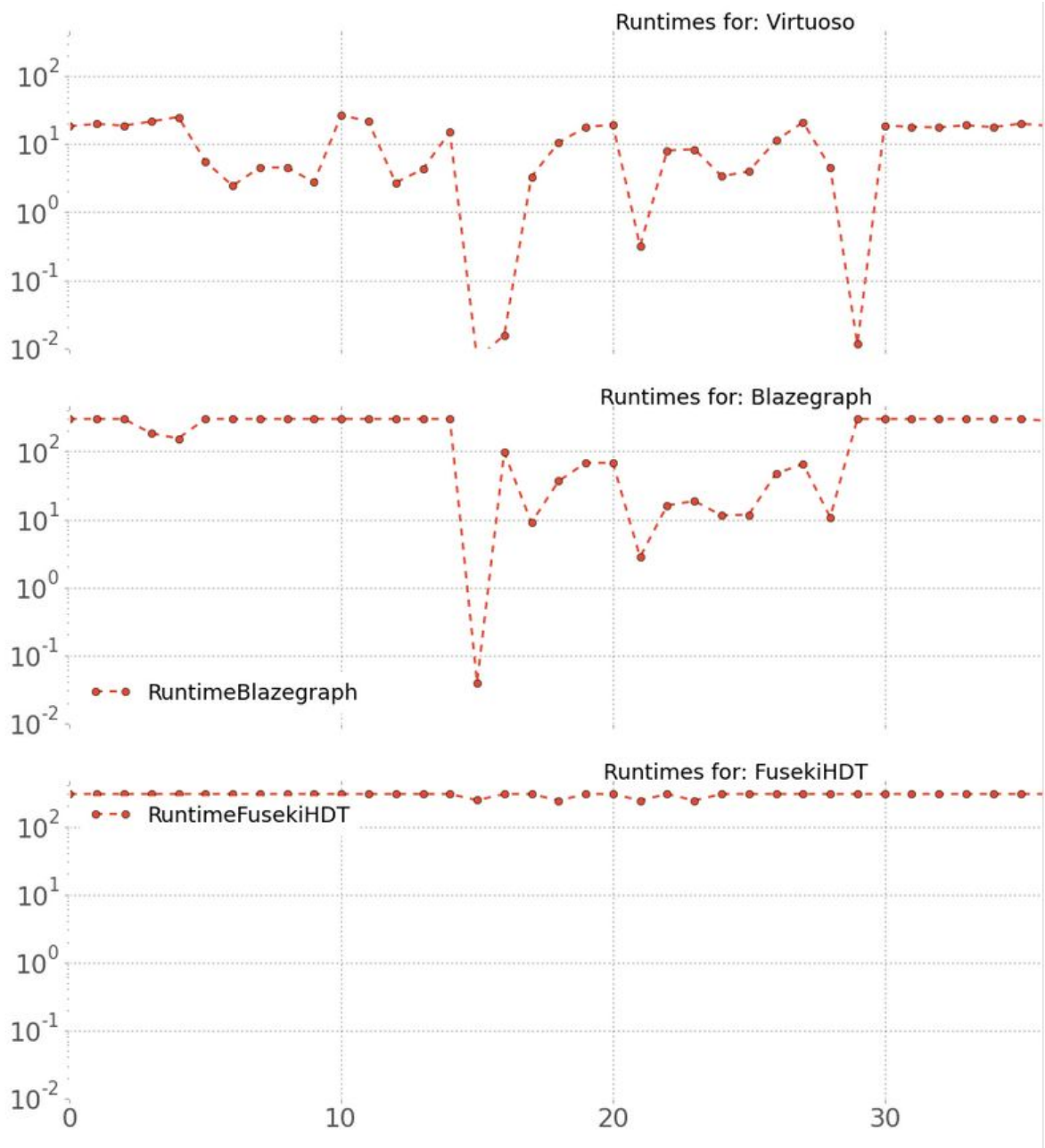


Figure 7: Runtimes for the different triple stores on 50 queries provided by Europeana. From left to right the queries are in the same order as the google document, but this can be verified in the csv files of the different stores.

Runtimes in a box plot are shown in figure 8.

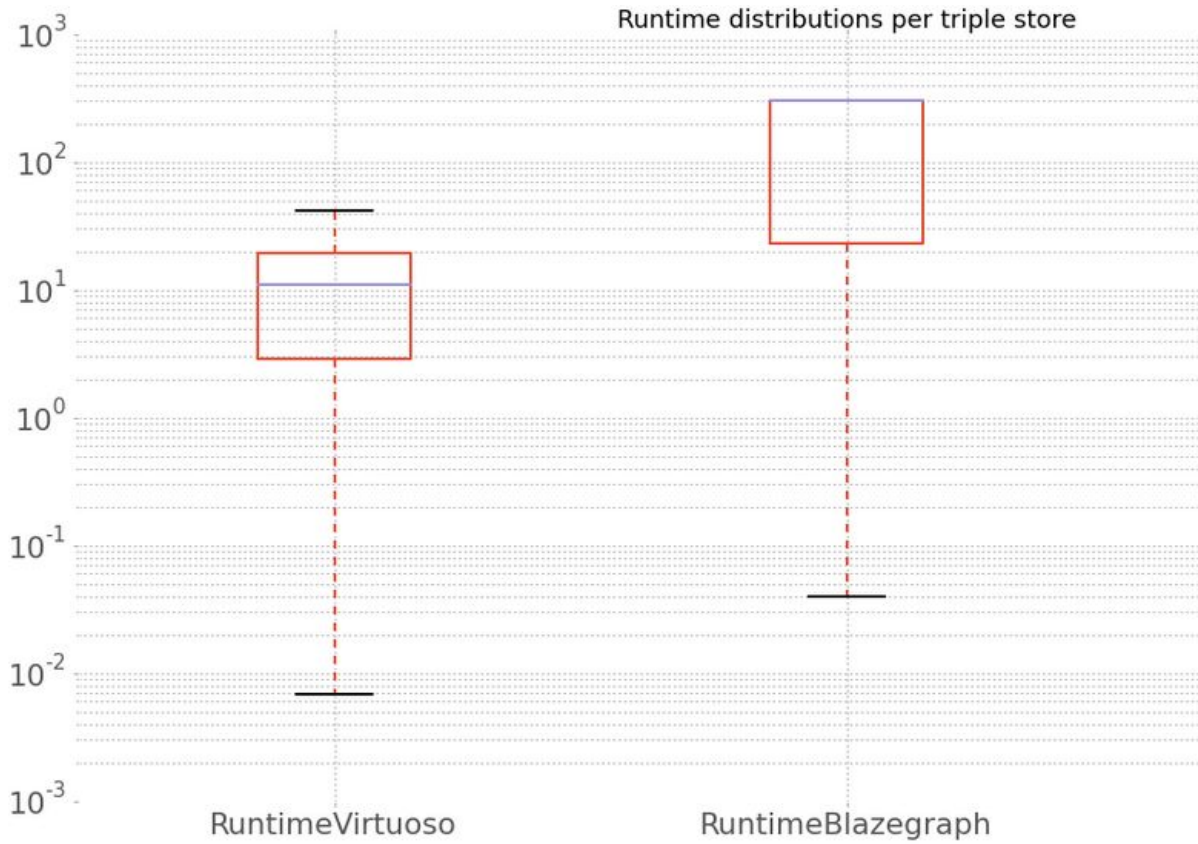


Figure 7: Runtime distribution Virtuoso versus Blazegraph. If we focus on the median results Virtuoso is approximately 30 times faster than Blazegraph. If we add all runtimes together Virtuoso requires 735 seconds to run all 50 queries, Blazegraph needs 9333 seconds which is still a factor 12.7

5.2.2 Query accuracy

The notebooks also provide with 2 visualizations on the number of errors per query and also on the number of results per query. Since the visualization are large we'll stick to the observations in this document.

Important result of counting the number of results per query is the fact that for the queries with many results **Virtuoso seems to limit the number of results** to 1048576 per query. We contacted Open Link Support, removing the ResultSetMaxRows parameter should address this problem (although our value was set to 1 billion). This is worth further investigation by Europeana as it makes it easier for Virtuoso to complete the queries without getting a timeout.

If the actual number of results per query are studied (numberOfResultsPerQuery.csv) we can see that the number is sometimes close but not identical. We hypothesize that this is due to the problems in the loading phase:

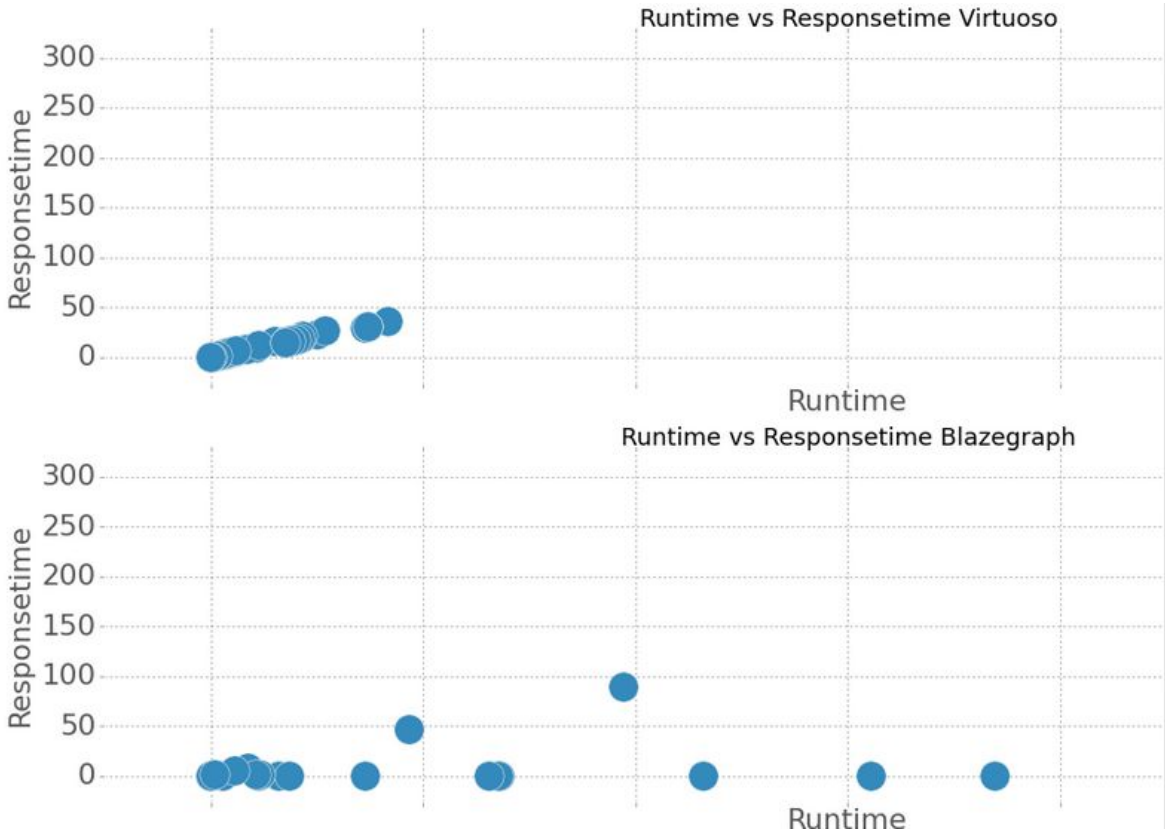
- Fuseki started from the RDF/XML dataset and has 1.9 billion triples
- Virtuoso and Blazegraph started from RDF/Triple dataset but for Blazegraph a number of problematic files were omitted:
    - Virtuoso COUNT(*): 1 592 172 574 triples

- Blazegraph COUNT(*): 1 585 090 298 triples

Studying the percentage of errors per query shows that the results are rather consistent: apart from a small number of exceptions, failure to answer a query is generally consistent across multiple runs.
Not mentioned before is that Virtuoso actually failed to process the queries of section 5 (Query51, 52 and 53) while Blazegraph had no problems with these queries. In the performance measurement these errors are not registered as timeouts but as zero runtime.

5.2.3. Run time versus Response time

If we compare Blazegraph and Virtuoso on final parameter to study is the distinction between query runtime and query response time. In the case of Blazegraph fitting a line to the curve mainly follows the X-axis, meaning that Blazegraph is very soothed for providing (partial) results as soon as they become available while the curve for Virtuoso is actually a bisector showing that Virtuoso doesn't support streaming partial results. But since Virtuoso is also very fast at processing the full query this shouldn't be weighed too heavily in the evaluation.



6. Overview of deliverables for Europeana

Deliverables for Europeana are made available at a private github repository owned by Europeana.

In this github repository there's a directory per component:

- benchmarker: installation files for SPARQL Query benchmarker
- blazegraph, fuseki, virtuoso: installation and measurements per store
- prereq: docker installation scripts
- datapreparation: query validation and HDT generation scripts
- postprocessing: contains scripts for visualization and analysis of the results of the benchmarking

The most useful deliverable is the set of detailed wiki pages:

- In principle the same names are chosen for the wiki pages as for the source code directories
- For the triple stores there is a page on setup in data loading and a page on measurements

7. Recommendations for Europeana

Based on prior experience: Virtuoso and Blazegraph are the most promising players. Given sufficient memory the stores have a performance which is in the same range, although Virtuoso in general is faster. When reducing the available memory Virtuoso is the only store that can guarantee a decent performance.

Based on the specific results for the Europeana dataset and queries it seems that Virtuoso is outperforming Blazegraph still by an order of magnitude. Some results might change as a consequence of some anomalies detected in the benchmark (such as the max #results) and it might be interesting to rerun the benchmark with a higher timeout parameters and perhaps only single threaded but the conclusion of this benchmark is expected to hold.

Based on past experiences with the Triple store vendors we can state that both companies are helpful w.r.t. offering support even when no commercial license is acquired.

With respect to the data format we strongly encourage to improve the data quality of the RDF/Turtle data as this data format has clear advantages as opposed to RDF/XML in terms of performance and preprocessing.

Finally it has become clear that stores with a commercial offering vastly outperform purely open source solutions such as Jena Fuseki. Although this offering might be of some value in further evaluation the LDF system in WP1.

As a final note we'd also like to mention that evaluating the runtime of queries in a triple store is not the only way to benchmark systems and Europeana should definitely weigh in different aspects, such as reasoning capabilities, fulltext search support,... For this matter having a look at the documents provided in section 3.2 is recommended. We do advise against relying on the score provided by Wikidata since the rating of Virtuoso doesn't seem to be the result of thorough testing and analysis.