Diversified Stress Testing of RDF Data Management Systems

Güneş Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee

David R. Cheriton School of Computer Science, Waterloo, ON, Canada
{galuc,ohartig,tamer.ozsu,kdaudjee}@uwaterloo.ca

Abstract. The Resource Description Framework (RDF) is a standard for conceptually describing data on the Web, and SPARQL is the query language for RDF. As RDF data continue to be published across heterogeneous domains and integrated at Web-scale such as in the Linked Open Data (LOD) cloud, RDF data management systems are being exposed to queries that are far more diverse and workloads that are far more varied. The first contribution of our work is an in-depth experimental analysis that shows existing SPARQL benchmarks are not suitable for testing systems for diverse queries and varied workloads. To address these shortcomings, our second contribution is the Waterloo SPARQL Diversity Test Suite (WatDiv) that provides stress testing tools for RDF data management systems. Using WatDiv, we have been able to reveal issues with existing systems that went unnoticed in evaluations using earlier benchmarks. Specifically, our experiments with five popular RDF data management systems show that they cannot deliver good performance uniformly across workloads. For some queries, there can be as much as five orders of magnitude difference between the query execution time of the fastest and the slowest system while the fastest system on one query may unexpectedly time out on another query. By performing a detailed analysis, we pinpoint these problems to specific types of queries and workloads.

Keywords: RDF, SPARQL, systems, benchmarking, workload diversity.

1 Introduction

With the proliferation of very large, heterogeneous RDF datasets such as those in the Linked Open Data (LOD) cloud [6], there is increasing demand for high-performance RDF data management systems. A number of such systems have been developed; however, as queries executed on these systems become increasingly more diverse [4], [10], [16], these systems have started to display unpredictable behaviour, even to the extent that on some queries they time out (cf., Fig. 4). This behaviour is not captured by existing benchmarks. The problem is that data that are handled by these RDF data management systems have become far more heterogeneous [10], and web applications that are supported by these systems have become far more varied [4], [16], but existing benchmarks do not have the corresponding variability in their datasets and workloads. Consequently, problems go undetected in evaluations using existing benchmarks until systems are actually deployed. To address these shortcomings, we have designed the Waterloo SPARQL Diversity Test Suite (WatDiv) that offers stress testing tools to reveal a much wider range of problems with RDF data management systems.

P. Mika et al. (Eds.) ISWC 2014, Part I, LNCS 8796, pp. 197–212, 2014.

Our contributions with WatDiv and the work leading up to its design can be summarized in three steps. First, we introduce two classes of query features, namely, structural (cf., Section 2.1) and data-driven features (cf., Section 2.2) that should be used to evaluate the variability of the datasets and workloads in a SPARQL benchmark. More specifically, with these features we differentiate as much as possible those types of queries that may result in unpredictable system behaviour and are indicators of potential flaws in physical design. For example, in a previous work, we illustrate that the choice of physical design in an RDF system is very sensitive to the types of joins that the system can efficiently support [2]. Hence, we introduce a feature called "join vertex type". Likewise, we note that a system's performance depends on the characteristics of the data as much as the query itself. Consequently, we introduce additional features that capture multiple notions of selectivity and result cardinality.

Second, we have performed an in-depth analysis on existing SPARQL benchmarks using the two classes of query features that we introduce. Our experimental evaluation demonstrates that no single benchmark (including those that are based on actual query logs) is sufficiently varied to test whether a system has consistently good performance across diverse workloads (cf., Section 3). Furthermore, these benchmarks do not provide the tools to localize problems to specific types of queries if needed. For example, it would be useful if one could diagnose that the system under test has problems with queries that have a particular join vertex type, and drill down the evaluation if necessary. These are exactly the type of evaluations that we aim to facilitate with WatDiv.

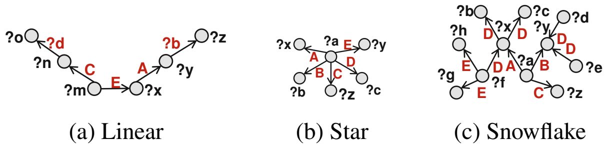
Third and last, we use WatDiv to experimentally evaluate five popular RDF data management systems (cf., Section 5). Our discussion consists of two parts. First, we demonstrate that evaluations using a diverse workload can help reveal problems about systems that existing benchmarks cannot identify. Second, we show that by analyzing results across one or more structural and data-driven features, it is possible to diagnose and reason about specific problems—a feature not supported by any other benchmark.

2 Preliminaries

This section defines query features based on which we shall discuss the diversity of SPARQL benchmark workloads. These features can be categorized into two classes: (i) structural features and (ii) data-driven features. We assume the reader is familiar with the RDF data model [21] and the SPARQL query language [14].

We define these features over a basic fragment of SPARQL, namely, basic graph patterns (BGPs) with filter expressions. For the sake of brevity, we denote queries in this fragment by a pair $\bar{B} = \langle B, F \rangle$, hereafter, referred to as a *constrained BGP* (*CBGP*), where B is a finite set of triple patterns (i.e., a BGP) and F is a finite set of SPARQL filter expressions. Hence, by using the algebraic syntax for SPARQL [3], a CBGP $\bar{B} = \langle B, F \rangle$ with $F = \{f_1, \dots, f_n\}$ is equivalent to a SPARQL graph pattern P of the form $\big((\dots(B \text{ FILTER } f_1)\dots) \text{ FILTER } f_n\big)$ (if $F = \emptyset$, then P is the BGP B). Consequently, the *evaluation* of $\bar{B}$ over an RDF graph G, denoted by $[\![\bar{B}]\!]_G$, is the bag (multiset) of solution mappings $[\![P]\!]_G$ as defined by the standard SPARQL query semantics [3], [14].

While a discussion of a more expressive fragment of SPARQL is certainly possible, for our purposes, this is not necessary: the objective of our benchmark is stress testing,



(a) Linear (b) Star (c) Snowflake

Fig. 1. Possible query structures

with an emphasis on revealing issues with the physical design of RDF data management systems. To this end, we try to generate test queries that are as diverse as possible within this basic fragment of SPARQL and deliberately avoid testing of complex functionality such as filters, unions, aggregation and optional graph patterns.

2.1 Structural Features

Every BGP (as used by a CBGP) combines a set of triple patterns into one of numerous query structures such as those in Fig. 1a–1c. As a basis for comparing the structural diversity of different sets of CBGPs we introduce four features.

Triple Pattern Count: This feature refers to the number of triple patterns in (the BGP of) a CBGP. Triple pattern count allows one to broadly distinguish between simple and structurally complex queries. Ideally, we would like an RDF system to execute simple queries extremely fast while scaling well with increasing number of triple patterns. In fact, DBpedia query logs [17] reveal that while in general most queries contain only a few triple patterns, users may issue (albeit infrequently) queries having more than 50 triple patterns.

Join Vertex Count: This feature represents the number of RDF terms (i.e., URIs, literals, and blank nodes) and variables that are the subject or object of multiple triple patterns in a CBGP. Hereafter, we refer to these terms and variables as *join vertices* of the CBGP. Formally, if $\mathcal{T}$ and $\mathcal{V}$ denote the set of all RDF terms and the set of all variables, respectively, an element $x \in (\mathcal{T} \cup \mathcal{V})$ is a *join vertex* of CBGP $\bar{B} = \langle B, F \rangle$ if there exist two distinct triple patterns $tp = \langle s, p, o \rangle$ and $tp' = \langle s', p', o' \rangle$ such that (i) $tp \in B$ and $tp' \in B$, (ii) $x \in \{s, o\}$, and (iii) $x \in \{s', o'\}$.

Join Vertex Degree: For each join vertex x of a CBGP $\bar{B} = \langle B, F \rangle$, the *degree* of x is the number of triple patterns in B whose subject or object is x. Hereafter, for any such triple pattern $\langle s, p, o \rangle \in B$ with $x \in \{s, o\}$ we say that the triple pattern is *incident* on x.

Join vertex count and join vertex degree offer a finer distinction of structural complexity than the triple pattern count. For example, the two queries in Fig. 1a and Fig. 1b have the same number of triple patterns but they differ in their join vertex count and join vertex degrees. That is, Fig. 1a is a long linear-shaped query with multiple (4) low-degree (2) join vertices, whereas Fig. 1b is a star-shaped query with a single high-degree (5) join vertex. A system may show completely different performance for these two queries and a benchmark should capture such blind spots if any.

Join Vertex Types: The data representation and indexing schemes employed by RDF systems can result in completely different behaviour on different types of joins [2], and a benchmark should include a sufficiently large sample of queries for each join type. Consequently, we distinguish the following three (mutually non-exclusive) types of join vertices: A join vertex x of a CBGP $\bar{B} = \langle B, F \rangle$ is of *type SS*$^+$ if $x = s$ for every triple pattern $\langle s, p, o \rangle \in B$ that is incident on x; similarly, x is of *type OO*$^+$ if $x = o$ for every $\langle s, p, o \rangle \in B$ that is incident on x; and x is of *type SO*$^+$ if $x = s$ and $x = o'$ for two triple patterns $\langle s, p, o \rangle \in B$ and $\langle s', p', o' \rangle \in B$ (both of which are incident on x, respectively). For example, the join vertices $?a$, $?x$ and $?y$ in Fig. 1c have types SS$^+$, SO$^+$and OO$^+$, respectively.

2.2 Data-Driven Features

The structural query features (discussed above) are often not sufficient. More specifically, a system's choice of a query (execution) plan depends on the characteristics of the data as much as the query itself. For example, systems rely heavily on selectivity and cardinality estimations for query plan optimization [23]. Consider the following example: A system chooses to break down a BGP $B = \{tp_A, tp_B, tp_C\}$ into its triple patterns and to execute them in a specific order, namely, tp_A, tp_B and then tp_C. The system picks this particular query plan because the subset of triples that match tp_A is smaller. Furthermore, it estimates the intermediate result cardinalities to be sufficiently low and decides to use in-memory data structures and algorithms. To enumerate different plan choices, we consider the following test cases:

- queries have a diverse mix of result cardinalities;
- a single or few triple patterns are very selective, while the remaining ones are not;
- all of the triple patterns in a query are almost equally selective (hence, there is a higher probability that the optimizer picks a sub-optimal query plan due to estimation errors); etc.

Next, we define result cardinality and notions of selectivity, and explain how we use them in our evaluations to distinguish among such different test cases.

Result Cardinality: This feature represents the number of solutions in the result of evaluating a CBGP $\bar{B} = \langle B, F \rangle$ over an RDF graph G. Recall that this result, denoted by $[\![\bar{B}]\!]_G$, is a bag (multiset) of solution mappings (cf. Sec. 2.1). Consequently, if Ω denotes the set underlying the bag $[\![\bar{B}]\!]_G$ and $\mathrm{card}_{[\![\bar{B}]\!]_G}$ denotes the function that maps each solution mapping $\mu \in \Omega$ to its cardinality in the bag [3], we define the result cardinality of $\bar{B}$ over G by

$$\textsc{card}(\bar{B}, G) = \sum_{\mu \in \Omega} \mathrm{card}_{[\![\bar{B}]\!]_G}(\mu). \tag{1}$$

Filtered Triple Pattern Selectivity (f-TP Selectivity): Given some CBGP $\bar{B} = \langle B, F \rangle$ and a BGP B^* such that $B^* \subseteq B$, we write $\lambda^F(B^*)$ to denote the CBGP $\bar{B}' = \langle B', F' \rangle$ with $B' = B^*$ and $F' = \{f \in F \,|\, \mathrm{vars}(f) \subseteq \mathrm{vars}(B^*)\}$, where $\mathrm{vars}(\cdot)$ denotes the variables in a filter expression or a BGP. Then, for any triple pattern $tp \in B$ in a CBGP $\bar{B} = \langle B, F \rangle$, the *f-TP selectivity* of tp over an RDF graph G, denoted by $\textsc{sel}^F_G(tp)$, is the ratio of distinct solution mappings in $[\![\lambda^F(\{tp\})]\!]_G$ to the number of triples in G. Formally, if Ω denotes the set underlying the (bag) query result $[\![\lambda^F(\{tp\})]\!]_G$, then

$$\textsc{sel}^F_G(tp) = {}^{|\Omega|}/_{|G|}. \tag{2}$$

In our evaluations, we use three related measures. We use the result cardinality of a CBGP as it is defined, and we compute the *mean* and *standard deviation* of the f-TP selectivities of the triple patterns in the CBGP. The latter is especially important in distinguishing queries whose triple patterns are almost equally selective from queries with varying f-TP selectivities.

While result cardinality and f-TP selectivity are useful features, they are not entirely sufficient. More specifically, once a system picks a particular query plan and starts executing it, it is often the case that there are intermediate solution mappings which do not make it to the final result. What this means is that *all* triple patterns of a CBGP contribute to its overall "selectiveness", or stated differently, in every join step, some intermediate solution mappings are being pruned. Contrast this to another possible case in which the overall "selectiveness" of a CBGP can be attributed to a single triple pattern in that CBGP. In that case, a system could use runtime optimization techniques such as sideways-information passing [18] to *early-prune* most of the intermediate results, which may not be possible in the original example (for a more detailed discussion refer to [2]). From a testing point of view, it is important to include both cases. In fact, in Section 5.5, we shall revisit this example and experimentally show that systems behave differently on these two cases. To capture these constraints, we study two more features, namely BGP-restricted and join-restricted f-TP selectivity. The former is concerned with how much a filtered triple pattern contributes to the overall "selectiveness" of the query, whereas the latter is concerned with how much a filtered triple pattern contributes to the overall "selectiveness" of the join(s) that it participates in. Just as we do with f-TP selectivity, for our evaluations, we compute the mean and standard deviation of these two features.

BGP-Restricted f-TP Selectivity: For any triple pattern $tp \in B$ in a CBGP $\bar{B} = \langle B, F \rangle$, the *$\bar{B}$-restricted f-TP selectivity* of tp over an RDF graph G, which is denoted by $\textsc{sel}_G^F(tp \,|\, \bar{B})$, is the fraction of distinct solution mappings in $[\![\lambda^F(\{tp\})]\!]_G$ that are *compatible* (as per standard SPARQL semantics [3]) with a solution mapping in the query result $[\![\bar{B}]\!]_G$. Formally, if Ω and Ω' denote the sets underlying the (bag) query results $[\![\lambda^F(\{tp\})]\!]_G$ and $[\![\bar{B}]\!]_G$, respectively, then

$$\textsc{sel}_G^F(tp \,|\, \bar{B}) = \frac{\big|\{\, \mu \in \Omega \,|\, \exists \mu' \in \Omega' : \mu \text{ and } \mu' \text{ are compatible} \,\}\big|}{|\Omega|}. \tag{3}$$

Join-Restricted f-TP Selectivity: Given a CBGP $\bar{B} = \langle B, F \rangle$, a join vertex x of $\bar{B}$, and a triple pattern $tp \in B$ that is incident on x, the *x-restricted f-TP selectivity* of tp over an RDF graph G, denoted by $\textsc{sel}_G^F(tp \,|\, x)$, is the fraction of distinct solution mappings in $[\![\lambda^F(\{tp\})]\!]_G$ that are compatible with a solution mapping in the (join) query result $[\![\lambda^F(B^x)]\!]_G$ with $B^x \subseteq B$ being the subset of all the triple patterns in B that are incident on x (i.e, $B^x = \{tp \in B \,|\, tp \text{ is incident on } x\}$). Hence, if Ω and Ω' denote the sets underlying $[\![\lambda^F(\{tp\})]\!]_G$ and $[\![\lambda^F(B^x)]\!]_G$, respectively, then

$$\textsc{sel}_G^F(tp \,|\, x) = \frac{\big|\{\mu \in \Omega \,|\, \exists \mu' \in \Omega' : \mu \text{ and } \mu' \text{ are compatible} \,\}\big|}{|\Omega|}. \tag{4}$$

3 Evaluation of Existing SPARQL Benchmarks

Even though existing SPARQL benchmarks [7], [12], [17], [20] offer valuable testing capabilities, we demonstrate in this section that they are not suitable for stress testing RDF systems. We consider the following 4 benchmarks:

– The *Lehigh University Benchmark* (LUBM) [12] was originally designed for testing the inferencing capabilities of Semantic Web repositories.
– The *Berlin SPARQL Benchmark* (BSBM) [7] contains multiple use cases such as (i) explore, (ii) update, and (iii) business intelligence use cases. Furthermore, it goes into testing how well RDF systems support different (and important) SPARQL features, namely, aggregation, union, and optional graph patterns.
– *SP^2Bench* [20] tests various SPARQL features such as union and optional graph patterns.
– The *DBpedia SPARQL Benchmark* [17] (DBSB) uses queries that have been generated by mining actual query logs over the DBpedia dataset [5]. Thus, it contains a more “diverse set of queries” [17].

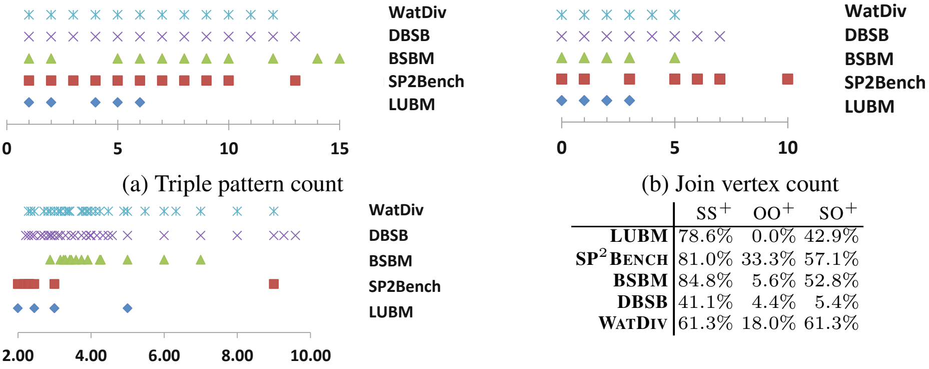
We assess the diversity of existing benchmarks using the structural and data-driven features presented in Section 2. In our evaluations of benchmarks, we only consider SELECT queries. For BSBM, we focused on the explore use case and generated 100 queries per query template. We observed this to be a sufficiently large sample to understand the general properties of BSBM. For DBSB, we analyzed a sample of 12500 queries that were drawn uniformly at random from the subset of SELECT queries in the query logs (the other two benchmarks have a fixed number of queries). For WatDiv, we generate the same number of queries (12500). Recall that the query features in Section 2 are defined over CBGPs. For this reason, when analyzing existing benchmarks (with respect to these features), we first translate each complex non-CBGP query into a CBGP by replacing OPT and UNION operators with AND. Hereafter, we refer to these CBGPs (including those for which translation was not necessary) as the queries of the benchmark. To compute the statistics reported in this section, for each benchmark, we generated a benchmark-specific dataset of 1 million triples, and executed all of the queries in the benchmark.

3.1 Evaluation Using Structural Features

Consider Fig. 2a, which compares queries in each benchmark with respect to their triple pattern count (x-axis).[1] Benchmarks are stacked along the y-axis. For each benchmark, the presence of a point indicates that the benchmark contains at least one query with the corresponding number of triple patterns indicated by the x-axis value. Fig. 2a–2c and Fig. 3a– 3f should be read similarly. The actual *distribution* of queries with respect to these features are available in the online version of this paper.[2]

[1] For the time being ignore WatDiv in these figures. The results about WatDiv are not important for this section. We discuss WatDiv in Section 4.

[2] `https://cs.uwaterloo.ca/~galuc/watdiv/paper/`



(a) Triple pattern count

(b) Join vertex count

	SS$^+$	OO$^+$	SO$^+$
LUBM	78.6%	0.0%	42.9%
SP^{2}Bench	81.0%	33.3%	57.1%
BSBM	84.8%	5.6%	52.8%
DBSB	41.1%	4.4%	5.4%
WatDiv	61.3%	18.0%	61.3%

(c) Join vertex degree (mean)

(d) Distribution w.r.t. join vertex types

Fig. 2. Analysis w.r.t. structural features: in Fig. 2a–2c, each point indicates the presence of a query with the corresponding x-axis value for a given feature

While most benchmarks contain large queries with more than 10 triple patterns (Fig. 2a)[3], LUBM contains only small queries—not exceeding 6 triple patterns in cardinality. Furthermore, LUBM's join vertex count is also lower than the other benchmarks (Fig. 2b). This is reasonable as LUBM is intended for semantic inferencing. In fact, the true complexity of an LUBM query lies in its semantics, not in its structure. For this reason, the suitability of LUBM for performance evaluation is limited if the system under test does not support inferencing.

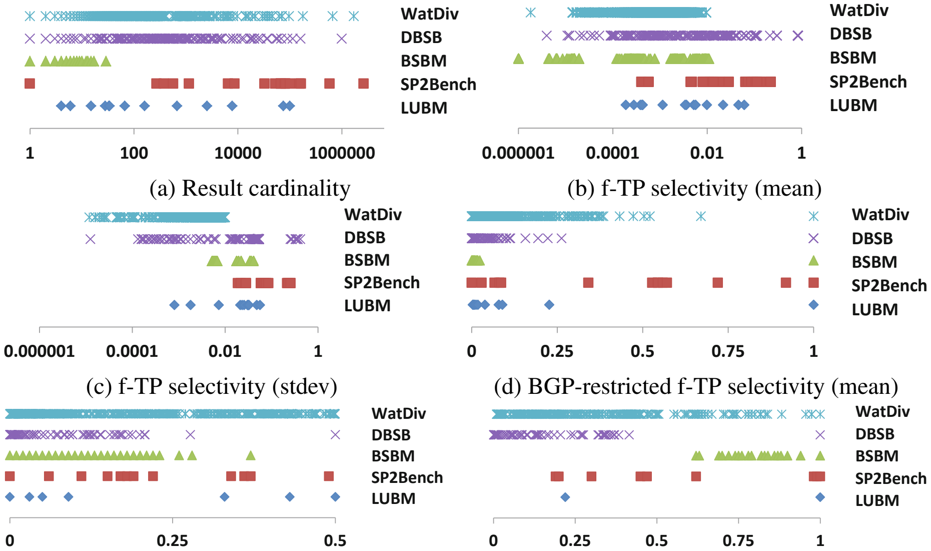
By considering mean join vertex degrees (Fig. 2c), we observe that DBSB is more diverse than any of the synthetic benchmarks (i.e., LUBM, BSBM, SP^{2}Bench). LUBM contains fairly simple queries (cf., Fig. 2a), which explains why the mean join vertex degree is also low for most of these queries. SP^{2}Bench contains (i) linear queries that are long, or (ii) star queries that are large and centered around a single join vertex, but not much in between; hence, the join-vertex degree values are concentrated at the two ends of the x-axis in Fig. 2c. BSBM contains queries that are a little bit more diverse in their join vertex degrees, but it does not test the two extremes as SP^{2}Bench does.

In Fig. 2d, we compare and contrast benchmarks with respect to the types of join vertices present in each of the queries. This comparison reveals three problems: LUBM does not contain any query with an OO$^+$join; BSBM contains some, but their percentage is significantly low. In DBSB, queries with both OO$^+$and SO$^+$joins have a low percentage. Consequently, these three benchmarks may be biased towards particular physical designs that are more effective for SS$^+$(or SO$^+$) joins, which limits the suitability of these benchmarks for stress tests.

3.2 Evaluation Using Data-Driven Features

Regarding result cardinality, the following observations can be made. BSBM contains only low-cardinality queries, SP^{2}Bench contains almost only high-cardinality queries,

[3] Some DBSB queries have as many as 50 triple patterns, but for clarity of presentation we are not displaying them.



(a) Result cardinality

(b) f-TP selectivity (mean)

(c) f-TP selectivity (stdev)

(d) BGP-restricted f-TP selectivity (mean)

(e) BGP-restricted f-TP selectivity (stdev)

(f) Join-restricted f-TP selectivity (mean)

Fig. 3. Analysis w.r.t. data-driven features at 1 million triples: each point indicates the presence of a query with the corresponding x-axis value for a given feature

and LUBM contains only medium-cardinality queries (cf., Fig. 3a), which reveals another limitation of what each of these three benchmarks can test individually.

Fig. 3b–3c show another issue with existing benchmarks. Although benchmarks are fairly diverse with respect to f-TP selectivity (i.e., especially DBSB and BSBM), the standard deviation of f-TP selectivities of filtered triple patterns (within any single query) is generally high. As explained in Section 2.2, this implies that these benchmarks are missing the test case in which the triple patterns are more or less equally selective.

As depicted in Fig. 3d, among the four benchmarks, only SP^{2}Bench has a diverse selection of queries regarding mean BGP-restricted f-TP. LUBM, BSBM and DBSB have queries in which either the mean value is 1.0, indicating that each triple pattern in separation does not contribute to the selectiveness of the query, or the mean is extremely low, indicating the opposite. For BSBM, the contrast is even extreme. Fig. 3e highlights an even further problem with DBSB and BSBM. For these two benchmarks, the variation in BGP-restricted f-TP lies mostly in the lower end of the spectrum, which indicates that these benchmarks cannot be used to test with queries in which triple patterns contribute unevenly to the pruning of intermediate results (cf., Section 2.2).

Finally, consider Fig. 3f, which compares benchmark queries using join-restricted f-TP (mean). One can observe two important limitations. First, both LUBM and SP^{2}Bench queries sparsely cover the spectrum of possible values. Second, although BSBM and DBSB are much more diverse, they cover completely different ends of the spectrum. A system can generate completely different query plans for these two scenarios, and therefore, stress testing should use workloads that include both scenarios.

3.3 Summary of Findings

In summary, the best known benchmarks (including DBSB, which is based on actual query logs), individually, are not sufficiently diverse to test the strengths and weaknesses of different physical design choices employed by RDF systems. Aggregating results from multiple benchmarks is not a good solution to the diversity problem either. First, the underlying datasets have completely different characteristics; therefore, we may get queries with completely disjoint distributions across the structural and data-driven features. For example, even though it may appear, based on Fig. 3f, that DBSB and BSBM complement each other (i.e., they cover the opposite ends of the set of possible x-axis values), Fig. 3a suggests that it is not quite so. The problem is that these two benchmarks do not complement each other on *all* possible features. Hence, in an aggregated (hypothetical) benchmark, we would still be missing queries with *high* cardinality and *high* join-restricted f-TP selectivity values. Second, scalability is an issue. It is not clear (i) how we can generate more queries given that some of the above-mentioned benchmarks have a fixed number of queries, or (ii) how results from multiple benchmarks should be combined given that each benchmark has its own scalability restrictions. Our benchmark is designed to address these issues.

4 Waterloo SPARQL Diversity Test Suite (WatDiv)

WatDiv consists of multiple tools[4] that enable diversified stress testing of RDF data management systems:

- The *data generator* generates scalable datasets at user-specified scale factors—a common feature of benchmarks. A more interesting feature is that data are generated according to the WatDiv schema[5] with customizable value distributions.
- The *query template generator* traverses the WatDiv schema and generates a diverse set of query templates (which is the first step in generating a workload for the stress tests). Users can specify the number of query templates to be generated as well as certain restrictions on the query templates such as the maximum number of triple patterns or whether predicates in triple patterns should be bound.
- Given a set of query templates, the *query generator* instantiates these templates with actual RDF terms from the dataset (which is the second and last step in generating a workload for the stress tests). The number of queries to be instantiated per query template can be specified by users.
- Given a WatDiv dataset and test workload, for each query in the workload, the *feature extractor* computes the structural and data-driven features discussed in Section 2. For this to work, the tool needs to point to a third party RDF data management system that is already installed on the system.

Dataset Description: What distinguishes WatDiv datasets from existing RDF benchmarks is the diversity of the structuredness: some entities in WatDiv are well-structured,

[4] `http://db.uwaterloo.ca/watdiv/`

[5] `http://db.uwaterloo.ca/watdiv/watdiv-data-model.txt`

meaning that they contain few optional attributes, while some others are less well-structured [10]. We discuss in Section 4 that this enables the generation of test queries that are far more diverse in their data-driven features.

Three properties contribute to the diversity of WatDiv. First, instances of the same entity type (i.e., class) do not necessarily have the same properties. Consider the different types of entities used in WatDiv.[6] *Product* instances may be associated with different *Product Categories* (e.g., Book, Movie, Classical Music Concert, etc.), but depending on the category a product belongs to, it will have a different set of properties. For example, products that belong to the category "Classical Music Concert" have the properties *opus*, *movement*, *composer* and *performer* (in addition to the properties that are common to every product), whereas products that belong to the category "Book" have the properties *isbn*, *bookEdition* and *numberOfPages*.

Second, even within a single product category, not all instances share the same set of properties. For example, while *isbn* is a mandatory property for books, *bookEdition* ($Pr = 0.5$) and *numberOfPages* ($Pr = 0.25$) are optional properties, where Pr indicates the probability that an instance will be generated with that property. Users are able to modify the WatDiv schema, hence these probabilities.

Third, a group of attributes can be correlated, which means that either all or none of the correlated attributes in that group will be present in any instance of the entity type. For example, *opus* and *movement* are two correlated properties for "Classical Music Concert" products (cf. <pgroup> construct in the WatDiv dataset schema).

Test Queries: The benchmark queries are generated in two steps. First, a set of query templates are created by performing a random walk over the graph representation of the schema of the dataset (i.e., query template generator). In this regard, we use the following (internal) representation: every entity type in the schema corresponds to a graph vertex, relationships among entity types (i.e., which correspond to RDF predicates in the instantiated dataset) are represented using graph edges, and each vertex is annotated with the set of properties of that entity type. This produces a set of BGPs with a maximum n triple patterns, where n was set to 15 in our experiments. Note that we generate BGPs with triple patterns that have unbound subjects and objects, whereas their predicates are bound. Then, k uniformly randomly selected subjects/objects are replaced with WatDiv-specific placeholders (i.e., placeholders are enclosed within percentage $[\%]$ signs in the benchmark). In the second step, placeholders in each query template are instantiated with RDF terms from the WatDiv dataset (i.e., query generator). To this end, the WatDiv tools maintain, for each placeholder, a set of values that are applicable to that placeholder, and during the instantiation step, a value is drawn uniformly at random. For the study in this paper, we generated 12500 test queries from a total of 125 query templates (i.e., the same number of queries we sampled in DBSB). These queries are available online.[7]

Discussion: In Fig. 2a–2d and Fig. 3a–3f, we characterize the aforementioned 12500 WatDiv test queries. With respect to most of the structural query features, WatDiv has comparable characteristics to DBSB and it is far more diverse than LUBM, SP^{2}Bench

[6] http://db.uwaterloo.ca/watdiv/#dataset

[7] http://db.uwaterloo.ca/watdiv/stress-workloads.tar.gz

and BSBM (cf., Fig. 2a–2c). For example, the mean join vertex degree values are densely distributed between 2.0 and 10.0, indicating a rich mix of queries. Furthermore, with respect to join vertex types, WatDiv has a much more balanced distribution than DBSB: a significant 18.0% of queries in the WatDiv workload have OO^+-type join vertices, compared to only 4.4% in DBSB, and 61.3% versus 5.4% for queries with SO^+joins.

With respect to most of the data-driven features, WatDiv is far more diverse, often filling in the gaps that are not supported by existing benchmarks (cf., Fig. 3d, 3e and 3f). For example, while DBSB and BSBM cover only the opposite ends of the spectrum of mean join-restricted f-TP selectivity values, WatDiv covers the full spectrum (cf., Fig. 3f). With respect to mean f-TP selectivity (hence, also standard deviation), WatDiv covers a lower range of values than DBSB and other benchmarks (cf., Fig. 3b–3c). This is because in DBSB there are unselective queries that return the whole dataset, that is, the subjects, predicates and objects in a triple pattern are all unbound. In contrast, recall from Section 4 that for our evaluation we generated queries in which the predicates in a triple pattern are bound (enabling this feature in WatDiv is a configuration option). Therefore, for this feature WatDiv complements the other benchmarks. Overall, due to the comprehensiveness of WatDiv, it has enabled us to reveal performance issues about existing RDF systems that were missed in studies that used the other four benchmarks.

5 Evaluation of RDF Systems

We used WatDiv to evaluate a number of existing RDF data management systems. In this section, we report our experimental results and discuss various issues with existing systems.

5.1 Systems Under Test

RDF systems can be classified broadly into two categories in terms of their data representations: (i) tabular and (ii) graph-based. For tabular implementations, one option is to represent data in a single large table. While earlier triplestores followed this approach [8,9], it has been demonstrated that maintaining redundant copies with different sort orders and indexes can be much more effective [19]. Consequently, in our evaluations we include the popular prototype RDF-3x [19] (v0.3.7) that follows the latter approach. It has also been argued that grouping data can significantly improve performance for some workloads [22]. Hence, a second option is to group data by RDF predicates, where data are explicitly partitioned into multiple tables (one table per predicate) and the tables are stored in a column-store [1]. We test the effectiveness of this approach on MonetDB [15] (v1.7), which is a state-of-the-art column-store. A third option is to natively represent RDF graph structure, for which we use the prototype system gStore [24] (v0.2). We also test three industrial systems, namely, Virtuoso Open Source (VOS) [11] (v6.1.8 and v7.1.0) and 4Store [13] (v1.1.5). Both VOS and 4Store group and index data primarily based on RDF predicates. Furthermore, VOS 6.1 is a row-store and VOS 7.1 is a column-store.

5.2 Experimental Setup

In our experiments, we use a commodity machine with AMD Phenom II ×4 955 3.20 GHz processor, 16 GB of main memory and a Seagate $3.AA$ hard disk drive with 100 GB of free space at the time of experimentation. The operating system on the machine is Ubuntu 12.04 LTS.

In this paper, our objective is to understand how well state-of-the-art RDF systems perform on a diverse SPARQL workload; however, we do *not* intend to test the scalability of these systems given more computational nodes and/or CPUs. Therefore, we restrict each system to use single-threading. WatDiv is equally suitable for scalability experiments, but these results are not included in this paper.

In our stress-tests we use two versions of the WatDiv dataset, one generated at scale-factor 100 and the other at 1000, which correspond to approximately 10 million and 100 million triples, respectively. Recall from Section 4 that we use 12500 queries generated from 125 query templates.

We evaluate the systems using a warm cache. Therefore, we generate two workloads: a warmup workload and a test workload, both containing the same 12500 queries. On each system, first the warmup workload and then the test workload is executed. To achieve higher confidence, we repeat the experiments 5 times. Furthermore, to reduce the effects of query interactions within the test workload, every time, the sequence of queries in the test workload is randomized (the warmup experimental run is just another randomized test run). This way, for each query in a test sequence, we measure and record its execution time as well as various structural and data-driven features about that query. For practical reasons, whenever query execution exceeds 60 seconds, we automatically timeout, proceed with the next query in the sequence, and ignore that query in the consecutive runs for that system.

5.3 Results

The experimental results are summarized in Fig. 4a–4d. The complete results with error margins are available in the aforementioned online version of the paper. Fig. 4a displays, for each system, the total execution time (averaged over the five randomized sequences) of the test workload. Fig. 4b depicts, for each system, the percentage of queries in the workload that particular system is the fastest (timeouts are ignored) or up to 10 times slower than the fastest system, and so on. Fig. 4c–4d display for each query in the workload (x-axis), the query execution time (in milliseconds) of the fastest as well as the slowest system for that query (which may be different systems for different queries). For presentation purposes, queries are sorted according to their maximum execution times. Note that for some queries, the maximum execution time is capped at 60 seconds, which marks the timeout threshold.

gStore ran into errors during the execution of some of the queries: we do not consider these cases in our discussions. Consequently, the percentages for gStore in Fig. 4b do not add up to 100%. Furthermore, gStore timed out on the queries on the larger dataset, hence, we excluded it from Fig. 4b.

5.4 Observations

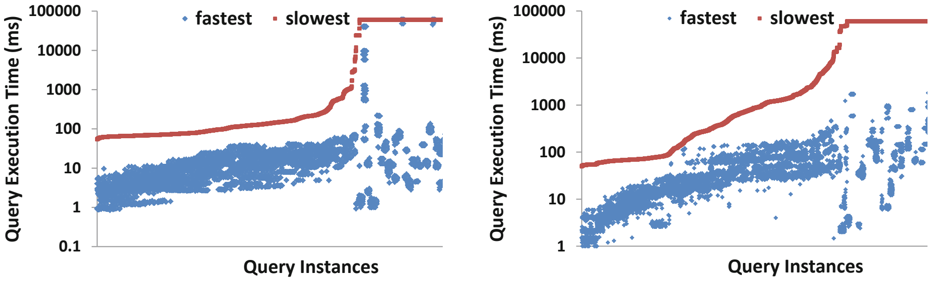
Regarding Fig. 4a, we make two observations. First, VOS (and to some extent RDF-3x) perform much better than the other three systems on the larger dataset. Second,

	RDF-3x	VOS [6.1]	VOS [7.1]	MonetDB	4Store	gStore
10M triples	58,312	**41,612**	51,268	48,329	94,289	n/a
100M triples	97,409	75,224	**74,997**	139,015	260,045	n/a

(a) Total workload execution time (in seconds) for the systems under test

		10M triples						100M triples					
		RDF-3x	VOS [6.1]	VOS [7.1]	MonetDB	4Store	gStore	RDF-3x	VOS [6.1]	VOS [7.1]	MonetDB	4Store	gStore
fastest		11.4%	6.5%	18.7%	31.7%	0.8%	30.9%	20.9%	0.0%	22.6%	56.5%	0.0%	n/a
1–10×	slower	77.2%	67.5%	63.4%	65.0%	49.6%	35.8%	60.9%	59.1%	54.8%	31.3%	53.0%	n/a
10–100×		6.5%	23.6%	13.0%	1.6%	41.5%	2.4%	13.9%	40.0%	20.0%	2.6%	21.7%	n/a
100–1K×		3.3%	1.6%	4.1%	0.0%	0.8%	0.0%	3.5%	0.9%	1.7%	6.1%	15.7%	n/a
1K–10K×		1.6%	0.8%	0.8%	1.6%	7.3%	0.0%	0.0%	0.0%	0.9%	3.5%	7.0%	n/a
10K–100K×		0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.9%	3.5%	7.0%	n/a

(b) Performance breakdown (10M and 100M triples)



(c) Gap between the recorded execution times of the fastest and slowest systems at 10M triples (per query instance)

(d) Gap between the recorded execution times of the fastest and slowest systems at 100M triples (per query instance)

Fig. 4. WatDiv Results: Robustness of Existing Systems

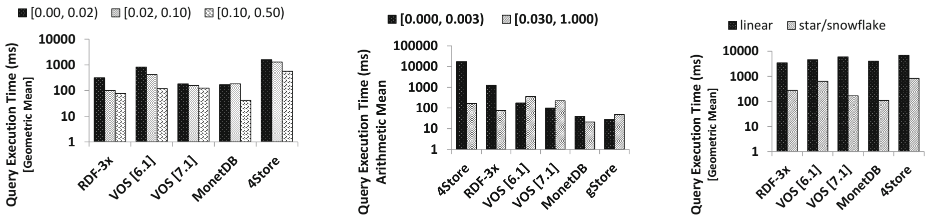
although VOS has the lowest total execution time for the whole workload (Fig. 4a), it is the fastest system in not more than 23 percent of the queries (Fig. 4b). This highlights an interesting trade-off between robustness across a diverse set of queries versus speed within a specific type of workload.

Note that no single system is the absolute winner in all of the queries (cf., Fig. 4b. Furthermore, note that each system performs poorly (i.e., a few orders of magnitude worse than the fastest system) in a significant percentage of queries in the workload.

The results in Fig. 4c–4d highlight two more issues. First, for most queries, there can be 2 orders of magnitude difference between the fastest and slowest system, and in the worst case, this gap can be as large as 5 orders of magnitude (note that this gap exists even when query execution times are grouped by query template). Second, the worst case gap widens from the smaller to the larger dataset.

In summary:

– No single system is a sole winner across all queries;
– No single system is the sole loser across all queries, either;



(a) BGP-restricted f-TP selectivity (stdev) values at 100M triples

(b) BGP-restricted f-TP selectivity (mean) at 10M triples

(c) Query structure at 100M triples

Fig. 5. Detailed evaluation: results are analyzed across various combinations of features

- For some queries, there can be as much as 5 orders of magnitude difference in the performance (i.e., query execution time) between the best and the worst system for that query; and
- The winner in one query may timeout in another.

5.5 Detailed Analysis

In this section, we provide a more detailed evaluation by drilling down into particular query features (and combinations thereof). Hypothetically speaking, it is possible to perform such analyses using any possible combination of features (including any additional feature not covered by our study). However, due to space limitations, we focus on a few special cases where the results stand out, and while doing so, we demonstrate how WatDiv can be used for stress testing.

As our first exercise, we quantify an observation that we made in Section 2.2. That is, we want to test whether systems behave differently for queries in which all (or most) triple patterns contribute almost equally to the overall "selectiveness" of the query (Case-A) versus the case in which the overall "selectiveness" of the query can be attributed to a single (or few) triple patterns (Case-B). To distinguish between these two cases, we rely on the standard deviation of BGP-restricted f-TP selectivity, where a low (resp., high) standard deviation implies Case-A (resp., Case-B). For this exercise, we take into account only the queries with result cardinality ≤ 2000 (i.e., selective queries). We divide the spectrum of standard deviation values into three intervals such that we have an equal number of queries in each interval (approximately 3300 queries per interval). Fig. 5a depicts, for each system, the geometric mean of the query execution times of all queries in each of the three intervals. We note that, for all four systems, the (mean) query execution times decrease as the standard deviation of BGP-restricted f-TP selectivity increases. These results indicate that, while systems have integrated techniques to early-prune intermediate results [18], these techniques do not seem to be effective for Case-A.

Next, we demonstrate a case in which different systems show varied behavior on a particular type of workload. In this exercise, we consider only the queries with a single join vertex and result cardinality ≤ 2000. Then, based on the mean of BGP-restricted f-TP selectivity, we devised two types of workloads: one in which queries have very low mean BGP-restricted f-TP selectivity, and the other in which the mean is high (each interval contains approximately 2200 queries). The former workload captures those queries in which due to data distributions, the query itself becomes much more selective than the individual triple patterns participating in the query. Fig. 5b illustrates an interesting trend: while the five systems behave similarly to some extent for the latter workload, they have completely differing performance in the former one. An investigation that may reveal a reason for this observation is beyond the scope of this paper.

Last, we test whether systems are biased towards a particular query structure (i.e., linear vs. star/snowflake). To this end, we select two sets of queries: (i) those queries with mean join vertex degree ≤ 3.0 and join vertex count ≥ 3 (representing linear queries), and (ii) those with mean join vertex degree ≥ 5.0 and join vertex count ≤ 2 (representing star or snowflake queries). The results in Fig. 5c demonstrate that all of the four systems are indeed biased against linear queries, highlighting a room for improvement.

6 Conclusions

In this paper, we discuss WatDiv. First, we introduce a set of query features that can be used for assessing the diversity of the data and workloads in a SPARQL benchmark. We explain why these features are important and how they relate to special test cases that need to be included in a stress testing tool. Then, we discuss our experimental evaluation of existing SPARQL benchmarks with a specific emphasis on identifying test cases that are not handled by these benchmarks, which led us to the development of WatDiv. Our experimental evaluation of existing RDF data management systems with WatDiv demonstrate that these systems are not sufficiently robust across a diverse set of queries. Then, we use WatDiv to drill down into specific combinations of query features to reveal problems that only this type of stress testing could reveal. Specifically, we illustrate cases where all of the evaluated systems show bias against a particular type of workload, or where a particular system has some advantage over the others for a specific type of workload. We believe that evaluations that involve stress testing as demonstrated in this paper are crucial to build more robust RDF data management systems. For future work, we consider extending WatDiv to support provenance and temporal data.

References

1. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: SW-Store: a vertically partitioned DBMS for semantic web data management. VLDB J. 18, 385–406 (2009)
2. Aluç, G., Özsu, M.T., Daudjee, K.: Workload matters: Why RDF databases need a new design. Proc. VLDB 7(10), 837–840 (2014)
3. Arenas, M., Gutierrez, C., Pérez, J.: On the semantics of SPARQL. In: Semantic Web Inf. Man., pp. 281–307 (2009)
4. Arias, M., Fernández, J.D., Martínez-Prieto, M.A., de la Fuente, P.: An empirical study of real-world SPARQL queries. CoRR abs/1103.5043 (2011)

5. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: DBpedia: A nucleus for a web of open data. In: Aberer, K., et al. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007)
6. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Semantic Web Inf. Syst. 5(3), 1–22 (2009)
7. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. Int. J. Semantic Web Inf. Syst. 5(2), 1–24 (2009)
8. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A generic architecture for storing and querying RDF and RDF Schema. In: Proc. 1st Int. Semantic Web Conference, pp. 54–68 (2002)
9. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the semantic web recommendations. In: Proc. 13th Int. World Wide Web Conf. - Alternate Track Papers & Posters, pp. 74–83 (2004)
10. Duan, S., Kementsietsidis, A., Srinivas, K., Udrea, O.: Apples and oranges: a comparison of RDF benchmarks and real RDF datasets. In: SIGMOD Conference, pp. 145–156 (2011)
11. Erling, O.: Virtuoso, a hybrid RDBMS/graph column store. IEEE Data Eng. Bull. 35(1), 3–8 (2012)
12. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. Web Semantics 3(2-3), 158–182 (2005)
13. Harris, S., Lamb, N., Shadbolt, N.: 4store: The design and implementation of a clustered RDF store. In: Proc. 5th Int. Workshop on Scalable Semantic Web Knowledge Base Systems, pp. 81–96 (2009)
14. Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 query language. W3C Recommendation (March 2013)
15. Idreos, S., Groffen, F., Nes, N., Manegold, S., Mullender, K.S., Kersten, M.L.: MonetDB: Two decades of research in column-oriented database architectures. IEEE Data Eng. Bull. 35(1), 40–45 (2012)
16. Kirchberg, M., Ko, R.K.L., Lee, B.S.: From linked data to relevant data – time is the essence. CoRR abs/1103.5046 (2011)
17. Morsey, M., Lehmann, J., Auer, S., Ngonga Ngomo, A.-C.: DBpedia SPARQL benchmark – performance assessment with real queries on real data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 454–469. Springer, Heidelberg (2011)
18. Neumann, T., Weikum, G.: Scalable join processing on very large RDF graphs. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 627–640 (2009)
19. Neumann, T., Weikum, G.: The RDF-3X engine for scalable management of RDF data. VLDB J. 19(1), 91–113 (2010)
20. Schmidt, M., Hornung, T., Meier, M., Pinkel, C., Lausen, G.: Sp^{2}bench: A sparql performance benchmark. In: Semantic Web Inf, pp. 371–393 (2009)
21. Schreiber, G., Raimond, Y.: RDF 1.1 primer. W3C Note (February 2014)
22. Sidirourgos, L., Goncalves, R., Kersten, M., Nes, N., Manegold, S.: Column-store support for RDF data management: not all swans are white. Proc. VLDB 1(2), 1553–1563 (2008)
23. Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: Sparql basic graph pattern optimization using selectivity estimation. In: Proc. 17th Int. World Wide Web Conf., pp. 595–604 (2008)
24. Zou, L., Mo, J., Zhao, D., Chen, L., Özsu, M.T.: gStore: Answering SPARQL queries via subgraph matching. Proc. VLDB 4(1), 482–493 (2011)