Trying Not to Die Benchmarking – Orchestrating RDF and Graph Data Management Solution Benchmarks Using LITMUS

Harsh Thakkar
University of Bonn
Bonn, Germany 53117
hthakkar@uni-bonn.de

Yashwant Keswani
DA-IICT
Gandhinagar, India 382007
201301047@daiict.ac.in

Mohnish Dubey
University of Bonn
Bonn, Germany 53117
dubey@cs.uni-bonn.de

Jens Lehmann
University of Bonn
Bonn, Germany 53117
jens.lehmann@cs.uni-bonn.de

Sören Auer
University of Bonn
Bonn, Germany 53117
auer@cs.uni-bonn.de

ABSTRACT

Knowledge graphs, usually modelled via RDF or property graphs, have gained importance over the past decade. In order to decide which Data Management Solution (DMS) performs best for specific query loads over a knowledge graph, it is required to perform benchmarks. Benchmarking is an extremely tedious task demanding repetitive manual effort, therefore it is advantageous to automate the whole process. However, there is currently no benchmarking framework which supports benchmarking and comparing diverse DMSs for both RDF and property graph DMS. To this end, we introduce, the first working prototype of, LITMUS which provides this functionality as well as fine-grained environment configuration options, a comprehensive set of DMS and CPU-specific key performance indicators and a quick analytical support via custom visualization (i.e. plots) for the benchmarked DMSs.

CCS CONCEPTS

•Information systems →Graph-based database models; Database management system engines; Database performance evaluation; Resource Description Framework (RDF); *Extraction, transformation and loading;* Query representation; **•Theory of computation →***Database query languages (principles);*

KEYWORDS

Benchmarking, Linked Data, Performance Analysis, RDF Stores, Graph Stores, Automated Framework

ACM Reference format:
Harsh Thakkar, Yashwant Keswani, Mohnish Dubey, Jens Lehmann, and Sören Auer. 2017. Trying Not to Die Benchmarking – Orchestrating RDF and Graph Data Management Solution Benchmarks Using LITMUS. In *Proceedings of Semantics2017, Amsterdam, Netherlands, September 11–14, 2017,* 8 pages.
DOI: 10.1145/3132218.3132232

Semantics2017, Amsterdam, Netherlands

DOI: 10.1145/3132218.3132232

1 INTRODUCTION

Over the last few years, the amount and availability of Open, Linked and Big data on the web has increased. Simultaneously, there has been an emergence of a number of Data Management Solutions (DMSs) to deal with the increased amounts of structured data. The available DMSs for graph structured data can be broadly divided in two categories on the basis of the data model they address: 1.) *Triple Stores*, which employ the RDF Graph data model and 2.) *Graph Databases*, which frequently use the Property Graph (PG) data model. Apart from the format of the dataset that they consume, there are several other differences in the manner in which they build indexes and execute queries.

In order to objectively decide which DMS are suitable with respect to particular scenarios, benchmarks involving particular query loads over characteristic datasets have been defined. Some of the existing benchmarking tools have their own dataset generators and corresponding queries to run on these datasets [1, 4]. However, none of the tools allow the users to benchmark both of the above mentioned categories of graph DMSs, i.e. triple stores and graph databases, in a unified and comparable manner. We argue that the the increasing number of available DMSs in both categories necessitates a benchmarking tool which is sufficiently versatile to perform those benchmarks.

To this end, we present LITMUS – an open, and a extensible framework, which allows the users to benchmark DMS for RDF and property graph data models on a given dataset and query workload. To enable this, the dataset will be presented as RDF dataset to triple stores and as property graph to graph databases. The dataset is then queried using queries written in SPARQL [15] for the RDF DMSs and Gremlin [16] for the Graph DMSs.

LITMUS is a comprehensive framework, that will serve the academicians, researchers, DMS developers and the organizations who employ DMSs for the efficient consumption of their data, by allowing a choke-point driven performance comparison and analysis of various Data Management Solutions (Graph and RDF-based), with respect to different third-party real and synthetic datasets and queries.

Contributions: (1) To the best of our knowledge, LITMUS is the first framework able to – **support both RDF and PG benchmarking (by SPARQL querying of property graphs).**

Moreover, LITMUS provides benefits, which are partially present in other benchmarking frameworks but not in their combination. In particular, LITMUS is able to:

(2) promote reusability via providing a unified open extensible architecture for orchestrating user-driven benchmarks
(3) provide a list of comprehensive CPU and memory-based metrics and parameters for performance evaluation,
(4) offer full automation of the underlying tedious sub-tasks, and
(5) support a comprehensive post-benchmark performance reporting via custom visualization using tables and plots.

Limitations:
At present, LITMUS does *not* support the following features:

- Federated querying – Multiple sources/endpoints being queried at the same time.
- Parallel querying & updates – Multiple users/clients querying and updating a single source at the same time.
- Support for generation of synthetic datasets and queries. We "use" other benchmarks that offer such support and leave it up to the user to chose what they deem fit according to their needs.

The rest of the article starts with Section 2 reporting the state-of-the-art in benchmarking, followed by Section 3 portraying the functional architecture of LITMUS. Section 4 elaborates the curated LITMUS environment and infrastructure, followed by presenting the selection of KPIs considered for DMS evaluation in Section 5. Finally, Sections 6 and 7 illustrate the proposed framework in action and conclude the paper, presenting future directions.

2 RELATED WORK

We summarize the current state of the art in benchmarking with respect to (a) relational databases, (b) graph databases, (c) RDF stores, and (d) cross-domain benchmarking efforts in Table 1.

The existing benchmarks, as shown in this table, have various domain-specific strengths. However, they also display limitations regarding the need of having an integrated generalized benchmarking framework. The existing efforts, for instance, (i) do not offer the capability of benchmarking both RDF and property graphs in a single environment; (ii) with the exception of HOBBIT, do not offer an end-to-end benchmarking and result visualization solution of cross domain DMSs; and (iii) do not allow easy integration of existing benchmarks in an user-driven fashion.

LITMUS addresses the above mentioned shortcomings and serves as an end-to-end benchmarking solution with the capability of bencharmking RDF and Property graphs. It promotes interoperability, reusability and replicability of existing benchmarks via visualization of benchmark results, all wrapped in one open, extensible framework.

3 THE LITMUS FRAMEWORK

The proposed LITMUS framework consists of a number of plug-n-play modules, which ensures interoperability and extensibility of future DMSs and datasets in the existing infrastructure. We describe the first prototype of LITMUS Benchmark Suite in Figure 1, which showcases the interaction between various constituent modules. A detailed description of the framework, in its final stage, can be referred from [21]. We now summarize the function of each module in LITMUS:

GUI module: provides a graphical interface for the user to allow easy configuration of the benchmark to be executed. It allows the user to select from the integrated DMSs, datasets, queries and KPIs, and save, import, export configurations, results, etc. artifacts produced during the benchmark. Figure 2 shows a screen of the GUI module.

Dataset module: consists of various scripts for loading the user selected datasets in corresponding DMSs. It also converts a given RDF dataset in to property graphs if a converted version is not available. For the integrated datasets, we already provide the converted version within.

DMS modules: consists of various scripts for loading user-specified configuration to the corresponding DMSs and preparing them for a benchmark.

Controller & Tester module: consists of various scripts for executing the benchmark by – (i) preparing the DMSs for warm-cache or cold-cache settings by warming up the DMSs (if specified); then (ii) executing respective SPARQL and Gremlin queries against RDF and Graph DMSs in a controlled fashion.

Analysis & Visualization module: collects the results (log files) from the executed benchmark of both RDF and Graph DMSs. The `matplotlib`[1] python library is used for generating box-plots of benchmark results, where as statistical analysis is carried out using the the `Pandas` python library, to calculate various parameters, e.g. arithmetic mean, median, standard deviation etc.

LITMUS docker: The whole framework is encapsulated in a single configurable docker container, to ensure necessary isolation during the benchmarking process.

Query module: In the current version of LITMUS, we do not have a separate query module, since we provide limited SPARQL queries and their Gremlin translation. This is to be addressed in the next version, which will provide on-the-fly query translation using "Gremlinator". In a nutshell, this translation requires mapping SPARQL algebra operators to corresponding Gremlin operators as proposed in [22], and there by constructing corresponding graph pattern matching Gremlin traversals.

The complete source is well documented and made available publicly[2]. The first prototype of LITMUS framework (v0.1) is released on Docker Hub platform[3] for encouraging first hand experience and user feedback.

4 THE LITMUS ENVIRONMENT

We now describe in brief the cultivated environment for LITMUS, consisting of datasets, DMSs, queries and the benchmark environment configuration supported in its current (v0.1) release.

4.1 Integrated Datasets

LITMUS framework currently provides support for benchmarking RDF DMSs (which are RDF stores), and Graph DMSs (which are Graph stores) using corresponding versions of Linked data (in RDF and Graph formats) against a set of corresponding queries (SPARQL

[1] matplotlib – https://matplotlib.org/
[2] LITMUS Benchmark Suite – https://github.com/LITMUS-Benchmark-Suite/
[3] LITMUS docker – https://hub.docker.com/r/litmusbenchmarksuite/litmus/

Table 1: A survey of the state-of-of-the-art in benchmarking efforts of Relational, RDF and Graph DMSs.

Type	Benchmark	RDB.	RDF	Graph	Description
Single-domain	TPC [13]	✓			The Transaction Processing Performance Council (**TPC**) [13] is well established for benchmarking relational DMSs. TPC provides a range of benchmarks such as the online transaction processing benchmarks **TPC-C** and **TPC-E** (which employ transactions per minute metric), the analytics **TPC-H** and decision support **TPC-DS** (which employ the queries per hour and cost per performance metrics).
	XGDBench [6]			✓	is a graph DMS benchmarking platform for cloud computing systems, which is an extension of the famous Yahoo! Cloud Serving Benchmark to the graph domain. The authors benchmarked graph DMSs – AllegroGraph, Fuseki, Neo4j, an OrientDB using XGDBench on Tsubame 2.0 HPC cloud environment.
	HPC [7]			✓	The **HPC** Scalable Graph Analysis Benchmark consists of a range of tests for examining a variety of independent attributes of the hardware of High Performance Computing systems. HPC addresses graph-specific tasks such as graph suitability transformations and graph analysis over graph DMSs in a distributed environment.
	Graph500 [12]			✓	is a benchmark for data intensive supercomputing systems and its applications. It does not consider benchmarking typical graph databases.
	DBPSB [11]		✓		The DBpedia SPARQL Benchmark **(DBPSB)** benchmarks RDF DMSs using DBpedia by creating a query workload derived from the DBpedia query logs.
	LUBM [9]		✓		The Lehigh University Benchmark (**LUBM**) benchmarks RDF DMSs over a large synthetic dataset that complies to a university domain ontology.
	WatDiv [1]		✓		The Waterloo SPARQL Diversity TEST Suite (**WatDiv**) benchmarks RDF DMSs using their synthetic data and query generators in order to analyze the corelation between DMS performance against varying query structures and complexities (query typology).
	SP2Bench [20]		✓		is one of the most commonly used synthetic data-based RDF DMS benchmarks, which uses the schema of the DBLP bibliographic dataset (http://dblp.uni-trier.de/db/) to generate custom sized datasets.
	IGUANA [19]		✓		is a generic SPARQL benchmark execution framework focused on benchmarking RDF DMSs and federated querying.
	FEASIBLE [18]		✓		is a feature-based (data-driven and structural) SPARQL benchmarking framework for RDF DMSs. It employs an automatic approach for the generation of benchmarks using query logs.
	LSQ [17]		✓		consists of real world SPARQL queries extracted from the logs of public SPARQL endpoints. These queries are extracted from four public endpoints: DBpedia (logs 232 million triples), Linked GeoData (LGD) (1 billion triples), Semantic Web Dog Food (SWDF) (300 thousand triples) and the British Museum (BM) 1.4 million triples).
	HOBBIT [14]		✓		is an end-to-end benchmarking platform (in early stage) focused towards large-scale benchmarking on all aspects of the Linked Data life cycle. It will enable data, query and task generation functionalities (in later stages) for benchmarking of RDF DMSs under custom stress loads for the querying of RDF graphs using industrial use-case queries.
Cross-domain	BSBM [4]	✓	✓		The Berlin SPARQL Benchmark (**BSBM**) is a synthetic data-based e-commerce use case scenario for benchmarking RDF and Relational DMSs. It provides custom generators for creating datasets and queries of custom size and typology.
	Pandora	✓	✓		Pandora (http://pandora.ldc.usb.ve/) is a benchmark which uses the BSBM data to assess the performance of RDF stores against relational stores (i.e. Jena-TDB, MonetDB, GH-RDF-3X, PostgreSQL, 4Store).
	Quertzal-RDF [5]	✓	✓		is a RDF and Graph DMS benchmarking framework, which offers a novel SPARQL to SQL translation engine for multiple backends. Its current version supports benchmarking DB2, PostgreSQL and Apache Spark. It offers custom query loads for both DBpedia (real) and LUBM (synthetic) datasets.
	Graphium [8]		✓	✓	Is a benchmarking plus result visualization effort, comparing RDF stores against Graph stores (i.e. Neo4J, Sparksee/DEX, HypergraphDB, RDF-3X) on custom graph datasets including a 10M triple dataset (using the BSBM data generator).
	LDBC [2]		✓	✓	The Linked Data Benchmark Council (**LDBC**) is focused on curating industry-strength benchmarks for both graph and RDF DMSs. It introduces a choke-point driven analysis methodology for analyzing and developing benchmark workloads.

and Gremlin). We list the datasets that were converted from the RDF graphs to Property graphs (PGs), to ensure a uniform and fair benchmarking process.

In the current version of LITMUS, we do not consider the semantics of blank RDF nodes (as in DBpedia and Wikidata) during the conversion of RDF graphs to PGs. Addressing these underlying semantics of RDF graphs, requires an in-depth study of information preserving techniques. We provide a proof-of-concept implementation for transforming RDF graphs (in this case directed edge-labelled

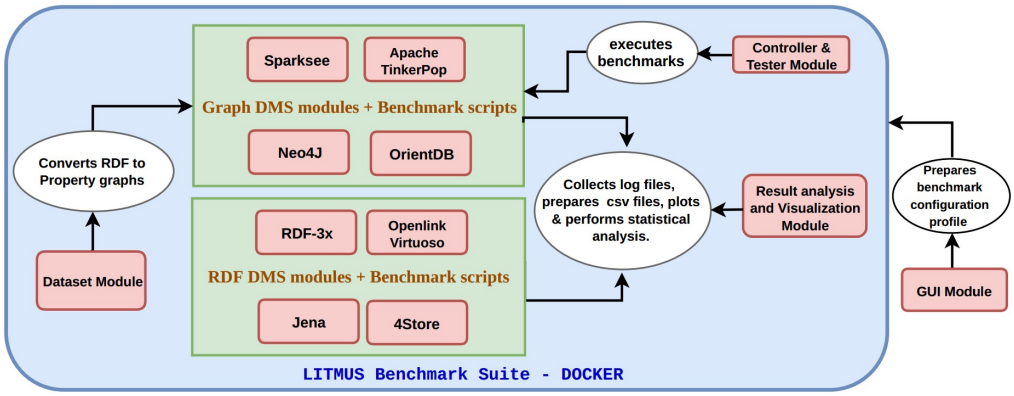


Figure 1: The architecture of the first working prototype of LITMUS Benchmark Suite.

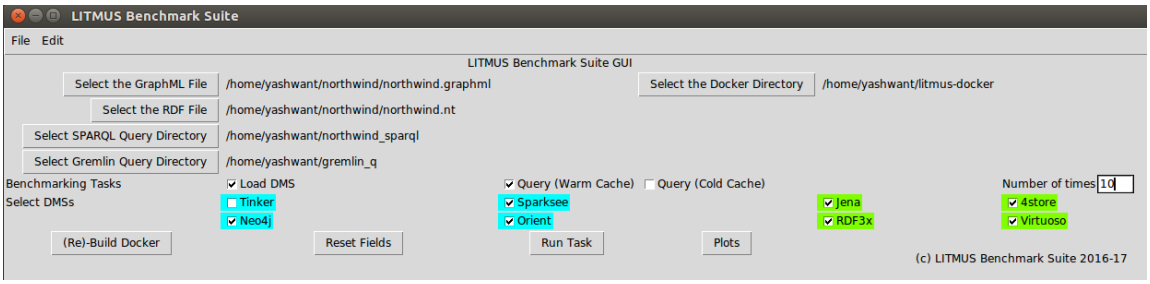


Figure 2: GUI of LITMUS Benchmark Suite.

multi-graphs) [.nt files] to PGs (directed, edge-labeled, attributed multi-graphs) [.graphml files].

We present a set of rules used for the conversion of RDF graphs to PGs. Given that a RDF triple consists of {s p o.}, each of the S, P are IRIs and O can be either a IRI or a literal/value. We distinguish between two types of RDF triples as: (i) attribute triple– if the object is a literal; (ii) relationship triple– if it is a URI. Attribute triples correspond to properties in a PG, and relationship triples to edges. Furthermore, predicates { s p o } can be URI, which can be labels (rdfs:label), types (rdfs:type), etc in a RDF graph. Depending on the type of a predicate, we distinguish whether the properties are of edges or nodes in a PG. We point the interested reader to [10], for a detailed understanding and illustration of the RDF → PG transformation.

Berlin SPARQL Benchmark [4] (BSBM) – is a synthetic dataset built around an e-commerce use case, where a set of products is offered by different vendors and different consumers have posted reviews about products. BSBM offers custom dataset and query generator scripts, which can be used to generate datasets and queries of varying size and complexity. We provide generated RDF data (.nt file) and converted PGs (.graphml file) (using custom scripts) for 1M and 10M triples with the v0.1 of LITMUS.

Waterloo SPARQL Diversity Test Suite [1] (WatDiv) – is a synthetic dataset which is again based on the e-commerce use case scenario, however the distinct characteristic that all instances of the same entity have mixed number of of attributes. We provide generated RDF data (.nt files, using its data generation script) and converted PGs (.graphml files) (using custom scripts) for 1M and 10M triples with the v0.1 of LITMUS.

DBpedia [3] – is a crowd-sourced community effort to extract structured information from Wikipedia and make this information available on the Web. The DBpedia dump consists of multiple files in the *ttl* format. We provide a proof-of-concept property graph of DBpedia. However, we do not benchmark it since it consists blank node semantics, which are not currently supported by LITMUS v0.1 framework. The script developed for the conversion can be found here[4].

Northwind[5] – is a synthetic dataset describing an ecommerce scenario about the sales and purchase transactions that happen between the company Northwind Traders and its customers and suppliers respectively. We provide both RDF data (.nt files) and converted PGs (.graphml files) with the v0.1 of LITMUS.

4.2 Integrated DMSs

LITMUS currently provides support for benchmarking eight DMSs (four each of RDF & Graph DMS), as listed below:

RDF DMSs The following RDF DMS can be evaluated in the Litmus framework:

(1) Openlink Virtuoso[6]
(2) gh-RDF-3x[7]

[4]DBpedia Property graph converter https://github.com/LITMUS-Benchmark-Suite/dbpedia-graph-convertor
[5]Northwind Database https://northwinddatabase.codeplex.com/
[6]Openlink Virtuoso – https://virtuoso.openlinksw.com/
[7]RDF-3X – https://github.com/gh-rdf3x/gh-rdf3x

Table 2: Query feature design and description

Query No.	Feature	Count	Description
C1-C3	CGPs	3	Queries with mixed number of BGPs
F1-F3	FILTER	3	CGPs with a combination of >=1 FILTER
L1-L3	LIMIT+OFFSET	3	CGPs with a combination of >=1 LIMIT constraints
G1-G3	GROUP BY	3	CGPs with GROUP BY feature
Gc1-Gc3	GROUP COUNT	3	CGPs with GROUP BY + COUNT
O1-O3	ORDER BY	3	CGPs with ORDER BY feature
U1-U3	UNION	3	CGPs with UNION feature
Op1-Op3	OPTIONAL	3	CGPs with a OPTIONAL BGPs
M1-M3	MIX	3	CGPs with a combination of varying features
S1-S3	STAR	3	CGPs forming a STAR shape execution plan
TOTAL	**10**	**30**	-

(3) Apache Jena TDB[8]
(4) 4Store[9]

For each of the RDF DMS, LITMUS includes two shell scripts for – (i) benchmarking the process of loading a RDF dataset in a RDF DMS; and (ii) benchmarking the SPARQL query execution process against a DMS. We employ the *elapsed time* parameter of the `'/usr/bin/time'` utility has been used to measure the execution time for both the tasks for all the RDF DMSs. **Graph DMSs** The following Graph DMS can be benchmarked in the current release of LITMUS:

(1) Sparksee (formerly known as DEX Graph)[10]
(2) Neo4j[11]
(3) OrientDB[12]
(4) Apache TinkerPop[13]

For each Graph DMS, LITMUS includes four scripts (2 shell scripts, 2 groovy scripts) which are used for – (i) benchmarking the process of loading a Graph dataset in a Graph DMS, using the Gremlin Groovy console; and (ii) benchmarking the Gremlin Query execution against a Graph DMS, using the Gremlin Groovy console. The execution time for both the tasks has been measured using the `'System.currentTimemillisecs()'` Groovy function.

4.3 Supported Queries

To demonstrate a benchmark using LITMUS, we curated a query dataset including both SPARQL and Gremlin queries following the query features as summarized in Table 2. A total number of 30 SPARQL queries were created (3 of each query feature) for each RDF dataset. We created their corresponding Gremlin counterparts manually for each Graph dataset. The queries are executed using both warm and cold cache settings, where a *warm cache*: implies that the cache is not cleared after each query run, and *cold cache*: implies that the cache is cleared using the `'echo 3 > /proc/sys/vm/drop_caches'` unix command after each query. Running the queries in two configurations allow the users to study the correlation between performance of the DMSs with respect to query, dataset-specific characteristics, and the order in which they are run. Additionally, the influence of factors like the query length, query size, Graph patterns on the performance of the system can be seen when run in the queries are run in warm cache configuration.

[8]Apache Jena – https://Jena.apache.org/
[9]4Store DMS – http://www.4store.org/
[10]Sparksee Technologies – http://www.sparsity-technologies.com/
[11]NEO4J – https://neo4j.com/
[12]Orient DB – http://orientdb.com/
[13]Apache TinkerPop – http://tinkerpop.apache.org/

4.4 Execution environment

It is very important to ensure that all DMSs run under identical conditions for eliminating any bias towards a specific run, and avoiding any inconsistencies and anomalies observed in results. As a result the following set of rules are followed to ensure a fair evaluation procedure.

(1) Each query execution task is carried out individually and is ran several times (default: 10 times, user can define this before-hand) for each DMS to nullify the effect of anomalies.
(2) Every run of the task is run in isolation. No other unnecessary process(es) is running in the background during the benchmark.
(3) Each dataset loading task makes use of a new location for every run. This ensures that no run is getting an undue advantage of an already existing set of files.

5 PERFORMANCE EVALUATION

LITMUS caters a wide variety of performance evaluation parameters and metrics to allow an in-depth analysis of underlying internal and external factors of a DMS.

5.1 Selected Parameters

Perf-tool utility. LITMUS uses the `perf-tool`[14] utility to measure a variety of CPU and RAM-specific parameters, e.g. `L1d`-cache-misses, `L1i`-cache-misses, `DTLB`-misses, etc. for enabling a comprehensive analysis of the participating DMSs. We segregate the parameters offered by the `perf-tool` utility into four groups. Both the benchmarking tasks, viz. (i) loading the dataset a DMS, and (ii) executing a query on a DMS, are run separately for each group of parameters.

These parameters also enable the users to identify the reason(s) for a superior or inferior performance of any particular DMS. We present an itemization of the parameters considered to evaluate a performance of a DMS:

(1) *Cycles* : The number of cycles taken to execute a task (e.g. loading a dataset, etc.).
(2) *Instructions* : The number of instructions executed per given task.
(3) *Cache references* : The total number of cache references made during a given task.
(4) *Cache misses* : The total number of cache misses occurred during a given task.
(5) *Bus cycles* : The number of bus cycles taken during a given task.
(6) `L1` *data cache loads* : The total number of L1 cache loads that occur during a given task.
(7) `L1` *data cache load misses* : The total number of L1 data cache load misses that occur during a given task.
(8) `L1` *data cache stores* : The L1 data cache stores that occur-during a given task.
(9) `dTLB` *loads* : The data translation lookaside buffer (dTLB) loads that occur during a given task.
(10) `dTLB` *load misses* : The dTLB loads misses that occur during a given task.

[14]Perf tool – https://perf.wiki.kernel.org/index.php/Main_Page

(11) LLC *loads* : The Last level Cache (LLC) loads that occur during a given task.
(12) LLC *load misses* : The LLC load misses that occur during a given task.
(13) LLC *stores* : The LLC stores that occur during a given task.
(14) *Branches* : The total number of branches that are encountered during a given task.
(15) *Branch misses* : The total number of branches missed during a given task.
(16) *Context switches* : The total context switches that happen during a given task.
(17) *CPU migrations* : The CPU migrations that occur during a given task.
(18) *Page faults* : The page faults that occur during a given task.

5.2 Selected Metrics

Apart from the dataset loading time and query execution time (both warm and cold caches) for each DMS and each query, LITMUS provides a list statistical metrics for result aggregation and analysis. We provide support for computing various **mean** [μ] (e.g arithmetic, harmonic [H_M] and geometric [G_M]), **median** [$\tilde{x}$], **standard deviation** [σ_x], **variance** [σ^2], **minimum** [$min(x)$] and **maximum** [$max(x)$] for all of the above mentioned CPU and memory-specific parameters using the `pandas`[15] python data analysis library. Furthermore, we also provide functionality to export all the metrics results, in CSV file (comma separated value) format and LaTeX-tabular format.

5.3 Data Visualization

LITMUS provides automated support for visualizing results of the benchmark using the python `matplotlib` data visualization library in the form of boxplots to ease the process of decision making. The boxplot presents the median value, first quartile, third quartile and the extreme outliers. The Inter Quartile Range(IQR) is defined as the difference of the value at the third quartile and the first quartile. An extreme outlier, is defined as a value which is not in the range (*first quartile* $-1.5 * IQR$, *third quartile* $+1.5 * IQR$). These extreme outliers correspond to the anomalous runs which were executed. We use a two color coding scheme to highlight the difference between RDF DMSs (using green) and Graph DMSs (using blue). A distinct plot is generated for each parameter (as mentioned in the above sections), used for each task per DMS.

6 LITMUS IN ACTION

We demonstrate the working of LITMUS to showcase its applicability, functionality and suitability for conducting benchmarks in a user-configured fashion. Keeping in mind the page limit and the extensive amount of results and plots generated during the benchmark, we list only subset of the complete benchmark results (which includes benchmarking only a few parameters, queries and tasks). We only present the results of benchmarking all DMSs using the **Northwind** dataset for brevity. However, a complete set of results can be accessed online via the links provided later in this section. **Benchmarking Tasks**: (i) Dataset loading time; and (ii) Query execution (both Warm Cache & Cold Cache) time

[15]Pandas Data Analysis Library – http://pandas.pydata.org/

6.1 Experimental Setup

We curated the following configuration for executing LITMUS: **CPU**: Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz; **RAM**: 8 GB DDR3; **L1d & L1i Caches**: 32 KB; **L2 Cache**: 256 KB; **L3 Cache**: 3072 KB; **RDF DMSs**: Openlink Virtuoso [7.2.5], Apache Jena TDB [3.2.0] , 4store [1.1.5], gh-RDF3X; **Graph DMSs**: Apache TinkerPop [3.2.4], Neo4J [1.9.6], Sparksee [5.1], OrientDB [2.1.3]

6.2 Preliminary Results

We now present selected plots generated by LITMUS after the benchmarking process. A complete set of results (including all plots, CSV files, and tables in LaTeXformat) for the executed benchmark are made public and can be found here[16].

Table 3 presents the **loading time** performance comparison of loading Northwind dataset for all DMSs respectively. Here, in terms of dataset loading time, we observe that Virtuoso is the fastest followed by TinkerPop (Tinker) and the slowest reported time is by Jena.

Furthermore, Tables 4 and 5 present the **execution time** (both cold and warm cache) performance comparison on *Query 14* (cf. Appendix A, SPARQL listing 1, Gremlin listing 2) for all DMSs respectively. Here, we observe that (for query 14): (i) For *warm cache*- RDF3X is the fastest in terms of query execution time, followed by Virtuoso, whereas TinkerPop (tinker) is the slowest; and (ii) For *cold cache*- Virtuoso is the fastest in terms of query execution time, followed by Neo4j, whereas 4Store is the slowest.

The better performance of Virtuoso and RDF3X (RDF DMSs) in terms of query execution can be traced back to the fact that they both inherently maintain implicit indices. The default indexing scheme[17] in Virtuoso enables it to declare 2 full indices (PSOG, POGS) and 3 partial indices (SP, OP GS) over the RDF graphs. Whereas RDF3X maintains 6 hash-based indices (SPO, POS, OPS, PSO, OSP, SOP) over the RDF graphs giving them an upper edge in terms of performance. In case of TinkerPop (Graph DMS), these indices have to be declared explicitly by the user, depending on their need. Since, we did not explicitly declare any index, hence weaker performance is observed.

Figure 3 presents sample plots of CPU migrations for both cold and warm caches, page-faults and number of instructions executed, for Query 20 (cf. Appendix A, SPARQL listing 3 and Gremlin listing 4). By doing so we demonstrate the versatility of LITMUS in terms of generating plots of varying details of selected KPIs.

7 CONCLUSION & FUTURE WORK

In this paper, we present the first working prototype of LITMUS Benchmark Suite, which is a currently a work in progress. LITMUS is a novel framework enabling benchmarking of both RDF and Property graphs via supporting execution of SPARQL queries over graph databases. It also provides support for visualizing results of benchmarked DMSs using custom plots and an easy to use GUI. In its complete capacity, LITMUS will serve as a common platform for benchmarking RDF, Graph and Relational DMSs, promoting

[16]Complete benchmark results https://goo.gl/BKcbQE
[17]RDF indexing scheme in Virtuoso – https://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtRDFPerformanceTuning#RDFIndexScheme

Table 3: The loading time (in seconds) performance comparison for Northwind (respective versions) in all the DMSs.

DMS	G_mean	H_mean	Max	Mean	Median	Min	Var.
4Store	0.89	0.86	5.010	0.97	0.83	0.640	0.45
Jena	8.21	8.21	9.780	**8.22**	8.16	7.700	0.13
Neo4J	1.48	1.47	2.023	1.49	1.44	1.278	0.031
OrientDB	3.51	3.48	5.612	3.53	3.43	2.832	0.22
RDF3X	0.69	0.68	0.920	0.69	0.66	0.580	0.006
Sparksee	0.72	0.72	0.888	0.72	0.73	0.640	0.001
Tinker	0.61	0.61	1.138	0.62	0.60	0.477	0.010
Virtuoso	0.27	0.27	0.580	**0.27**	0.27	0.250	0.002

Table 4: The warm cache execution time (in seconds) performance comparison for running Query 14 (respective version) on all DMSs.

DMS	G_mean	H_mean	Max	Mean	Median	Min	Var.
4Store	0.021345	0.020474	0.270	0.026250	0.0200	0.020	0.001562
Jena	0.175700	0.175530	0.199	0.175875	0.1740	0.165	0.000065
Neo4J	0.026553	0.026302	0.046	0.026825	0.0270	0.019	0.000017
OrientDB	0.030312	0.029398	0.155	0.032475	0.0310	0.023	0.000410
RDF3X	0.000000	0.000000	0.030	**0.000750**	0.0000	0.000	0.000023
Sparksee	0.027483	0.027321	0.045	0.027675	0.0270	0.023	0.000013
Tinker	0.209582	0.203346	0.363	**0.216650**	0.2075	0.136	0.003564
Virtuoso	0.000000	0.000000	0.026	0.001600	0.0010	0.000	0.000016

Table 5: The cold cache execution time (in seconds) performance comparison for running Query 14 (respective version) on all DMSs.

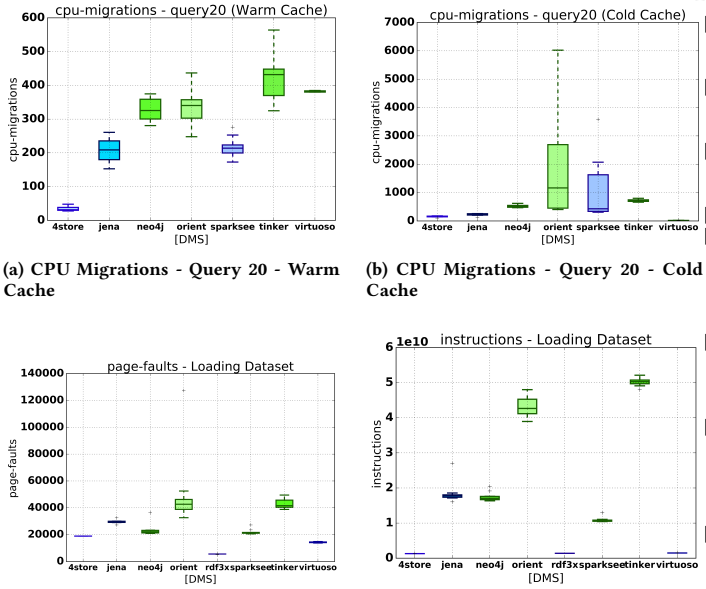
DMS	G_mean	H_mean	Max	Mean	Median	Min	Var.
4Store	4.560672	4.558177	5.050	**4.563250**	4.510	4.340	2.492506e-02
Jena	0.180934	0.180595	0.200	0.181275	0.179	0.163	1.273327e-04
Neo4J	0.028353	0.027910	0.044	0.028875	0.027	0.021	3.626603e-05
OrientDB	0.051870	0.051043	0.091	0.052875	0.049	0.041	1.329327e-04
RDF3X	0.566901	0.563066	0.730	0.571000	0.540	0.470	5.101538e-03
Sparksee	0.044497	0.044061	0.071	0.044950	0.045	0.034	4.435641e-05
Tinker	0.179499	0.177933	0.258	0.181050	0.187	0.136	5.741000e-04
Virtuoso	0.001278	0.001206	0.003	**0.001375**	0.001	0.001	3.429487e-07

easy interoperability, reusability and replicability of existing benchmarks.

As compared to other benchmarking efforts, e.g. Graphium [8], LITMUS provides an end-to-end benchmarking solution ensuring full flexibility to user. With LITMUS it is possible to easily orchestrate benchmarking by adding other DMSs and use various real and synthetic data, whereas, the prior is a one time benchmarking effort result. LDBC [2] on the other hand is an established independent authority which leads a community effort towards standardizing Graph DMS benchmarks and also a graph query language. It consists of individual benchmarks such as the social network, graph-analytics and semantic publishing benchmarks respectively. However, to the best of our knowledge, LDBC does not provide an open extensible automated framework such as LITMUS, which can be used for both small and large scale benchmarking appealing both industry and the academia researchers.

As near future work we aim to – (i) add support for more RDF and Graph DMSs; (ii) integrate *Gremlinator* for enabling automatic SPARQL → Gremlin query translation; (ii) increase the set of supported KPIs, undertaking a systematic study of existing KPIs; and (iii) devise a novel RDF → Property graph converter addressing the requirement to represent complete semantics of RDF graphs (i.e. blank nodes. etc).

As distant future work, we aim to cultivate support for benchmarking relational DMSs by integrating existing solutions for SPARQL query → SQL query translation. A substantial amount of work has been done over the years in the this domain, therefore novel research is not required to be carried out, only a systematic study and integration of existing working solutions.



(a) CPU Migrations - Query 20 - Warm Cache

(b) CPU Migrations - Query 20 - Cold Cache

(c) Page Faults - loading a dataset

(d) Instructions - loading a dataset

Figure 3: Demonstration of sample plots generated by LITMUS Benchmark Suite.

Acknowledgements. This work is supported by the funding received from EU-H2020 WDAqua ITN (GA. 642795).

A SPARQL AND GREMLIN QUERIES

Query 14 (C1): "List all the products in the "beverage" category."

```
select distinct ?s ?o where {
    ?s <http://northwind.com/model/productName> ?o .
    ?s <http://northwind.com/model/category> <http://
     northwind.com/Category-1> }
```

Listing 1: Query 14 in SPARQL.

```
g.V().has("name","Beverages").in("inCategory").values("
    name")
```

Listing 2: Query 14 in Gremlin.

Query 20 (Gc2): "Group the products based on the count of their total number of units on order."

```
select (COUNT(?unitsOnOrder) as ?total) where {
    ?a <http://northwind.com/model/product> ?b .
    ?b <http://northwind.com/model/unitsOnOrder> ?
     unitsOnOrder . } GROUP BY(?unitsOnOrder)
```

Listing 3: Query 20 in SPARQL.

```
g.V().match(__.as('a').hasLabel("product").as('b').values
    ('unitsOnOrder').as('c')).select('c').groupCount()
```

Listing 4: Query 20 in Gremlin.

REFERENCES

[1] Güneş Aluç, Olaf Hartig, M Tamer Özsu, and Khuzaima Daudjee. 2014. Diversified stress testing of RDF data management systems. In *International Semantic Web Conference*. Springer, 197–212.

[2] Renzo Angles, Peter A. Boncz, Josep-Lluis Larriba-Pey, and others. 2014. The linked data benchmark council: a graph and RDF industry benchmarking effort. *SIGMOD Record* (2014), 27–31. DOI: http://dx.doi.org/10.1145/2627692.2627697

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer, 722–735.

[4] Christian Bizer and Andreas Schultz. 2009. The berlin sparql benchmark. (2009).

[5] Mihaela A Bornea, Julian Dolby, Anastasios Kementsietsidis, Kavitha Srinivas, Patrick Dantressangle, Octavian Udrea, and Bishwaranjan Bhattacharjee. 2013. Building an efficient RDF store over a relational database. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 121–132.

[6] Miyuru Dayarathna and Toyotaro Suzumura. 2012. XGDBench: A benchmarking platform for graph stores in exascale clouds.. In *CloudCom*. IEEE Computer Society, 363–370. http://dblp.uni-trier.de/db/conf/cloudcom/cloudcom2012.html#DayarathnaS12

[7] D. Dominguez-Sal, P. Urbón-Bayes, A. Giménez-Vañó, and others. 2010. Survey of Graph Database Performance on the HPC Scalable Graph Analysis Benchmark. In *Proceedings of the 2010 International Conference on Web-age Information Management (WAIM'10)*. Springer-Verlag, 37–48. http://dl.acm.org/citation.cfm?id=1927585.1927590

[8] Alejandro Flores, Guillermo Palma, Maria-Esther Vidal, and others. 2013. GRAPHIUM: visualizing performance of graph and RDF engines on linked data. In *Proceedings of the 2013th International Conference on Posters & Demonstrations Track-Volume 1035*. CEUR-WS.org, 133–136.

[9] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. 2005. LUBM: A Benchmark for OWL Knowledge Base Systems. *Web Semant.* 3 (2005), 158–182. DOI: http://dx.doi.org/10.1016/j.websem.2005.06.005

[10] Olaf Hartig. 2014. Reconciliation of RDF* and property graphs. *arXiv preprint arXiv:1409.3288* (2014).

[11] Mohamed Morsey, Jens Lehmann, Sören Auer, and others. 2011. *DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data*. Springer Berlin Heidelberg, 454–469. DOI: http://dx.doi.org/10.1007/978-3-642-25073-6_29

[12] Richard C Murphy, Kyle B Wheeler, Brian W Barrett, and James A Ang. 2010. Introducing the GRAPH 500. *Cray User's Group (CUG)* (2010).

[13] Raghunath Nambiar, Nicholas Wakou, Forrest Carman, and others. 2011. *Transaction Processing Performance Council (TPC): State of the Council 2010*. Springer Berlin Heidelberg, 1–9. DOI: http://dx.doi.org/10.1007/978-3-642-18206-8_1

[14] Axel-Cyrille Ngonga Ngomo and Michael Röder. 2016. HOBBIT: Holistic benchmarking for big linked data. *ERCIM News* 2016, 105 (2016).

[15] Eric Prud, Andy Seaborne, and others. 2006. SPARQL query language for RDF. (2006).

[16] Marko A Rodriguez. 2015. The gremlin graph traversal machine and language (invited talk). In *Proceedings of the 15th Symposium on Database Programming Languages*. ACM, 1–10.

[17] Muhammad Saleem, Muhammad Intizar Ali, Aidan Hogan, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2015. LSQ: The Linked SPARQL Queries Dataset. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*. 261–269. DOI: http://dx.doi.org/10.1007/978-3-319-25010-6_15

[18] Muhammad Saleem, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. 2015. FEASIBLE: A Feature-Based SPARQL Benchmark Generation Framework. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*. 52–69. DOI: http://dx.doi.org/10.1007/978-3-319-25007-6_4

[19] Muhammad Saleem, Ricardo Usbeck, Michael Roder, and Axel-Cyrille Ngonga Ngomo. 2016. SPARQL Querying Benchmarks. (2016). https://www.researchgate.net/publication/300005911_SPARQL_Querying_Benchmarks

[20] Michael Schmidt, Thomas Hornung, Michael Meier, and others. 2009. SP2Bench: A SPARQL Performance Benchmark. In *Semantic Web Information Management*. Springer, 371–393. http://dblp.uni-trier.de/db/books/collections/Virgilio2009.html#SchmidtHMPL09

[21] Harsh Thakkar. 2017. Towards an Open Extensible Framework for Empirical Benchmarking of Data Management Solutions: LITMUS. In *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part II*. 256–266. DOI: http://dx.doi.org/10.1007/978-3-319-58451-5_20

[22] Harsh Thakkar, Dharmen Punjani, Maria-Esther Vidal, and Sören Auer. 2017. Towards an Integrated Graph Algebra for Graph Pattern Matching with Gremlin. In *Proceedings of the 28th International Conference, DEXA 2017, Lyon, France, August 28-31, 2017, Proceedings, Part I*. Springer, 81–91.