RESEARCH

Reproducible Query Performance Assessment of Scalable RDF Storage Solutions

Dieter De Witte[1*], Laurens De Vocht[1], Dieter De Paepe[1], Filip Pattyn[2], Kenny Knecht[2], Hans Constandt[2], Jan Fostier[1], Ruben Verborgh[1] and Erik Mannens[1]

Abstract

Background: An increasing number of applications in the biomedical domain rely on Linked Data spanning multiple datasets. Choosing a strategy for running federated queries over Big Linked Data is a challenging task: given the abundance of Linked Data storage solutions, it is not straightforward to make an informed choice between platforms.

Results: We provide an extensive review of state-of-the-art Resource Description Framework (RDF) storage solutions. We assessed query performance of seven RDF systems: four commercial and three research prototypes. We explored the benchmark space, iterating over dataset sizes, query templates, and query contexts for different hardware and engine configurations. In addition to an artificial benchmark, we conducted real-life tests with queries from a biomedical search application. Results were interpreted with the help of decision trees trained on query features.
To facilitate reproducibility, we release our benchmark data in a rich event format. Additionally, we provide post-processing scripts in the form of Jupyter notebooks and tools that facilitate deployment and configuration of benchmark components.

Conclusions: Results show that single-node triple stores benefit greatly from vertical scaling and proper configuration, while horizontal scalability is still a real challenge to most systems. Research prototypes based on federation, compression, or Linked Data Fragments still lag behind by an order of magnitude in terms of performance. Furthermore, we demonstrate the need for careful analysis of contextual factors influencing query runtimes: server load, availability, caching effects, and query completeness all perturb the benchmark results. We believe that by making the full benchmarking pipeline publicly available, we facilitate future benchmarking efforts.

Keywords: Resource Description Framework (RDF); Benchmarking Methodology; Big Linked Data; Distributed Querying; Semantic Web

Introduction

The vision of the *Semantic Web* is to serve as a global distributed database which can be autonomously explored by intelligent agents. The Linked Open Data (LOD) project [1] is an implementation of this vision and already contains over 1,000 interconnected datasets in Resource Description Framework (RDF) format. Semantic Web technology has a lot to offer to multidisciplinary research domains. For example, Life Sciences span multiple domains ranging from pharmacy to genetics to clinical trials. In the LOD cloud prominent Life Sciences datasets are UniProt (protein function), Drugbank (drugs) and chEMBL (properties of molecules). Being able to interact with these datasets as one virtual source requires technology capable of both managing Big Linked Data, as well as successfully answering complex federated queries.

Challenges

Datasets on the Semantic Web can be queried using RDF database systems that (partially) support the SPARQL Protocol and Query Language (SPARQL). There is an abundance of RDF storage solutions each with their own strengths and weaknesses. Choosing an

*Correspondence: drdwitte@gmail.com
[1]imec - IDLab - Ghent University, iGent Tower - Technologiepark-Zwijnaarde 15, BE 9052 Ghent, Belgium
Full list of author information is available at the end of the article

RDF database and system architecture therefore requires making trade-offs:

- What features are required for the system of choice given the use case at hand, what features are optional?
- What hardware is required to achieve a certain performance?
- What are the risks when using research prototypes instead of mature vendor-backed solutions?

To make matters even more complicated, database vendors are continuously improving their products rendering previous benchmark results quickly outdated. The goal of this work is to provide an up-to-date view on the RDF storage solution space. By providing scripts for deployment and post-processing of results, we provide a feasible approach to run benchmarks with own data and queries within a limited time window. This work also offers a methodology to make benchmarks more reproducible and therefore the results more easily generally applicable.

Research Questions

The work presented here is structured around 4 research questions:

1 *How to run a query performance benchmark in a reproducible and reliable way?* This will be mostly addressed in the section 'Benchmark Approach'. Reliability is the focus of result sections 'Query Result Completeness' and 'Benchmark Error Analysis', which deal with errors and query completeness.

2 *What are the different options and the associated trade-offs when choosing a Linked Data infrastructure setup in the context of Big Linked Data? How can different setups be compared?* In the section 'Results I' we will demonstrate that the choice of engine depends on the size of the dataset. In 'Results II' we will investigate which engine works best for certain query types. How completely different solutions can be compared is the subject of the section 'Benchmark Cost' in 'Results III'.

3 *What is the relative influence on the measured performance of contextual factors for the different RDF solutions? Is the impact similar for all solutions?* In the section 'Results II' we will demonstrate that query runtime results cannot be interpreted without taking into account the *query context*: server load, availability, caching effects, etc. all have an impact on the individual runtimes.

In section 'Results III' we run a custom Life Sciences benchmark in the context of federated faceted browsing.

Our Contribution

- **Reusable Feature Matrix:** Since RDF systems have a wide range of diverse features, one system might be preferred above another depending on the specific use case. To facilitate this decision-making step, we created a Feature Matrix. This matrix consists of 50 RDF database features for 12 systems. The user can assign weights to each of these features which enables the creation of a ranking, useful for system architects having to make a database pre-selection.
- **Reusable Benchmark Methodology:** This paper demonstrates a methodology to evaluate RDF storage solutions on a data and query-set of choice with a focus on reproducibility and reusability. To enable RDF architects to more easily run benchmarks with their own queries and data, we release scripts to facilitate deployment and post-processing of the results. The pitfalls in the interpretation of the results are highlighted and suggestions are formulated to circumvent them and draw the right conclusions.
- **Demonstration on Big Linked Data:** We demonstrate our methodology to evaluate the ability of today's triple stores in terms of scalability with big biomedical data sources and complex real-world queries. This research paper builds on the results of 51 new benchmark runs using 4 different datasets and 7 RDF storage systems.
- **Query runtime results in context:** This work tries to create a balanced view on performance parameters, such as query runtime, by putting them in a context, thereby no longer viewing query executions as stateless.
- **Benchmark Cost:** Using financial cost as the *dependent variable* enables the comparison of systems with different hardware, licensing costs, and architectures. This makes it possible to quantify certain trade-offs. For example, querying federated resources versus offloading everything and hosting it on a local single- or multi-node system.

Related Work

There is an abundance of Linked Data benchmarks mainly operating on *artificial* datasets, the most pop-

Table 1: Overview of recent (2011-2016) benchmarking results. Virtuoso is abbreviated as Virt and GraphDB as Gra.

Benchmark/Paper	Year	Dataset(s)	Triple Stores	Nodes x RAM	Remarks
Graux et al. [2]	2016	WatDiv1k, LUBM1k, LUBM10k	Standalone (CumulusRDF [3],...) HDFS with prep.: S2RDF [4],... HDFS no prep.: PigSPARQL [5],...	10 x 17GB	
SPB [6]	2016	SPB64M, SPB256M, SPB1B	Virtuoso, GraphDB	Virt(192 GB), Gra(64GB)	
Hernandez et al. [7]	2016	Wikidata	4store [8], Blazegraph, GraphDB, Jena TDB, Virtuoso, Neo4J, PostGreSQL	1 x 32GB	
S2RDF [4]	2016	WatDiv10M, WatDiv100M	S2RDF [4], H2RDF+ [9], Sempala [10], PigSPARQL [5], SHARD [11], Virtuoso	10 x 32GB, Virt(1 x 32GB)	
BigRDFBench [12]	2015	13 real datasets	FedX [13], SPLENDID [14], ANAPSID [15], FedX+HiBISCuS [16], SPLENDID+HiBISCuS	1 x 8GB	FedBench + 18 new queries
FEASIBLE [17]	2015	generator	Virtuoso7, Sesame, Jena TDB, OWLIM-SE	1 x 16GB	
WatDiv [18]	2014	WatDiv10M, WatDiv100M	MonetDB [19], RDF-3X [20], Virtuoso6, Virtuoso7, gStore [21], 4store	1 x 16GB	
Cudré-Mauroux et al. [22]	2013	BSBM (10, 100, 1000M) DBPSB	4store, Hive+HBase, CumulusRDF, Couchbase, Jena+HBase [23]	2^n x 8GB $n = 0, 1, ...4$	
BioBenchmark Toyama [24]	2012	5 biological datasets (10M - 8000M) Uniprot, DDBJ,...	4store, BigData, Mulgara, Virtuoso, OWLIM-SE	1 x 64GB	5-20 queries per dataset
FedBench [25]	2011	11 endpoints with ≤ 50M	SPLENDID, Alibaba, Sesame	1 x 32GB	14 federated queries (7 life sciences, 7 cross-domain)

ular ones being (chronologically) the Lehigh University Benchmark [26] (LUBM), the SPARQL performance benchmark [27] (SP^{2}Bench), and the Berlin SPARQL benchmark [28] (BSBM). The shortcomings of these early benchmarks were addressed in recent work, which resulted in the Waterloo SPARQL Diversity Test Suite [18] (WatDiv). This new benchmark focuses on *diversity* both in terms of the query properties and data properties. The first is achieved by generating queries from 20 Basic Graph Pattern (BGP) query templates with different shapes. The latter affects the triple pattern selectivity and therefore reveals the ability of the internal query planning algorithms in RDF systems to make the most efficient choice to resolve a query. More diverse SPARQL queries can be generated through the FEASIBLE [17] benchmark generator. The queries are selected by first converting them to normalized feature vectors and then choosing a set of mutually distant queries. Similarly, the Semantic Publishing Benchmark [6] (SPB) provides complex nested queries and uses all SPARQL 1.0 operators.

A recurring criticism on synthetic benchmarks is that they have very little in common with real application domains [29]. Therefore, it is not possible to generalize benchmark results of RDF databases on artificial data to real-world use cases. One of the first real-world benchmarks is the DBpedia SPARQL benchmark [30] (DBSB) which uses mostly BGPs extracted from actual server logs. Specifically for the Life Sciences domain, BioBenchmark Toyama 2012 [24] evaluates 5 triple stores on 5 biological datasets (Cell Cycle Ontology [31], Allie [32], PDBj [33], UniProt [34], and DDBJ [35]), ranging from 10 million to 8 billion triples.

All benchmarks mentioned so far focus on single-node RDF databases. FedBench [25] is a system to test query federators. They evaluate 3 federated systems using 14 real-world federated queries, of which 7 from the Life Sciences domain. In recent work, BigRDFBench [12] increases the number of datasets from 11 to 13 and adds 18 new federated queries. Instead of just focusing on query runtime, other performance metrics are taken into account such as source selection and query correctness. An alternative heuristic approach for automatically generating federated queries is the SPARQL Linked Open Data Query Generator [36] (SPLODGE).

The previously mentioned benchmarking efforts focus on native RDF systems. A first generalization involves adding other graph and relational databases. For example, in the WikiData benchmarking effort [7], Neo4J and PostgreSQL were added. A second generalization involves mapping SPARQL workloads on NoSQL and Hadoop-based systems [22, 2, 4]. Graux [2] compared 3 different types of systems: (i) Standalone NoSQL-based approaches such as CumulusRDF [3] which translates queries to the Cassandra Query Language; (ii) HDFS-based (Hadoop Distributed File System [37]) approaches with a data preparation phase such as S2RDF [4]; and (iii) HDFS-based approaches which natively store RDF, such as PigSPARQL [5].

RDF systems are also evaluated in two European H2020 projects: LDBC [38] and HOBBIT [39]. Within LDBC a number of RDF benchmarks were developed [40], one benchmark is based on social network data [41] and SPB [6] is based on a data publishing case with BBC. In the HOBBIT project a platform is being built to offer industry a unified approach for running benchmarks related to their actual workloads.

In Table 1 we provide an overview of the most recent benchmarking results together with information on their time of release, the datasets used, the systems tested, and the hardware setup.

Positioning this work

Scalable Approaches Our work distinguishes itself from other efforts by considering seven scalable approaches to handle Big Linked Data: four vendor-backed RDF databases and three research prototypes. Any system can be tested with our approach, the only requirement is support for the SPARQL protocol.

Comprehensive The benchmark is comprehensive in evaluating query performance: the dataset volume, the different hardware and engine configurations, the query properties, and the query context are all taken into account. We also explicitly analyze whether the query results are complete.

Life Sciences domain We provide an up-to-date review of state-of-the-art RDF storage systems both on artificial datasets as well as on Life Sciences data.

Reproducible We make it easy to reproduce our benchmark results by providing scripts and images for automatic deployment of the benchmark components as well as the notebooks for post-processing.

Unbiased Results Conclusion are drawn across multiple approaches to Linked Data querying by focusing on the financial cost instead of query runtimes. Results are interpreted in an unbiased fashion, with the help of decision trees trained on query features.

Prior work

In our initial conference paper [42] we evaluated 4 RDF databases on WatDiv [18], with 3 different dataset sizes: 10M (10 million), 100M and 1000M triples. These *Vendor* systems were run as-is, without any configuration.

Ontoforce [43] provided us with a real-world Life Sciences dataset used in the back-end of their product DISQOVER [44]. This proprietary data and query-set was previously analyzed in our SWAT4LS proceeding [45]. of which we provide a summary in the section 'Datasets and Queries'. In this proceeding we analyzed the queries according to their SPARQL keywords and structural features [46], an approach we will further extend in this work.

Counter-intuitive results in these conference papers motivated re-running some of the benchmarks in this work. We paid close attention to query completeness and engine configuration. For the latter we distinguish between a *Documented* and an *RFI-optimized* configuration (see section 'Store Preselection').

In the current work all seven systems are evaluated on both WatDiv and the Ontoforce benchmark. (this was previously not the case!)

Benchmark approach

In this section we describe our benchmark methodology: we define a benchmark space and describe how it was explored. We explain how we made a pre-selection of triple stores and how they were configured. We describe the datasets and queries used in this work. Finally we list our efforts at making the benchmark more easily reproducible.

Benchmark Space Exploration

Following parameters are assessed:

- **The choice of database engine:** We assess 7 different systems, 4 *Vendors* and 3 *Prototypes*. For the *Vendors* we considered Blazegraph, GraphDB, Virtuoso and a non-disclosed Enterprise System (ES), which did not give permission to disclose its name. The *Prototypes* are HDT [50], FedX [13], and Triple Pattern Fragments [51].

- **The server memory:** We distinguish between 32GB and 64GB of RAM.
- **The size of the (optionally) distributed system:** We run tests for single and 3-node setups when supported by the RDF database. Federated systems are configured with $N+1$ nodes, with N the number of slave nodes (1 or 3), and 1 federator node. To clarify: $N = 3$ thus corresponds to 3 instances for *Vendor* systems, while $N = 3$ for federated setups requires $3+1$ instances. The choice for $N = 3$ is related to the fact that for one of the systems only a 3-node configuration is available.
- **The query properties:** The WatDiv benchmark query-set contains BGP queries, while the Ontoforce dataset consists mainly of complex aggregation-based queries with many filters.
- **The number of dataset triples:** We run 3 datasets of WatDiv, with 10 million, 100 million, and 1 billion triples. The Ontoforce dataset contains 2.4 billion triples.
- **The way in which the RDF system is configured:** We test two configurations for WatDiv1000M, namely the *Documented* and the *RFI-optimized* configuration, which will be defined later in this section.
- **The state of the system when the query is launched:** We distinguish between a single-threaded warm-up run and a multi-threaded stress test (5 clients). We also investigate whether caching effects play a role in the runtime behavior.

We are here: 27/09 18u

Testing every possible combination of parameters is very time and resource consuming and not necessarily the most informative. Therefore we opted for a greedy exploration of this space consisting of 51 2-phase benchmarks (incl. re-runs), each with a warm-up and a consecutive stress test (see Table 2).

Table 2: Benchmark overview.

Systems	Setup	WatDiv 10M	WatDiv 100M	WatDiv 1000M	Onto-force
Vendors (4)	32	✓	✓	✓	
	64			✓	
	64/Opt			✓	✓
Multi-Node (2)	3 x 32			✓	
	3 x 64/Opt				✓
Prototypes (3)	64		✓	✓	✓
	3 x 64		✓	✓	✓

Store Preselection and Configuration

We created a Feature Matrix and evaluated a number of stores on a subset of those features (a similar approach as in Stegmaier [47]) to make a preselection of RDF engines. We combined two ideas to create a Feature Matrix, to simplify the RDF store selection process:

- We consulted the DB-Engines ranking [48], which orders database systems according to their data model and online popularity, as measured by the mentions on social platforms such as StackOverflow, Twitter, and LinkedIn. DB-Engines also supports comparing multiple features of different systems.
- WikiData selected the most appropriate RDF store to host their data by having experts assign weights to desired features [49]. These weights allowed them to calculate a score per data store and rank the different systems.

The feature matrix contains a broad selection of suitable features specific for RDF engines and allows for multi-way comparisons. Engines are ranked by assigning weights to these features. The feature matrix is available online (see suppl material 1)), and can be freely downloaded and extended. To back the scoring, we added a layer of trust by linking to the source of information. The criteria for selection of the *Vendor* systems are the following: SPARQL 1.1 compliance, systems with a machine or maintained Docker image, no restrictions on the number of triples that can be ingested, and support for multi-node deployment. This led to 4 *Vendor* systems: Blazegraph,... Additionally, we added 3 additional research *Prototypes* with unique approaches to handling RDF data: HDT [50], which is a queryable read-only binary compression format, FedX [13] often included in benchmarks for federated querying, and Triple Pattern Fragments [51] as a first implementation of the Linked Data Fragments concept.

Selected stores are shown in Table 3 together with their shorthand notation.

We run the benchmarks using two strictly defined configurations: *Documented* and *RFI-optimized*.

The *Documented* configuration corresponds to the recommended settings from the vendor documentation, which takes into account the available server memory and the dataset size. The *RFI-Optimized* configuration was obtained after sending out a Request For Information (RFI) to the commercial vendors involved. The RFI asked them to provide us with scripts or configuration files to achieve optimal performance

Table 3: List of the tested systems and their acronyms.

System	Shorthand
Blazegraph 2.1.2	**Bla**
Undisclosed Enterprise Store	**ES**
GraphDB 7.0.1	**Gra**
Virtuoso 7.2.42	**Vir**
FluidOps [52] (with FedX 3.1.2 [13])	**FedX**
HDT-Fuseki 4.0.0 [53]: Jena Fuseki to query HDT	**Fus**
Triple Pattern Fragments: Server.js 2.2 [54], Client.js 2.0 [55]	**TPF**

on the WatDiv1000M benchmark. GraphDB, Virtuoso, and Blazegraph responded positively to this request. A fourth commercial vendor, ES, did not respond to our RFI. Note that this configuration is not necessarily an optimal match for the real-world benchmark, as the data and queries were not shared with the vendors.

Datasets and Queries

We used the WatDiv benchmark generator [18] to create three artificial datasets of 10, 100, and 1000M (million) triples. WatDiv datasets are also used for federated setups. In these setups the dataset was partitioned using subject hash partitioning [56, 57] which led to three equally-sized datasets. Note that this partitioning scheme benefits star-shaped queries, as they can be resolved without inter-node communication.

The WatDiv queries reveal the ability of today's triple stores to handle different types of complex join operations. The queries are generated from 20 query templates (BGPs) in four categories:

- L: Linear chains (**L1 - L3**)
- S: Star-shaped queries with one central node (**S1 - S7**)
- F: Snowflake queries are a combination of S queries (**F1 - F5**)
- C: Combinations of the above (**C1 - C3**)

Per query template we generated 20 queries corresponding to 400 queries in total.

A real-world proprietary dataset was provided by Ontoforce. The query and dataset properties were analyzed in prior work [45]. The actual dataset cannot be disclosed but the majority of the data is from public sources. The dataset consists of 2.4 billion triples spanning 107 datasets. PubMed, chEMBL, NCBI-Gene, DisGeNET, and EPO are the largest graphs with PubMed already making up 60% of the data. biblio?

The query-mix provided by Ontoforce is publicly available (see Suppl material) and was extracted from the user logs of the DISQOVER search interface. The queries are interactive federated queries associated with faceted browsing [58, 59], for example:

"Get the number of drugs per development phase having 'migraine' in their description for manufacturer 'Sandoz inc'. Phases come from chEMBL, manufacturers come from DrugBank."

The corresponding query features are: Triple Patterns (11), nested select queries (3), query file size (< 1kB), operators: OPTIONAL (1), GROUP(1), ORDER(1), COUNT(1), UNION(1), FILTER(7), FILTER IN(2).

The queries of the two benchmarks are very different in nature. The 1,223 DISQOVER queries are rich in SPARQL-features and sub-queries are common. This is a stark contrast with WatDiv for which all queries are BGPs. The DISQOVER queries are automatically built by the system from more general queries to which additional FILTER statements are added, while browsing the UI. Aggregation operators and FILTER operations are therefore predominant. A large fraction of queries is also non-conjunctive [60], making them even more challenging [61]. Queries with over 10 triple patterns are common and more specifically unbound triples, with three variables, occur often. The actual binding occurs in the additional FILTER statements. Half of the queries are COUNT DISTINCT queries and these are also the most time consuming to resolve. Due to the automated way of generating queries, their formulation is not optimized in terms of performance [62].

A quick and reusable benchmarking scheme

Public Compute Infrastructure
The choice of hardware in benchmarks is often related to the availability of systems in a research group's data center. We used three different types of servers on the Elastic Compute Cloud (EC2) of Amazon Web Services [63] (AWS), shown in Table 4.

Table 4: Instance types used in benchmarks and their purpose.

Instance Type	vCPUs (no.)	RAM (GB)	Goal
r3.xlarge	4	30	Original Choice
r3.2xlarge	8	61	Current Reference
c3.2xlarge	8	15	Benchmarker

An additional advantage of this approach is that the benchmark financial cost can be explicitly provided. Using financial cost as a metric allows the comparison of benchmarks with different setups. Also the cost of certain preprocessing steps such as bulk loading or compression can be included in the comparison.

Reproducible installations and configurations
A reproducible installation strategy is obtained by using Amazon Machine Images (AMIs) offered by the system vendors on the AWS Marketplace [64]. When no AMI is available we turned to well-maintained Docker images [65]. The AMIs come with a *Pay-As-You-Go* (PAGO) license. The following AMIs and Docker images were selected:

- PAGO AMIs: Virtuoso [66], GraphDB [67], ES
- Docker Hub: TPF server [68], HDT-tools [69], Virtuoso Open Source [70]
- Self-provided Docker images: Blazegraph [71], HDT-Fuseki [53]
- Manual installation: FedX [72] 3.1.2 was installed manually with the Virtuoso Adapter plugin.

The different configuration settings (*Default* and *Optimized*) for the Vendor systems are publicly available (see Supp material).

Reusable Benchmark Components
The SPARQL Query Benchmarker software [73] is a mature SPARQL-over-HTTP benchmarking tool which is highly customizable. We ran the software in *benchmark* mode where it can operate given a SPARQL endpoint URI and a list of SPARQL query files. The software was run with a timeout parameter of 300s for the WatDiv benchmarks and 1200s for the Ontoforce benchmark and with 1 single-threaded warm-up run and a multi-threaded (5 threads) stress test run where 5 clients each execute a full query mix independently and in randomized order. Note that we left a 2-hour time gap in between the ingest phase and the warm-up run to ensure all processes related to ingest had finished. The choice for timeout parameters is related to practical considerations:

- Initial tests revealed that the WatDiv timeout is sufficient for most queries to complete.
- The Ontoforce benchmark timeout was set to keep the total benchmark execution time within affordable boundaries.

Reusable Post-processing and Unbiased Conclusions
When the benchmark successfully terminates, a CSV-file is generated containing the summary results per query: median runtime, median response time, etc. In our initial benchmarks [42] this CSV-file was used, but with the Ontoforce dataset several issues surfaced:

- The summary results (number of results per query and query runtimes) are not correct in benchmarks where many problems arise. For example, in the calculation of the average runtime, results where the query was unsuccessfully resolved are also taken into account for the calculation of the average. It also makes it hard to verify the number of results per query. For example, a query with 10 results which is executed twice and of which one execution fails, is reported as having 5 results .
- If the benchmarker software fails, the CSV-file is not generated and the results are lost.
- A posteriori it is not possible to verify if a query was solved correctly.
- While the CSV-file contains useful results, it is still a summarization and much information about the flow of the benchmarking process is lost.

These issues could be addressed by working with the raw benchmarker log files which contain more details. The post-processing pipeline parses this log file and converts it into a more detailed CSV file which contains *query events*. These events contain the essential information of a single query execution. Query events serve as the basis for all results in this research paper. The schema of a single query event is shown in Table 5. All event files and derived views are online [74].

Table 5: Schema of the query events used for all benchmark results in this work

Field	Range
sim_id	engine, number of nodes, memory, config.
query_name	400 IDs for WatDiv, 1,223 for Ontoforce
thread_id	6 ids
thread_type	warm-up (1 thread) or stress (5 threads)
order_id	the offset in the query mix for a thread
number_of_results	-1 if error, ≥ 0 otherwise
runtime	(seconds), error: -1, timeout: max. value
flag	SUCCESS, ERROR, TIMEOUT
correct	(IN)CORRECT (if #results $\neq$ consensus)

The query events can also be used to study query correctness since they contain the number of results per query and a flag for (un)successful query execution. For the Ontoforce benchmark however, almost half of the queries are COUNT queries, for which the result count does not provide any guarantees on correctness. To verify the correctness of these queries we extended the benchmarker software enabling it to store the actual query results, which allows us to compare the results of the COUNT queries.

To simplify the deployment of this modified benchmarker client, we automated this process by creating a

Docker container which automatically installs the software and its dependencies.

Next, we automated major parts of the post-processing of benchmark results, because (i) this saves the future benchmark user a lot of time parsing the benchmarker log files, (ii) provides the user with a large set of instant visual results, and (iii) allows knowledge-transfer to new benchmarking efforts through script re-use.

Jupyter notebooks [75] were used for the postprocessing. All notebooks are available online [76].

Practice has shown that the event format leaves room for unanticipated analysis. For example: dealing with incorrect queries, taking into account server load or caching effects, studying the reason behind one of the query engines crashing,...

Finally, also part of the conclusions drawn from the data are automated by using a machine learning approach to find the relation between certain *query features* and performance parameters. This ensures that the conclusions are not the result of the author's insights or biases.

Results I: Approaches to Linked Data at Scale

In this section we will study the query runtime distributions of different approaches for dealing with Linked Data at scale. To summarize runtime results we rely on mean and median values.

Queries are typically executed multiple times. The *query runtime* is defined as the median value of the different query executions in a run.

If we aggregate runtimes over *different queries*, for example when we aggregate per query template, we report both the *median* and *mean query runtimes*.

As the runtime distributions can be skewed, performance differences between systems are most often reported using the median runtime of an aggregate or benchmark run.

If we consider an ETL process, or equivalently a batch of queries, the mean runtime becomes more meaningful, as it directly translates to the total runtime of an aggregate or full run. In the following box plots we chose to report both.

Query response times correspond to the lag between sending a query and receiving the first server response. Some of the stores provide query results in a streaming fashion, therefore response times can be different from query runtimes. Response times are not captured in the current query event format but are captured in the SPARQL benchmarker summary CSV-files. For GraphDB and Blazegraph the response times are respectively 27% and 21% lower than the mean runtimes on WatDiv1000M. For the other engines the difference was close to zero.

A major concern when comparing query runtimes between different engines is *query completeness*. The current query event format, shown in Table 5, explicitly reports whether a query was solved correctly, meaning it has retrieved the complete set of results. In the sections 'Results III' query completeness is the topic of of the first subsection. To interpret the results in this section correctly, it is important to understand that queries, which have incomplete results for at least one benchmark, are completely discarded in the runtime comparisons.

Table 6: Conventions for describing benchmark setups.

Shorthand Notation	Full Description
Vir1_32_Def	Virtuoso - single node - 32GB RAM - Default Configuration
TPF3_64_Def	Triple Pattern Fragments - 3 slave nodes - 64GB RAM - Default Configuration
Gra1_64_Opt	GraphDB - single node - 64GB RAM - Optimized (RFI) Configuration

A description consists of a 3-character prefix describing the RDF storage solution, the number of nodes, the amount of memory and the configuration.

Finally, a subtle error can be made in query runtime comparison for benchmarks which involve a query engine that becomes unresponsive (engine failure). In the runtime comparisons we only consider the range between the first query and the last successful query. We name this the *benchmark survival interval*, this is shown in Figure 3.

In Table 6 we introduce a naming convention to describe the different benchmark setups.

Increasingly Large Datasets

The previous benchmark results [42] stem from the *None* configuration. In this section however, we use the *Default* configuration of the *Vendor* systems. In Figure 1 query runtime distributions are shown for the 4 *Vendor* systems for three different dataset sizes of the WatDiv benchmark: 10M, 100M, and 1000M (million) triples. Note that for these benchmark runs we used a setup with 32 GB memory.

- **Runtime vs Dataset Size:** Although only 3 data points are available for 10, 100 and, 1000M triples, it is interesting to investigate how the runtime scales when the dataset grows by a factor 10. If we focus on the average query runtimes (dots) two trends can be observed: **Vir1_32** has a nearly constant multiplication factor (mf) while for the other stores this is not the case. Going from 10M to 100M the mfs are 8, 11, and 17 for **Bla1_32**, **Gra1_32**, and **Vir1_32** respectively. Going from 100M the mf for **Vir1_32** is 19, but for the other systems mf > 120! A possible explanation for this trend break is that memory swapping occurs. This observation motivates the choice for 64GB memory instances as the central reference setup from which to explore the benchmark space.

- **Timeouts & Errors:** Most of the queries are executed successfully by all *Vendor* systems. For WatDiv1000M **Bla1_32** already has a timeout percentage of 11.6% and for **ES1_32** this is even 32.7%. Note however that these results do not affect the plots as we only use query events from the *benchmark survival interval*.
- **GraphDB vs Virtuoso:** In terms of median runtime both **Gra1_32** and **Vir1_32** are tied at 0.01s and 0.05s in the two leftmost panels of Fig. 1. With sufficient memory these engines can remain competitive. However, for 1000M dataset only **Vir1_32** is performing well, with an increase in median runtime with a factor of 18.6 compared to 100M run while the median runtimes of the other stores increase with a factor of 100 or more. **Gra_32**, more than the other stores, suffers from a long tail which has a major effect on the average runtimes for WatDiv10M and 100M. If we compare these runtimes, Virtuoso is 1.5-2 times faster.
- **Blazegraph vs. GraphDB: Bla1_32** competes with **Gra1_32** in terms of average runtimes but not in terms of median runtimes.
- **ES consistently last: ES_32**, even on WatDiv10M, lags by a factor of at least 5 to the other systems. For WatDiv100M already the nonlinear scaling behavior sets in, making it the only engine to experience problems with the 100M dataset.

Vertical Scaling

In the previous section we saw that the amount of memory is a critical parameter for benchmark performance. In this section we study the effect on the query runtimes of increasing the amount of memory to 64GB. The two leftmost panels of Figure 2 study the effect of vertical scaling.

- **Memory is no magic solution:** Especially for **Gra1_64_Def** and **Vir1_64_Def** hardly any improvement can be seen. Blazegraph takes full advantage of the additional memory, with a large shift in both median and mean runtimes. The strong hardware dependence of Blazegraph could be a motivation to also study the performance in a GPU setting [77], which is outside the scope of this work.
- **Speedups: Bla1_64_Def** has a speedup of 8.4 for its average runtime and 3.1 for its median runtime. From the other stores only **ES1_64_Def** benefits with a speedup of 1.8 for its average runtime.
- **Benefits for fast queries:** The most outspoken positive effect is the lowering of the lower boundary of the box plots.

RFI: Optimized Configurations

After contacting the vendors with our initial results one of the parties suggested to demonstrate the optimal operation of their database. This was formalized by sending out a Request-For-Information (RFI), specifying the benchmarks we were planning to run. 3 out of 4 vendors chose to participate in the RFI, which resulted in an *Optimized* configuration.

In Figure 2 the rightmost panel corresponds to running the benchmark with the *Optimized* configuration.

- **Sensitivity to configuration: Vir1_64** got no benefit from the RFI settings file. For **Bla1_64** the only improvement was to explicitly configure the timeout parameter on the server side. This avoids unnecessary overhead while the client is already disconnected. It leads to a speedup of approximately 3.5 for both runtime measurements. **Gra1_64** has the highest sensitivity to proper configurations. The provided scripts ensure a speedup of 9.4 for the average runtime and a median runtime speedup of 62.
- **32_Def to 64_Opt:** Moving from the left panel to the right in Figure 2, we clearly see results converging in the rightmost window with the **64_Opt** measurements. **Bla1_64** is the most efficient system for processing batch workloads with an average runtime of 1.95s per query, 4.05s and 6.32s for **Vir1_64** and **Gra1_64** respectively. In the query performance **Vir1_64** has a median runtime of 0.17s where **Gra1_64** and **Bla1_64** have runtimes of 0.65s and 0.74s respectively.
- **Runtime vs Dataset Size:** Returning to section 4.1 we can verify that the linear scaling behavior is largely restored, confirming our earlier hypothesis. Multiplication factors drop to 4.2 for Blazegraph, for Virtuoso and GraphDB mf ≈ 15.

- **Timeouts & Errors:** Apart from 5% timeouts for **Gra1_64_Opt**, no query errors are observed with the *Optimized* configurations.

Research Prototypes

As the initial goal of the research collaboration with Ontoforce was to find a solution to work with federated querying on top of live data sources on the Semantic Web, we discuss the results of **Fus1_64**, **FedX3_64**, **TPF1_64** and **TPF3_64**. Figure 3 deliberately has no relation with query runtimes. For these 3 systems engine failure and query errors are very common with only the **TPF*_64** systems surviving the entire benchmark.

- **FedX1_64:** The federation system with 1 source node is added to verify that FedX in fact manages to parse the queries. As the queries have the same form for WatDiv100M we only tested this in the 1000M setting.
- **Specific Templates cause crashes:** Where **TPF*_64** systems more gracefully timeout on the **C** templates, **C2** causes a crash in **Fus1_64** and **C3** in **FedX3_64**, upon their first occurrence in warm-up or stress run. **C3** is a query with very low triple pattern selectivity leading to large in-memory joins.
- **Crash investigation:** For **FedX3_64** the benchmark was terminated after running into constant timeouts for 8 hours. Upon inspection of the slave nodes (VOS), these turned out to be idle, while the federator node had its entire memory pool saturated, with the CPU load close to zero. A possible explanation might therefore point in the direction of issues with garbage collection. For **Fus1_64** after a number of queries a continuous timeout sequence sets in. Peculiar was that the specific HDT implementation for Fuseki seemed to ignore the timeout parameter which might explain why the server overloaded and became unresponsive.
- **Staying alive: TPF*_64** survive both WatDiv benchmarks, nonetheless up to 71% of the queries timeout for WatDiv1000M. For WatDiv100M the timeout ratio drops to 25% for **TPF3_64** and to 11% for **TPF1_64**.
- **Runtime Comparison:** Only for WatDiv100M comparing the runtimes of **TPF*_64** to the *Vendor* systems is meaningful due to the higher query success rates. Compared to **ES1_32**, the **TPF1_64** is 2.4 times faster in terms of median runtime and 12% in terms of average runtime. For **TPF3_64** the results are worse than **ES1_32**: 25% slower in median runtime, 40% slower for average runtime.

Horizontal scaling

An alternative to increasing the memory in a single-node server is to increase the overall resources by adding more nodes, thus creating a distributed system.

All 4 commercial RDF solutions support multi-node setups. GraphDB however, works only as a HA-solution (High-Availability): We did not evaluate this approach since it requires all data to be replicated on every node and does not support data partitioning, which is required to scale beyond the single-node resource limits. The performance can however be estimated since it is equivalent to a setup with N identical databases with a load balancer equally distributing the queries between the database replicas. The effect is a linear speedup in terms of completing a full query-mix. Virtuoso also supports a similar setup. The effect on the individual query runtimes should be limited, but not completely absent since the database load on the individual nodes will be smaller. The effect of database load on the query runtimes will be studied in the next section.

For Blazegraph support is required for setting up the multi-node system. This support was requested via the RFI but not fulfilled, which limited our comparison to **Vir3_32_Def**, **ES3_32_Def**, and **TPF3_64_Def**.

In Figure 4 we show pairwise comparisons of the three setups for which we have both a single and a 3-node benchmark.

- **Benchmark survival interval: Vir3_32_Def** and **TPF3_64_Def** managed to stay online during the entire Watdiv1000M benchmark, **ES3_32_Def** stopped responding after having completed 67% of the multi-threaded run.
- **Errors & Timeouts:** 65% of the queries of **ES3_32_Def** resulted in an HTTP 504 error, mentioning *Gateway Timeout*. Further study revealed that this timeout was due to an internal configuration parameter in the ES distributed setup, unfortunately we did not receive any feedback on this issue. **Vir3_32** successfully completed all queries. 70.6% of the queries result in a timeout for **TPF3_64_Def**.
- **Multi-node overhead:** For all setups additional nodes lead to overhead instead of runtime speedup. Runtime multiplication factors are 1.9 and 1.5 for **Vir** and **ES**. **TPF** has a negligible

overhead but is already very close to the query timeout.

In a discussion with OpenLink it was clarified that Virtuoso Cluster acts as a *distributed memory solution*. This implies that adding nodes does not lead to a speedup in the query runtimes, but the total of memory pool in the system increases, allowing it to handle larger datasets for which a single node instance might not be suited. Since the single node benchmark did not exhaust the memory, there is no advantage to be expected from a multi-node setup. As an indication, according to support a 32GB machine should be able to manage up to 3 billion triples (10GB per 1B triples). This observation, together with the lack of feedback on the issues with **ES3_32_Def** and the high timeout percentage for **TPF3_64_Def** motivated our decision to not run any additional benchmarks with this approach for WatDiv1000M.

Systems translating SPARQL queries to distributed platforms such as Hadoop [78, 2] are an alternative approach we did not test. Although these approaches are usually not recommended in a context with low-latency requirements they are specifically designed to operate in an ETL-setting. Results for S2RDF [4], which uses Apache Spark underneath, on Watdiv1000M indicate that a 10-node setup can be close to 10 times faster than a single-node Virtuoso server. Since these SPARQL-on-Hadoop solutions are not sufficiently mature and for example cannot be tested using a SPARQL endpoint definite conclusions can currently not be drawn. One observation to motivate this caution is the fact that Virtuoso is hardly affected when running multiple benchmark clients at once, as will be shown in section 5.2. The operational cost for these Hadoop setups can also not immediately be deduced.

Results II: Query Runtime Analysis in Depth

Only comparing query runtimes might be an oversimplification in benchmark analysis. When comparing the average runtimes of a batch of queries the slow tail of the distributions dominates have the largest impact on the result. In the first subsection we investigate whether certain query templates dominate the average runtimes. Query execution times depend on the state of the database, which motivates studying the query context. Previous results are still valid as all queries are executed 5 times and each time the median is taken to calculate average runtimes.

The next subsection studies the effect of the server load on the query runtimes by comparing a single-client benchmark with a stress test with 5 clients. In the following subsection we investigate the context of a query by studying caching effects.

We conclude by studying an often unreported effect: result completeness can have a big impact on the query runtimes and should always be verified.

Query runtimes for different template types

The queries of the WatDiv benchmarks are all BGPs but have different shapes and selectivity properties. The benchmark generator has 20 templates which can be further organized into 4 template categories (shapes). n Figure 5 we show the average runtime per template for 5 stores on WatDiv1000M.

- **Template timeouts:** For **TPF1_64** 11 runtime averages coincide with the benchmark timeout (300s). Successful queries are spread out over the different types: **F**:3, **L**:2 and **S**:3 . **ES1_64_Def** has timeouts for the 2 **C** queries, 2 **F** queries, and 1 **S** query. The other stores have no averages close to timeout.
- **Template winners: Vir1_64_Opt** is the fastest engine for 12 templates. These are divided as: **C**:1, **F**:2, **S**:4, and all 5 **L**-templates. **Gra1_64_Opt** performs best for 2 **S**-templates, **Bla1_64_Opt** wins on 1 **C**-, 2 **F**- and 1 S-template. Template **C3** was omitted due to query completeness issues. Blazegraph was the only engine to retrieve all results within the timeout boundary.
- **ETL winner:** The most important for the total benchmark time is the **C1**-template which explains why Blazegraph is the fastest in the full ETL-run.

In Figure 6 we demonstrate the impact of different hardware and engine configuration impact the runtime of the individual query runtimes. The improvement in average query runtime is rather homogeneous across the different query categories for all stores. The results in the *Optimized* setting are worse for Virtuoso as configuration was more conservative as compared to the *Default* configuration.

Single versus Multi-client stress testing

All results so far focused on the multi-threaded benchmark run, in which 5 benchmark clients are simultaneously executing the same query-mix in a (different) randomized order. It is however interesting to take into account the effect of server load. In Figure 7 we compare, per query, the runtime of the single-threaded warm-up run versus the runtime of the slowest multi-threaded run. We chose the slowest query as this has the highest probability of eliminating the effects of caching which will be studied in the next section. Note that for the *Prototype* systems the comparison is on WatDiv100M, while the *Vendor* systems are compared on WatDiv1000M.

- **Highest resilience against server loads**: The lowest multiplication factors (mf) are 1.1 , 1.2, and 1.4 for **Vir1_64_Opt**, **Bla1_64_Opt**, and **Fus1_64_Def** respectively.
- **Lowest resilience against server loads:** For **TPF*_64_Def** the mf is 1.8 - 1.9. **Gra1_64_Opt**'s mf is at 2.1, but for **ES1_64_Opt** we have an mf of 4.2.
- **Variance of query runtimes:** For Blazegraph and GraphDB the larger variance on the query runtimes might still be explained as being the result of caching. As we will see in the next section however, we don't observe caching effects for GraphDB and for Blazegraph we only see a weak effect for the slow-running queries (**C**-templates).

The Role of Caching in Query Runtime Results

Some data stores cache the results of queries. Especially in a benchmark where the same query is executed multiple times, this might lead to a large variance on the query runtimes. Although the approach with query events was not designed with support for studying caching effects in mind, having the order of the queries suffices. In an initial attempt we plotted the query runtimes as a function of the distance to their nearest preceding execution. For this distance we experimented with the number of intermediary queries, the total number of intermediary results, and the amount of time in between. Results were very similar but did not show any clear pattern. In Figure 8 however, the speedup with respect to the slowest query execution in the multi-client run is plotted as a function of the actual query runtime. This visualization allows an easy distinction between speedups which are caused by noise, mainly for very short query runtimes, and real caching effects. If no caching effects are present the plot should have all its dots on the X and Y-axis.

- **Stores with clear caching advantage:** The **TPF** server instances have NGINX [79] cache enabled. The similarity in results with other stores strengthens the idea that Figure 8 in fact shows caching behavior for **TPF*_64**, **ES1_64_Def**, and **Gra1_64_Opt**.
- **Caching differences per template type:** For **ES1_64_Def** and **Gra1_64_Opt** the **F**-templates (blue) dots correspond to the highest speedups. For **TPF*_64** query execution is in general slower than for the other systems, therefore **L** and **S**-queries, shift to the right and their speedups become more prominent. Small speedups for **Bla1_64** and **Vir1_64** are mostly limited to the **C**-template queries.
- **TPF1 vs TPF3:** As a result of the horizontal data partitioning scheme **S** and **F**-queries can be resolved locally for **TPF3_64** which explain the higher prevalence in the plot.

Query Result Completeness

In our SWAT4LS contribution [45] we discovered that query runtimes cannot be trusted without paying careful attention to query completeness. We revisited earlier results on WatDiv and discovered some inconsistencies as well.

- **Vir*_Def:** Running Virtuoso with the *Default* configurations gave it an advantage since in this setting the result count is limited to $100,000$. This only affects the **C3** queries for all sizes of WatDiv.
- **Vir*_Opt: Bla1_64_Opt** was the only engine to correctly solve all **C3** templates. This query returns the highest number of results: 42,063,279. Although Virtuoso was configured to report an unlimited number of results, we discovered that for multiple independent queries the result count is limited to the magic number: 1,048,576. (which is 2^{20}).

As mentioned in the introduction of section 4 none of the runtime results reported so far are affected by this query incompleteness as we discarded all queries for which at least one store had a different number of results as compared to the consensus. In practice this means that all **C3** queries had to be discarded. The impact on the runtime comparisons is big as **C3** has the highest runtimes and ignoring query completeness would put Blazegraph at a serious disadvantage.

Results III: Real-world Life Sciences Benchmark Results

The WatDiv benchmark can serve as an initial testing procedure when selecting an appropriate triple store for a certain use case. The ETL case for Ontoforce bears some similarity to the WatDiv benchmark.

The Ontoforce benchmark consists of interactive federated queries which are extracted from the user logs of the DISQOVER product. These queries are currently solved by combining an ETL preprocessing step, which integrates the different Life Sciences datasets offline using Ontoforce's own central ontology. This ETL step bears a lot of similarity with the WatDiv benchmark as it consists of mostly BGP queries. The queries of the Ontoforce benchmark are the result of faceted browsing, whereby, in practice, the facet filters are performed by a distributed search system (SOLR), but their product can also run with a SPARQL-based back-end. In this section we evaluate the ability of *Vendor* systems to work with these types of queries and therefore serve as an alternative to a search system.

The Ontoforce benchmark has a very challenging query set. Therefore the focus of section 6 will be far less on query runtimes but more on trying to extract insights which are generalizable. In this benchmark run the response times consistently coincide with the query runtimes. In subsection title 'Benchmark Error Analysis' we give more detailed insights in the behavior of the different systems on the Ontoforce benchmark. We pay special attention to query failures and query completeness. In the following subsection we try to automatically infer the reasons behind query success, failure, different error types, and slow versus fast running queries. This automation is achieved by making use of decision tree analysis which should circumvent bias introduced by human interpretation. In the final section we compare the results of all benchmarks in this research using *Benchmark Cost* as a unification parameter. This allows to make comparisons between setups which are very different in nature and see whether the trends into the benchmark results are consistent. This approach also takes into account the financial cost for data ingestion and the different licensing fees.

Benchmark Error Analysis

Error Frequencies The *Prototypes* and **Vir*** have been tested on the Ontoforce benchmark for our SWAT4LS [45] contribution. Note that **TPF** systems do not currently support all SPARQL operators and could therefore not be run on this benchmark. In Figure 9 we show the results for the *Vendor* systems. Each simulation consists of a small bar, corresponding to the single-threaded warm-up run, and 5 concatenated bars corresponding to 5 threads in the stress test. The Figure also shows that only Virtuoso simulations had a sufficiently wide benchmark survival interval to enable further analysis.

- **Bla1_64_Opt:** One major difference with the results on the WatDiv benchmark is Blazegraph's inability to handle the complexity of the Ontoforce queries, resulting in very short benchmark survival interval: it contains only 55 queries, of which 18 are timeouts.
- **Gra1_64_Opt:** GraphDB also did not survive the entire benchmark, but managed to stay up for 21% of the stress run. During the stress run it solved 40% of the queries successfully, the other queries resulted in a timeout. For 38% of the queries, at least one successful run is available in the stress run.
- **ES1_64_Def:** ES was definitely the least successful on the WatDiv benchmarks, but is the only store, apart from Virtuoso, for which the benchmark survival interval spans the entire benchmark. 58% of the queries were executed successfully. The remainder consists of 25% HTTP errors and 17% timeouts.
- **Vir1_*_Opt:** Virtuoso is both consistent and successful on this benchmark with only 1% of queries consistently failing, overall the success rate is 98%. These failures correspond mainly to queries which contain *property paths*. None of the other stores could handle these queries. It should be noted that during the creation of the DISQOVER product, Virtuoso was frequently used as a back-end system, which partially implies a certain favorable bias in the Ontoforce results.The **Vir1_32_Opt** in the SWAT4LS [45] paper had 41% incomplete queries. This re-run however, achieves the same figures as the 64GB run.
- **Vir3_64_Opt_*:** The **Vir3_64_Opt** setup was re-run multiple times, the different runs are identified with an additional sequence number 0-2: Although the success rate of **Vir3_64_Opt_0** is only 55%, 92% of the queries are successfully executed at least once, which makes it possible to make runtime comparisons. **Vir3_64_Opt_2** has far less reported errors. Post-processing revealed issues with query completeness (orange) for 37% of the queries.

Query Correctness. Previously published results [45] had counter-intuitive runtime results: **Vir1_32** and **Vir3_64_Opt_2** executed much faster than **Vir1_64**. Consequently, we studied the number of results per query:

- **Inter-thread consistency:** As a first step we analyzed whether for each individual system the number of query results was consistent for each query-mix. Without any exception this inter-thread consistency was confirmed.

- **Query consensus:** In the query event format, described in Table 5, one field indicates whether a query is correct or its result count incomplete. These values are obtained by creating a query consensus, with the following rules. If at least two separate *Vendor* systems agreed on the number of results we assume this results is 'correct', for 97.3% this is the case. If only 1 engine can solve a query we label these as 'uncertain'. Virtuoso solves 19 queries for which no consensus can be derived. For 13 queries none of the systems managed to generate a solution. 8 of these contain a property path operator, the other 5 have FILTER IN operators containing large URI lists, such that the file size of the query is between 10 and 100 kB.
- **Count Queries:** Of the 19 'uncertain' queries solved by Virtuoso 15 are COUNT queries. However, upon inspection the COUNT operator was always part of a sub-query, so this result can not be disproven. The benchmark software only reports the number of results per query. We extended it to also download the actual results to be able to verify whether the COUNT queries are consistent between the stores. However, no inconsistencies were found there.
- **Incorrect Query results:** Some of the Virtuoso benchmarks have incorrect results. The typical pattern is that the query is executed $< 1s$ and generates 0 results. 1 query also had the query result limit $= 2^{20}$. To get more insight into the context of incomplete queries we executed the **Vir3_64** benchmark an additional 3 times. In these runs the incorrect query results were not observed, but, but the new benchmarks never made it to the stress test, with the best run having a benchmark survival interval with a length of 228 queries.

Decision Tree Analysis of Query Features

Ontoforce has released the queries for this benchmark run. However, the queries are very complex and sometimes they take up 1 - 100 kB in disk space. To gain a deeper understanding into why queries fail, have timeouts and HTTP errors, why they are fast or slow to execute,... we created a set of features per query and fitted a decision tree [80] to the data. The 3 resulting trees are shown in Figure 10. We perform manual feature selection on the input features by removing highly correlated features. For example ORDER and LIMIT are highly correlated. The list of retained query features is given in Table 7 together with the highest correlated operators. By adding 'Query Engine' as an additional feature we can train the decision tree on all the available query event data for the Ontoforce Benchmark for all RDF solutions at the same time.

- **Dominant Feature:** The 'Query Engine' is the most important factor to segment the data in all 3 cases. The absence of this feature would in fact indicate that all systems have similar behavior. (we tested without this feature but the decision trees were not informative) **Vir1** thus is very different: it has fewer errors and query runtimes are significantly smaller.
- **Feature Importance:** If we take the number of node occurrences as a feature as a measure for feature importance then we see 3 features which occur in 5 nodes: TP, FILTER IN, FILTER. The FILTERS mainly play a role in the decision tree for runtime regression. In predicting failures and error types OPTIONAL, GRAPH and Q have the highest occurrences.
- **Highest failure rates:** The paths leading to samples with a high failure rate generally contain OPTIONAL operators. All engines except for **Vir1** suffer when $Q > 1$. **Gra1** also has a high failure rate for COUNT queries.
- **Most frequent error types:** For **Bla1** and **Gra1** the errors are all timeouts (purple). For **ES1** having multiple subqueries often leads to HTTP errors (green).
- **Queries with high runtimes:** For **Vir1** and **ES1** the FILTER IN operators are the main cause for high runtimes. For **Gra1** the presense of FILTERS pushes runtimes above 100s.

Finally we also investigate if the incorrect queries in the **Vir3** benchmarks had specific query features. Curiously, the problematic queries correspond to the most simple queries: $TP \leq 2$.

Benchmark Cost

In this section we aim to get a satellite view on the entire set of benchmarks conducted within this research. The penultimate trade-off for many applications in production is the financial cost for processing a certain workload. Our choice for using cloud hardware and AMIs enables this integrated view on all benchmarks: using cost we can compare single to multi-node setups, the cost for vertical scaling,...

All financial costs per store and for all benchmarks are shown in Figure 11. Costs stem from an hourly price for servers on Amazon EC2, together with an hourly license cost for the AMIs.

Table 7: Query Features and information on their range and correlations with other (discarded) features.

Feature	Prefix	Value	Range	Correlations
ORDER	ORD	frequency	[0,1]	LIMIT(0.88)
FILTER IN	FIL_IN	frequency	[0,16]	UNION(0.95), FS(0.95)
FILTER	FIL	frequency	[0,27]	tp_???(0.96), TP(0.95)
COUNT	CNT	frequency	[0,1]	DISTINCT(1.0)
Triple Patterns	TP	frequency	[1,38]	FILTER(0.95)
GRAPH	GRA	frequency	[0,1]	-
OPTIONAL	ORD	frequency	[0,9]	-
GROUP	GRP	frequency	[0,4]	-
(sub)Queries	Q	frequency	[1,10]	UNION(0.94), FILTER IN(0.94)
file size	FS	kilobyte	1, 10, 100	FILTER IN(0.97), UNION(0.95)
query engine	-	*Vendor*	-	-

The instance cost of the AWS hardware was $0.333 /hr for the 32GB server instances and $0.667 /hr for the 64GB instances. The licensing costs for the PAGO instances can be found on AWS marketplace and typically scale with the amount of memory per instance. For the 64GB instances, GraphDB's license cost is $1.4 /hr, for ES $2 /hr and for Virtuoso $0.80 /hr. Other systems tested have no licensing cost.

Additionally, before running a benchmark the data has to be ingested in the system. This cost is stacked on top of the query cost in Figure 11. For some cases the ingest cost is unimportant as reloading the data is required only rarely.

- **The price of vertical scaling:** Is adding more memory, and therefore a higher license and infrastructure cost a wise choice? If we focus on the *Optimized* configurations for Watdiv1000M both **Bla1** and **Gra1** have lower operational costs when running the higher end hardware. For **Bla1** the price is lowered from $27 to $13.5, for **Gra1** the reduction is from $298 to $230. For the latter mainly the bulk loading process makes it less competitive. For **Vir1** the price goes from $5 to $7.
- **The price of horizontal scaling:** As adding more nodes led to higher runtimes, this also translates to higher costs. For **TPF** the costs go from $168 to $323 ($\times$1.9) , for **ES** the costs rises from $112 to $475 ($\times$4.2) and for **Vir** from $5 to $42 ($\times$8.4)
- **The price for data ingestion: Gra1** seems to have the highest cost for loading the datasets, except for the Ontoforce benchmark. This is interesting as the Ontoforce benchmark has a much bigger dataset (2.4 BT). A possible explanation is that **Gra1** has trouble ingesting a single gzipped turtle file as was the case for WatDiv, while the Ontoforce dataset was ingested as 42 gzipped N-Quads files. For **Gra1_64_Def** many additional indexes are generated during ingest, which explains the lower cost for **Gra1_64_Opt**. Having more memory by itself can also impact the ingest process, for **Bla1** the ingest cost is lowered from $16 to $12. Virtuoso's bulk loader process is a real trump card in the cost comparisons. The load cost is $2.8 while **Bla1** in the optimal case has a cost of $12.6. The load cost is in fact larger than the runtime cost in this comparison. Also for the multi-node setups no advantage is obtained in the ingest phase. **Vir3** takes 4 times more time to ingest while the cost/hr is also 3 times higher. For **ES3** a 33% cost increase is measured, while for **TPF3** the ingestion becomes 50% cheaper. The latter however is not **TPF** specific as the ingestion corresponds to the partitioning and compression of the data with the HDT algorithm (for which we used a 128 GB high-memory infrastructure).
- **The most cost-effective solution: Vir1_32** is the cheapest solution both for WatDiv1000M as for the Ontoforce benchmark, with costs of respectively $5 and $19.

Conclusion

In this work we offer guidelines and tools to run a *reproducible* benchmark (**RQ1**). For the back-end we recommend working with hardware available via cloud providers, AMIs and Docker images for the system installation. We recommend releasing the configuration details for every store.

To enable critical reviewing benchmark output data should be released in its rawest form. The query event data in this work turned out to be an enabler for new unanticipated research questions. One example in this work is the study of caching effects.

The methods to arrive at certain research visualizations should be made available, which also provides knowledge transfer to future benchmark efforts.

In order to learn from challenging benchmarks, the benchmark approach should anticipate the occurrence of all sorts of errors. The information in these incomplete benchmark runs is in fact very valuable.

What are the trade-offs associated with certain setups (**RQ2**)? For every store we show the effect in terms of throughput and cost for vertical and horizontal scaling. Overall, the low-end setup **Vir1_32** gave the best results. For the other stores the best results are achieved with more memory and with *Optimized* configurations provided by the vendors.

Benchmark cost allows the comparison of a heterogeneous mix of RDF storage approaches. Research *Prototypes*, of which **TPF*_64** performed best in this study, still lag by an order of magnitude in terms of performance with the *Vendor* systems. The research community would benefit from more realistic and challenging benchmarks, as it might stimulate the further development of current prototypes up to a level where they can compete with existing *Vendor* systems.

Future benchmarking efforts should consider, at least locally, scanning a benchmark space. This necessity was demonstrated in this work by showing the effect of dataset size and by modifying the amount of server memory. An interesting result in this aspect is that the performance of the different *Vendor* systems converged as they were given better hardware and configurations.

In answering **RQ3** we demonstrated that query runtimes are an oversimplified representation of performance. Many contextual factors influence the runtime comparisons: certain query types might completely dominate the runtime averages, server load and caching effects have a different impact on the systems tested. Adding query completeness analysis, makes benchmark runtime results more trustworthy. Ignoring this aspect would have led to very different conclusions in this work.

The ranking of the different systems is not consistent if we change from artificial to real-world benchmarks (**RQ4**). This supports the advice to try and run use case specific benchmarks before deciding on a system architecture. Although it was difficult to extract transferable insights from the Ontoforce benchmark, the decision tree approach in fact shed some light on certain SPARQL query features which pose more problems to one system than another, giving the vendors some direction in optimizing their RDF storage solution. Due to the automated way of inferring these insights they can be more easily compared to other benchmarking efforts.

As for the future work, the results in this work definitely indicate a lot of room for improvement in multi-node RDF storage solutions. While Virtuoso's offering is the most advanced, it is not yet as stable as its single-node counterpart.

Ethics approval and consent to participate
Not applicable

Consent for publication
Not applicable

Availability of data and materials
onderstaande wat tunen All data generated or analysed during this study are included in this published article [and its supplementary information files]. The data that support the findings of this study are available from Ontoforce but restrictions apply to the availability of these data, which were used under license for the current study, and so are not publicly available. Data are however available from the authors upon reasonable request and with permission of [third party name].

Competing interests
Some of the co-authors in this research paper have contributed to the TPF implementation. (LDV and RV) Te other authors have no competing interests.

Funding
The research activities were funded by VLAIO (the Agency for Innovation and Entrepreneurship in Flanders) in an R&D project called SEQUEL with Ontoforce, Ghent University and imec (iMinds).

Author's contributions
The work presented in this paper was conducted in collaboration between all authors. DDW and LDV conducted the experiments. DDW analyzed the data and drafted the paper. FP, KK and HC helped defining a relevant Life Sciences use case and provided datasets and queries. JF, RV and EM revised the research paper and coordinated the research process. All authors have approved the final version of the manuscript.

Acknowledgements
We would like to acknowledge iLab.t who provided the high memory infrastructure to compress the benchmark datasets and a set of servers to run additional benchmarks.
We would also like to express our gratitude for the support provided by the staff working for the 4 vendors (Blazegraph, GraphDB, Virtuoso, FluidOps) we were allowed to disclose and to Rob Vesse who provided support for the SPARQL Query Benchmarker software.

Author details
[1]imec - IDLab - Ghent University, iGent Tower - Technologiepark-Zwijnaarde 15, BE 9052 Ghent, Belgium. [2]Ontoforce, Technologiepark-Zwijnaarde 19, BE 9052 Ghent, Belgium.

References

1. Bizer, C., Heath, T., Ayers, D., Raimond, Y.: Interlinking open data on the web. In: Demonstrations Track, 4th European Semantic Web Conference, Innsbruck, Austria (2007)
2. Graux, D., Jachiet, L., Genevès, P., Layaïda, N.: A multi-criteria experimental ranking of distributed SPARQL evaluators (2016)
3. Ladwig, G., Harth, A.: CumulusRDF: linked data management on nested key-value stores. In: The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2011), p. 30 (2011)
4. Schätzle, A., Przyjaciel-Zablocki, M., Skilevic, S., Lausen, G.: S2RDF: RDF querying with SPARQL on spark. Proc. VLDB Endow. **9**(10), 804–815 (2016)
5. Schätzle, A., Przyjaciel-Zablocki, M., Lausen, G.: PigSPARQL: Mapping SPARQL to pig latin. In: Proceedings of the International Workshop on Semantic Web Information Management, p. 4 (2011). ACM

6. Kotsev, V., Minadakis, N., Papakonstantinou, V., Erling, O., Fundulaki, I., Kiryakov, A.: Benchmarking RDF query engines: The LDBC semantic publishing benchmark
7. Hernández, D., Hogan, A., Riveros, C., Rojas, C., Zerega, E.: Querying wikidata: Comparing SPARQL, relational and graph databases. In: International Semantic Web Conference, pp. 88–103 (2016). Springer
8. Harris, S., Lamb, N., Shadbolt, N.: 4store: The design and implementation of a clustered RDF store. In: 5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009), pp. 94–109 (2009)
9. Papailiou, N., Konstantinou, I., Tsoumakos, D., Karras, P., Koziris, N.: H2rdf+: High-performance distributed joins over large-scale RDF graphs. In: Big Data, 2013 IEEE International Conference On, pp. 255–263 (2013). IEEE
10. Schätzle, A., Przyjaciel-Zablocki, M., Neu, A., Lausen, G.: Sempala: interactive SPARQL query processing on hadoop. In: International Semantic Web Conference, pp. 164–179 (2014). Springer
11. Rohloff, K., Schantz, R.E.: High-performance, massively scalable distributed systems using the mapreduce software framework: the shard triple-store. In: Programming Support Innovations for Emerging Distributed Applications, p. 4 (2010). ACM
12. Saleem, M., Hasnain, A., Ngomo, A.-C.N.: BigRDFBench: A billion triples benchmark for SPARQL endpoint federation
13. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization Techniques for Federated Query Processing on Linked Data. In: The Semantic Web - ISWC 2011, Proceedings, Part I, pp. 601–616 (2011)
14. Görlitz, O., Staab, S.: Splendid: SPARQL endpoint federation exploiting void descriptions. In: Proceedings of the Second International Conference on Consuming Linked Data-Volume 782, pp. 13–24 (2011). CEUR-WS. org
15. Acosta, M., Vidal, M.-E., Lampo, T., Castillo, J., Ruckhaus, E.: In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints, pp. 18–34. Springer, Berlin, Heidelberg (2011)
16. Saleem, M., Ngomo, A.-C.N.: Hibiscus: Hypergraph-based source selection for SPARQL endpoint federation. In: European Semantic Web Conference, pp. 176–191 (2014). Springer
17. Saleem, M., Mehmood, Q., Ngomo, A.-C.N.: Feasible: a feature-based SPARQL benchmark generation framework. In: International Semantic Web Conference, pp. 52–69 (2015). Springer
18. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified stress testing of rdf data management systems. In: International Semantic Web Conference, pp. 197–212 (2014). Springer
19. Boncz, P.A., Zukowski, M., Nes, N.: MonetDB/X100: Hyper-pipelining query execution. In: CIDR, vol. 5, pp. 225–237 (2005)
20. Neumann, T., Weikum, G.: The RDF-3X engine for scalable management of RDF data. The VLDB Journal - The International Journal on Very Large Data Bases **19**(1), 91–113 (2010)
21. Zou, L., Mo, J., Chen, L., Özsu, M.T., Zhao, D.: gstore: answering SPARQL queries via subgraph matching. Proceedings of the VLDB Endowment **4**(8), 482–493 (2011)
22. Cudré-Mauroux, P., Enchev, I., Fundatureanu, S., Groth, P., Haque, A., Harth, A., Keppmann, F.L., Miranker, D., Sequeda, J.F., Wylot, M.: NoSQL databases for RDF: an empirical evaluation. In: International Semantic Web Conference, pp. 310–325 (2013). Springer
23. Khadilkar, V., Kantarcioglu, M., Thuraisingham, B., Castagna, P.: Jena-HBase: a distributed, scalable and efficient RDF triple store. In: Proceedings of the 2012th International Conference on Posters & Demonstrations Track-Volume 914, pp. 85–88 (2012). CEUR-WS. org
24. Wu, H., Fujiwara, T., Yamamoto, Y., Bolleman, J., Yamaguchi, A.: BioBenchmark Toyama 2012: an evaluation of the performance of triple stores on biological data. Journal of biomedical semantics **5**(1), 1 (2014)
25. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: Fedbench: A benchmark suite for federated semantic data query processing. In: International Semantic Web Conference, pp. 585–600 (2011). Springer
26. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Web Semantics: Science, Services and Agents on the World Wide Web **3**(2), 158–182 (2005)
27. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SPˆ 2Bench: a SPARQL performance benchmark. In: Data Engineering, 2009. ICDE'09. IEEE 25th International Conference On, pp. 222–233 (2009). IEEE
28. Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. International Journal on Semantic Web and Information Systems (IJSWIS) **5**(2), 1–24 (2009)
29. Duan, S., Kementsietsidis, A., Srinivas, K., Udrea, O.: Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 145–156 (2011)
30. Morsey, M., Lehmann, J., Auer, S., Ngonga Ngomo, A.-C.: DBpedia SPARQL benchmark–performance assessment with real queries on real data. The Semantic Web–ISWC 2011, 454–469 (2011)
31. Antezana, E., Egaña, M., Blondé, W., Illarramendi, A., Bilbao, I., De Baets, B., Stevens, R., Mironov, V., Kuiper, M.: The cell cycle ontology: an application ontology for the representation and integrated analysis of the cell cycle process. Genome Biology **10**(5), 58 (2009)
32. Yamamoto, Y., Yamaguchi, A., Bono, H., Takagi, T.: Allie: a database and a search service of abbreviations and long forms. Database **2011** (2011)
33. Kinjo, A.R., Suzuki, H., Yamashita, R., Ikegawa, Y., Kudou, T., Igarashi, R., Kengaku, Y., Cho, H., Standley, D.M., Nakagawa, A., et al.: Protein data bank japan (pdbj): maintaining a structural data archive and resource description framework format. Nucleic acids research (2011)
34. Consortium, U., et al.: Uniprot: a hub for protein information. Nucleic acids research (2014)
35. Tateno, Y., Imanishi, T., Miyazaki, S., Fukami-Kobayashi, K., Saitou, N., Sugawara, H., Gojobori, T.: Dna data bank of japan (ddbj) for genome scale research in life science. Nucleic acids research **30**(1), 27–30 (2002)
36. Görlitz, O., Thimm, M., Staab, S.: Splodge: systematic generation of SPARQL benchmark queries for linked open data. In: International Semantic Web Conference, pp. 116–132 (2012). Springer
37. Ghemawat, S., Gobioff, H., Leung, S.-T.: The google file system. In: ACM SIGOPS Operating Systems Review, vol. 37, pp. 29–43 (2003). ACM
38. Angles, R., Boncz, P., Larriba-Pey, J., Fundulaki, I., Neumann, T., Erling, O., Neubauer, P., *et al.*: The linked data benchmark council: A graph and RDF industry benchmarking effort. SIGMOD Rec. **43**(1), 27–31 (2014)
39. Ngomo, A.-C.N., Röder, M.: HOBBIT: Holistic benchmarking for big linked data
40. Boncz, P.: LDBC: Benchmarks for graph and RDF data management. In: Proceedings of the 17th International Database Engineering & Applications Symposium. IDEAS '13, pp. 1–2. ACM, New York, NY, USA (2013)
41. Erling, O., Averbuch, A., Larriba-Pey, J., Chafi, H., Gubichev, A., Prat, A., Pham, M.-D., Boncz, P.: The LDBC social network benchmark: Interactive workload. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 619–630 (2015). ACM
42. De Witte, D., De Vocht, L., Verborgh, R., Mannens, E., *et al.*: Big linked data etl benchmark on cloud commodity hardware. In: Proceedings of the International Workshop on Semantic Big Data, p. 12 (2016). ACM
43. Ontoforce - Powering Citizen Data Science. (Accessed August 31, 2018). http://www.ontoforce.com
44. DISQOVER. (Accessed August 31, 2018). https://www.disqover.com/
45. De Witte, D., De Vocht, L., Knecht, K., Pattyn, F., Constandt, H., Mannens, E., Verborgh, R.: Scaling out federated queries for life science data in production. In: Proceedings of the 9th International Conference on Semantic Web Applications and Tools for Life Sciences (2016)
46. Gallego, M.A., Fernández, J.D., Martínez-Prieto, M.A., de la Fuente, P.: An empirical study of real-world sparql queries. In: USEWOD Workshop (2011)
47. Stegmaier, F., Gröbner, U., Döller, M., Kosch, H., Baese, G.: Evaluation of current RDF database solutions. In: Proceedings of the

10th International Workshop on Semantic Multimedia Database Technologies (SeMuDaTe), 4th International Conference on Semantics And Digital Media Technologies (SAMT), pp. 39–55 (2009). Citeseer

48. DB-Engines Ranking - Popularity Ranking of Database Management Systems. (Accessed August 31, 2018). http://db-engines.com/en/ranking
49. Confirm Selection of Blazegraph for Wikidata Query. (Accessed August 31, 2018). https://phabricator.wikimedia.org/T90101
50. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF representation for publication and exchange (HDT). Journal of Web Semantics **19**, 22–41 (2013)
51. Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., Cyganiak, R., Colpaert, P., Mannens, E., Van de Walle, R.: Querying datasets on the web with high availability. In: International Semantic Web Conference, pp. 180–196 (2014). Springer
52. Veritas - The Leader in Enterprise Data Protection. (Accessed August 31, 2018). https://www.veritas.com/?ref=fluidops
53. HDT-Fuseki. (Accessed August 31, 2018). https://github.com/rdfhdt/hdt-java/tree/master/hdt-fuseki
54. Verborgh, R.: Github - LinkedDataFragments/Server.js A Triple Pattern Fragments Server for JavaScript. (Accessed August 31, 2018). https://github.com/LinkedDataFragments/Server.js
55. Verborgh, R.: Github - LinkedDataFragments/Client.js A JavaScript Client for Triple Pattern Fragments Interfaces. (Accessed August 31, 2018). https://github.com/LinkedDataFragments/Client.js
56. Zeng, K., Yang, J., Wang, H., Shao, B., Wang, Z.: A distributed graph engine for web scale RDF data. Proc. VLDB Endow. **6**(4), 265–276 (2013)
57. Harth, A., Umbrich, J., Hogan, A., Decker, S.: In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) YARS2: A Federated Repository for Querying Graph Structured Data from the Web, pp. 211–224. Springer, Berlin, Heidelberg (2007)
58. Ferré, S., Hermann, A., Ducassé, M.: Semantic Faceted Search: Safe and Expressive Navigation in RDF Graphs. Research Report PI 1964 (January 2011). https://hal.inria.fr/inria-00554093
59. Oren, E., Delbru, R., Decker, S.: In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) Extending Faceted Navigation for RDF Data, pp. 559–572. Springer, Berlin, Heidelberg (2006)
60. Conjunctive Query - Wikipedia. (Accessed August 31, 2018). https://en.wikipedia.org/wiki/Conjunctive_query#Relationship_to_other_query_languages
61. Picalausa, F., Vansummeren, S.: What are real SPARQL queries like? Proceedings of the International Workshop on Semantic Web Information Management - SWIM '11, 1–6 (2011)
62. Loizou, A., Angles, R., Groth, P.: On the formulation of performant {SPARQL} queries. Web Semantics: Science, Services and Agents on the World Wide Web **31**, 1–26 (2015)
63. Amazon EC2 Instance Types. (Accessed August 31, 2018). https://aws.amazon.com/ec2/instance-types/
64. AWS Marketplace. (Accessed August 31, 2018). https://aws.amazon.com/marketplace/
65. Docker Hub. (Accessed August 31, 2018). https://hub.docker.com/
66. AWS Marketplace - Virtuoso Universal Server 7.x (Enterprise – Cloud PAGO Edition). (Accessed August 31, 2018). https://aws.amazon.com/marketplace/pp/B011VMCZ8K
67. AWS Marketplace - GraphDB Cloud V7.0.1. (Accessed August 31, 2018). https://aws.amazon.com/marketplace/pp/B00OM7VXGW
68. Linkeddatafragments/server.js - Docker Hub. (Accessed August 31, 2018). https://hub.docker.com/r/linkeddatafragments/server.js/
69. Rfdhdt/hdt-cpp - Docker Hub. (Accessed August 31, 2018). https://hub.docker.com/r/rfdhdt/hdt-cpp/builds/
70. Tenforce/virtuoso - Docker Hub. (Accessed August 31, 2018). https://hub.docker.com/r/tenforce/virtuoso/
71. Laurensdv/docker-blazegraph. (Accessed August 31, 2018). https://github.com/laurensdv/docker-blazegraph/tree/master/2.1.2
72. Saleem, M., Khan, Y., Hasnain, A., Ermilov, I., Ngonga Ngomo, A.-C.: A fine-grained evaluation of SPARQL endpoint federation systems. Semantic Web **7**(5), 493–518 (2016)
73. SPARQL Query Benchmarker - Sourceforge. (Accessed August 31, 2018). https://sourceforge.net/projects/sparql-query-bm/
74. Detailed Overview All Benchmarks. (Accessed August 31, 2018). http://users.elis.ugent.be/drdwitte/resultscsv.html
75. Project Jupyter. (Accessed August 31, 2018). http://jupyter.org/
76. Postprocessing: Jupyter Notebooks. (Accessed August 31, 2018). http://users.elis.ugent.be/drdwitte/postprocessing.html
77. Howard, P.: Blazegraph GPU. (Accessed August 31, 2018). https://www.blazegraph.com/whitepapers/Blazegraph-gpu_InDetail_BloorResearch.pdf
78. Curé, O., Naacke, H., Baazizi, M.A., Amann, B.: On the evaluation of RDF distribution algorithms implemented over apache spark (2015)
79. NGINX - High Performance Load Balancer, Web Server and Reverse Proxy. (Accessed August 31, 2018). https://www.nginx.com/
80. Decision Trees - Scikit-Learn. (Accessed August 31, 2018). http://scikit-learn.org/stable/modules/tree.html

Figures

Figure 1: Query runtime distributions of *Vendor* systems for 3 different sizes of WatDiv. Dots correspond to average runtimes, while the horizontal lines in the box plots correspond to median runtimes. The difference is scaling behavior between **Vir_32** (linear) and the other stores emphasizes the different impact of server memory on runtime behavior. **Bla1_32** and **Gra1_32** are very close in terms of average runtimes, for individual queries GraphDB is superior except when scaling up to WatDiv1000M. **ES1_32** is the only store with timeout problems starting from WatDiv100M.

Figure 2: Query runtime distributions for WatDiv1000M showing the effect of increasing memory from 32GB (left) to 64GB (center) and *Optimized* configurations (right). Virtuoso hardly doesn't benefit from additional memory or better configurations. GraphDB is the most sensitive to proper configuration. In the right panel engine performance starts converging. In terms of average runtimes **Bla1_64_Opt** is the fastest, in terms of median runtimes both **Vir1_64_*** setups perform best.

Figure 3: Benchmark survival interval for 3 *Prototypes*. For early crashes the amount of queries until system failure is reported, as well as the query template causing the failure. **FedX3_64** crashes upon the first occurrence of a **C3** query. **Fus1_64** survives the warm-up run for WatDiv100M but crashes upon the first occurrence of a **C2** query in the stress test, for WatDiv1000M again the first **C2** query in the warm-up run causes the crash.

Figure 4: Pairwise comparison of query runtime distributions for single-node versus 3-node setups. None of the solutions achieve an average runtime speedup when adding more nodes, on the contrary overhead multipication factors of 1.9 and 1.5 are seen in left and center pane for **Vir3_32_Def** and **ES3_32_Def**. For **TPF3_64_Def** the overhead is negligible.

Figure 5: Average Runtime per query template for 5 single-node setups. Results are shown for 19 query templates in 4 template categories (C,F,L,S). Transparent bars show the runtime average per category. Arrows point at the smallest runtime for a certain query template. In the categories **C** and **F** Blazegraph performs best. Virtuoso is the winner for the **L** and **S** categories. Blazegraph is the fastest for 5 templates, Virtuoso for 12 templates and GraphDB for 2 **S**-templates. Template **C3** was omitted due to query completeness issues. Blazegraph was the only engine to retrieve all results.

Figure 6: Average runtime per query template for different engine hardware and configurations. For every engine and for every template category the effect of improving hardware and configuration is consistent and uniform. The only exception being Virtuoso for which the *Optimized* setting performs worse as compared to the *Default* setting.

Figure 7: Runtimes for single versus multi-client workloads: 1 vs. 5 threads. 5T runtime corresponds to the maximum runtime per query in the stress test, 1T is the runtime during the warm-up phase. The red line corresponds to the bisector, where the runtime for both workloads is equal. Dots are expected to be shifted up, which correspond to a multiplication factor. The closer the dots to the bisector the smaller the multi-client overhead. Dots below the bisector can be attributed to the natural variance in query runtimes. Average runtimes per store are also shown. **Bla1_64** and **Vir1_64** have the smallest overhead ($< 20\%$), for **ES1_64** has the largest ($> 300\%$).

Figure 8: Speedup in query runtime. We compare query runtimes in the multi-threaded run with slowest version of the query execution. With no caching all dots are expected on the X and Y-axis, the latter because of the noise on small query runtimes. If we focus on speedups > 2, especially **ES1** and **TPF*** seem to have the highest benefit.

Figure 9: Overview of successes and errors per query (Y-axis) and thread (X-axis) on the Ontoforce benchmark. Queries are sorted per system in order to group error behavior and are not consistent between simulations! Blazegraph has a short benchmark survival interval. **ES1**, **Gra1** and **Vir3** Cluster setups have a lot of errors but most queries execute successfully at least once, which allows runtime comparisons. **Vir3_64** was re-run multiple times and labeled with an additional index: **Vir3_64_Opt_0** is the most successful Virtuoso cluster run as query completeness analysis revealed that **Vir3_64_Opt_2** has unreported errors for 37% of the queries.

Figure 10: Decision Tree Analysis to identify the reason for query failure, certain error types, and high/low query runtimes. Input for all trees are feature vectors, also the query engine is added as a categorical feature. Rules in the decision trees are shown in red, sample sizes are encoded as the width of the bottom bar and the value is added inside the bars in bold. For each separate part the class distribution or the average runtime is reported below the bar.
<ins>Top:</ins> Classification into query success (blue) and failure (red) and incomplete. The query engine is an important decision rule, which demonstrates that Virtuoso behaves very different from the other systems.
<ins>Center:</ins> Classification of query failures into classes incomplete (orange), server error (green), and timeout (purple).
<ins>Bottom:</ins> Regression on query runtimes. Red corresponds to high query runtimes, white to low.

Additional Files

Sequel Project Website

The website links to all material related to this manuscript: datasets, notebooks with analysis, raw log files,. . . can be found here: http://users.elis.ugent.be/~drdwitte/index.html

Feature Matrix

Overview om Semantic Databases considered in this benchmark with together with a set of features and links where this information was found: http://users.elis.ugent.be/~drdwitte/featurematrix.html

Notebooks and CSV files for postprocessing

All CSV files with different views on the benchmark output together with the Jupyter notebook files showing the original analysis of the data can be found here: http://users.elis.ugent.be/~drdwitte/postprocessing.html

Figure 11: Benchmark Cost in $ to load and execute 2000 queries in a stress test for WatDiv1000M or Ontoforce datasets for different setups. All stacked bars consists the load cost stacked on top of the runtime cost. Bar width encodes the amount of nodes. For WatDiv **Vir1_32_Def** is the least expensive solution, mainly because **Bla1_64_Opt** has a much higher load cost. Also for the Ontoforce benchmark **Vir1_32_Opt** is the most cost-effective choice. The engine ranking is not conserved going from artificial to real-world benchmark.