Towards a Use Case Driven Evaluation of Database Systems for RDF Data Storage - A Case Study for Statistical Data

Matthäus Zloch[1], Daniel Hienert[1], Stefan Conrad[2]

[1] GESIS - Leibniz-Institute for the Social Sciences, Cologne, Germany
{matthaeus.zloch,daniel.hienert}@gesis.org
[2] Institute for Computer Science, Heinrich-Heine University Düsseldorf, Germany
conrad@cs.uni-duesseldorf.de

Abstract. To store Linked Data one may choose from a growing number of available database systems: from traditional relational databases to RDF triple stores, not to mention the area of NoSQL technologies. Comparisons of database systems often use benchmarks to evaluate systems with the best overall runtime performance. However, the structure of data and queries used in traditional benchmarks differ from real-world environments. They are typically not aligned with use cases from real-world applications. In this paper we investigate a use case driven approach where queries are grouped by means of more general application use cases. We evaluate and compare eight heterogeneous database systems (four relational, two graph, two column-oriented) with real-world application data and measure query runtime performance according to these use cases. The evaluation results show that the choice for a database system should not only be based on the overall query runtimes, as a naive evaluation would suggest, but preferably according to the use cases of the application itself.

1 Introduction

The research on RDF data management deals with the efficient storage and querying of RDF data. The driving factor for best performing query runtimes is the efficient implementation of the *domain model* in the *database schema* and the hereafter following physical data representation (data structures) which are specific to each database system. For implementing the schema database engineers have to overcome an *impedance mismatch* problem, which deals with the difficulty of mapping the domain model to the specific model a database system offers (cp. Figure 1). With a suitable schema design the internal database mechanisms such as cardinality estimation during the query optimization process or query rewriting can work properly, thus, returning query results faster.

As the number of available database systems grows, so does the demand for performance comparison. Although research on solutions outside the relational model has increased lately [8], relational databases are still subject to current

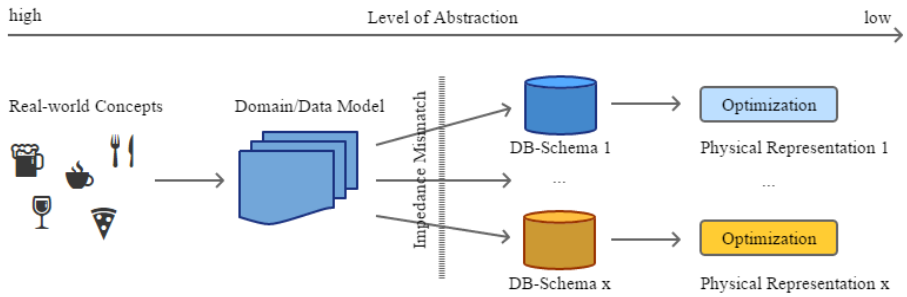

Fig. 1. Textbook modelling process. Queries are executed against a database-specific schema.

research for RDF data management [5,6,14]. The performance of schema implementations in database systems is typically estimated by benchmarks. However, benchmark results do not necessarily reflect real-world situations [1,15]. That is because many of the available benchmark suites use artificial schemata, data and queries. Some of them also address multiple use cases and various aspects such as exploration, updates, business intelligence [4], reasoning [11], and general coverage against features defined in the query language specifications [16]. Real-world queries executed against public SPARQL endpoints, however, focus only on a very small set of features [10]. Hence, the overall results of benchmarks may not be meaningful enough. Database performance issues might then be claimed to the specific type of database system in use. Given the vast amount of database system types, finding the most suitable database system for storing RDF data of an application is not a trivial task.

In this paper we address the issue of choice for a database system for RDF data management concerning optimized overall query runtime performance. In contrast to a typical benchmark approach where queries are executed in single sequences we group queries by global application use cases and study different query distributions. Each application use case contains queries that address a specific part of the model that has certain characteristics at the schema and the data level, leading to different query runtimes. Our assumption is that a use case driven approach may influence a decision about the choice for a suitable database system for an application. We consider "suitable" as the one that answers all queries in the shortest time.

The evaluation in this paper is motivated by our web portal Missy which provides Social Science research data. It uses a well-designed RDF vocabulary from the Social Sciences as domain model and data from a real-world application (cf. Section 3.1). We load the data into database systems of different types, four databases based on relational technology, two graph databases, and two databases with column-oriented storage (cf. Section 3.2). Then, we construct groups of queries identifying our application use cases (cf. Section 3.3) and introduce a general method to estimate overall performance (cf. Section 3.4). The evaluation results show that the question which database system to choose for

storing RDF data should not necessarily be made by considering overall query runtimes, as benchmarking suites would suggest, but preferably according to the use cases of the application (cf. Section 4).

2 Related Work

For storing RDF data in the relational model different strategies emerged which are dealing with the mapping of the graph structure that is implicitly given in RDF data to the relational model. An intuitive schema implementation is the *single table* representation, so-called "triple-store" or "monolithic schema" where all RDF statements are stored in one table with three columns. From the domain model perspective there is the *type-oriented* approach [19] which uses one table for each RDF data type. A third alternative uses *predicate-tables* [9] with a binary relation between subject and object per table, so-called "group-by-predicates". This schema takes the RDF graph perspective with the predicate-tables being the vertices between subject and object nodes.

Benchmark suites for RDF data focus either on *single-node* or *distributed* RDF data stores. This paper pays attention to single-node RDF data management, thus, related work is focused on that. To reproduce real-world situations some researchers focus on making benchmark suites more realistic. Aluç et al. created a benchmark called WatDiv that reveals physical design issues of RDF data management systems [1]. WatDiv addresses a wider range of problems than existing benchmark suites do, e.g. creating simple and complex queries with a varying number of features. However, it also uses artificial data and a fixed schema for testing. Saleem et al. developed FEASIBLE [15], a feature-based SPARQL benchmark that uses query logs to automatically generate queries for benchmarking frameworks.

Recent research developed new approaches to overcome the impedance mismatch problem described above and to optimize the query evaluation process. Bornea et al. considered a flexible relational representation for RDF data, a fourth "entity-oriented" alternative to classical RDF to relational model implementations [5]. They evaluated this using their benchmark suite. Aluç et al. physically fracture data according to workload, thus, introducing a schema-less and workload-driven "group-by-query" representation [2]. Pham et al. created a method to detect an emergent relational schema from RDF data to increase the quality of SPARQL query optimization [14].

NoSQL technologies, graph-databases in particular, constitute an important part in recent research activities for RDF data management. Cudré-Mauroux et al. conducted the first comparison of NoSQL databases for RDF data in a distributed setting [8]. The authors used schema and data of different benchmarks. They conclude that NoSQL systems can be competitive against distributed and native RDF stores. However, more complex queries were reported to perform poorly. Vicknair et al. evaluated a graph- and a relational database [17]. They used a schema from the domain of data provenance, formulated queries typically known from provenance systems, and created a database by a random genera-

tion of nodes and edges. They concluded that the use of graph databases seems premature for production use in their use case. Lee et al. evaluated a NoSQL and a relational database for real-world clinical data [13]. The authors used five different reasonable queries and measured query runtime. They concluded that XML databases are a promising solution but not yet ready for production use in clinical environments. The interested reader is pointed to a general overview of relational and NoSQL database systems [7]. In contrast to the related work, in this paper, real queries and real data from the Social Sciences domain are used.

3 Evaluation Design

The following section describes the setting of the evaluation. First, a brief introduction to the domain model as well as the data is given, followed by the database systems used in the evaluation. Use cases and the execution method are explained at the end of this section. In order to make the evaluation reproducible we have made all data and queries available on our website[3].

3.1 Domain Model and Data

The DDI-RDF Discovery Vocabulary[4] (DISCO) is utilized as a domain model in order to represent knowledge in the area of the Social Sciences. The main purpose of this vocabulary is to support the discovery, representation, and publication of survey metadata. It supports usage scenarios, which are typically applied by researchers from different domains such as the Social Sciences, data archives, libraries, and other statistical organizations (cf. Table 1 for examples). The implementation of the model in RDF facilitates the reuse of properties from other vocabularies. By using data linking and reasoning tools linkage to external sources in the Web of Linked Data is possible in order to enrich the metadata of resources.

Usage Scenario	**Natural Language Query**
Searching for studies by publishing agency	"Show me all the studies for the period *2000* to *2010* which are disseminated by the *ESDS Service of the UK Data Archive*."
Finding relevant studies by free text search or keyword	"Find all studies (titles, abstracts) with questions about *commuting to work*."
Searching for reusable questions using related concepts	"Find all questions which comprise *Age* in the context of *Income*."

Table 1. Examples of usage scenarios of DISCO which can be found in [18].

DISCO comprises around 25 entity types with many simple as well as more complex relationships. It contains a graph structure at its core (StudyGroup - Study - LogicalDataSet - Variable - Representation) representing the navigational structure of a dataset. It furthermore stores flat data with statistics and

[3] http://mazlo.de/papers/blink2017
[4] http://rdf-vocabulary.ddialliance.org/discovery.html

numerical values in a few entity types (CategoryStatistics, SummaryStatistics). For a detailed description of the model we refer to the specification[4] and the implementation of the model[5] in Java[6].

Data was taken from the project Missy[7] which provides systematically structured metadata for official statistics. Missy targets international researchers from the Social Sciences who are working with official microdata. Access to metadata is provided via a web portal. The Missy web application uses DISCO as its internal data model.

The original data is stored in a MySQL database of around 3GB in size. We exported the data into various formats and imported it into the other database systems. This data file covers about 45 studies with 159 research datasets (study level), around 21 thousand variables and around 3.3 thousand questions in total (variable level). Numerical values in form of statistical data is contained at the variable level with around 2 million entries. As RDF data, this subset makes around 18 million triples.

3.2 Database Systems

The selection of database systems for the evaluation was driven by objective and subjective aspects, which were *wide community acceptance* and a strength of *handling data of certain structure*, like graph-based or tabular-based; subjective aspects cover the *availability under an open source license* and *intuitive query language*, for the ease of translating queries. Further, heterogeneous database systems which implement different storage techniques and internal models are of interest. Based on this assumptions the following candidates were chosen:

Relational Technologies (1) *MySQL Community Edition* version 5.6.27[8], (2) *PostgreSQL* version 9.3.9[9], (3) *MariaDB* version 10.0.30[10]. The implementation of the domain model in Java facilitates the automatic generation of the schema[6] constituting 50 tables for 25 entity types. This schema is *type-oriented* respecting type inheritance, i.e. each entity type has its own table and the corresponding properties.
In addition we use (4) *Virtuoso6 Open-Source* version 6.1.6[11], a triple store implementation in Java and C. Virtuoso comes with "relational, graph, and document data management capabilities" and uses a relational database in the backend. It creates the relational schema of the RDF model from a graph perspective [9]. We imported data in the n-triple format into Virtuoso, which we exported from MySQL using Triplify [3]. In the n-triple format the data set constitutes around 18 million statements.

[5] https://github.com/missy-project/disco-model-impl
[6] http://hibernate.org
[7] Metadata for Official Statistics - http://www.gesis.org/missy
[8] http://www.mysql.com/products/community
[9] https://www.postgresql.org/
[10] https://mariadb.org/
[11] https://github.com/openlink/virtuoso-opensource

	Memory	Config	Index On	Query Via
(1) MySQL, (2) Postgresql, (3) MariaDB	2GB	default	all primary keys and all foreign keys	SQL+ JDBC
(5) Neo4j	2GB (default)	default + auto indexing on nodes and edges enabled	manual on node.id + automatic on nodes and edges	Cypher+ HTTP
(6) Stardog	2GB (default)	default + 1) Index on triples only enabled (command line) 2) Clustering disabled 3) Reasoning disabled	automatic	SPARQL+ HTTP
(4) Virtuoso6, (7) Virtuoso7	2GB (default)	default + 1) Clustering and segmentation disabled 2) Inference disabled 3) ResultsSetMaxRows = 1000000 4) MaxQueryExecutionTime = 600s	automatic	SPARQL+ HTTP
(8) Monetdb	default	default	default	SQL+ JDBC

Table 2. Configurations for each system used in the evaluation.

Graph-based Technologies (5) *Neo4j Community Edition* version 2.2.1[12], (6) *Stardog Community* version 4.0.5[13]. Both open source graph databases with wide community acceptance and continuous development. Neo4j exploits the mathematical model of a graph. Thus, the schema is implemented in terms of nodes and edges. Further, a node may have attributes attached. Edges represent the connection between nodes. The implicit graph model of RDF can be mapped by making each resource in RDF (a tuple in the relational model) to a node with attributes. Entity types are mapped to labels which are attached to nodes.
Stardog utilizes the native graph character of RDF data. Thus, data is stored by means of nodes and edges. During the import Stardog takes care of creating the schema, which is explicitly given in the set of n-triples. Indexes are expected to be created by Stardog itself. We used the same n-triple files for Stardog as for Virtuoso to import data.

Relational Technologies with column-oriented storage (7) *Virtuoso7 Open-Source* version 7.2.2[11] and (8) *Monetdb* version 11.25.3[14]. The main difference to Virtuoso6 is that Virtuoso7 stores data and uses indexes in a column-wise manner. Monetdb is an open source relational database also based on a columnar-oriented data storage. Like in the first group of database systems the schema *type-oriented* respecting type inheritance. We used the dump of MySQL and transformed it into import statements for Monetdb. We expect indexes to

[12] http://neo4j.com
[13] http://stardog.com
[14] https://www.monetdb.org/

be created automatically, since Monetdb relies on "its own decision to create and maintain indexes for fast access".

All three technologies have different internal models to represent data at the logical and physical level. Table 2 shows an overview of the configuration of the systems. Please note that each system is used with the default configurations that are provided by the installers. For MySQL we raised the buffer pool size to 2GB to have comparable RAM conditions for every system. For Virtuoso and Stardog inference capabilities were disabled, since this would slow down performance and distort the results.

3.3 Queries and Use Cases

We formed a set of overall 39 queries from three different sources: the official sample set of usage scenarios created for the domain model [18] (7 queries), the Missy application itself which runs a reasonable and advanced set of queries against the model (24 queries), and a subset of validation queries recommended for this vocabulary which were developed by Bosch et al. in [12] (8 queries). All queries are read-queries. A full list of all queries in detail can be found on the paper website[3].

Like in most applications, there are different actors in the system, thus, different contexts and intentions from which and with which a query may be executed.

Use case	Description	Queries
UC1: Navigation	A user browses through available datasets.	{ sd[1-3], dp[1-4], vd[1-3,5-7], qd[1-6] }
UC2: Statistics	A user wants to see details for frequencies of a variable.	{ vf[2-4], vf2b, vf3b }
UC3: Validation	A developer runs validation queries over the database.	{ dsv[1-6], dsv5b, dv }
UC4: User Query	A user executes a query over the SPARQL endpoint.	{ duc[1-7] }

Table 3. Four classes of queries, which are constructed according to business use cases of the application. Each set of queries addresses a specific area in the schema. The abbreviations stand for "dp" - "datapoint selection", "sd" - "study details", "vd" - "variable details", "qd" - "question details", "vf" - "variable frequencies", "dsv" - "domain specific validation", "dv" - "database validation", "duc" - "DISCO use case"

We assigned each query to a global business use case that is relevant in our application and which satisfies different user roles and requirements (cf. Table 3). UC1 contains the group of queries that are executed in the context of external users who browse available datasets on the website. These queries address parts of the schema that has graph-like characteristics, since most of the relationships consist of many-to-many relationships (StudyGroup - Study - LogicalDataSet -

Variable - Question). Queries in UC2 are more complex in terms of runtime. They are executed in the context of a variable, i.e. when a user has found a dataset and now wants to obtain statistics on specific variables. UC3 contains queries which address tabular-like structures and use more complex query functions like inner queries, aggregation, and group-by-having constructs. Queries in UC3 are executed by application developers to validate the data in the database. UC4 consists of a subset of queries (7 out of 15) which can be found in [18]. These queries are considered to be executed by external users, e.g. over a SPARQL-endpoint. Table 4 shows some statistics for the used queries.

Because the database systems used for evaluation use different query languages all queries have to be transformed into the query language of each system. That is SQL for all relational technologies and Monetdb, Cypher for Neo4j, and SPARQL for Stardog and Virtuoso 6/7. All queries were transformed to have an equivalent counterpart in the other languages and will return the same number of results.

Feature	UC1	UC2	UC3	UC4
No. of queries	19	5	8	7
No. of tables joined, max (avg)	18(9)	22(19)	6(2)	12(7)
No. of results, max. (avg)	275(24)	17(9)	9490(3221)	25(16)
Use of aggregate function	DISTINCT COUNT	DISTINCT	SUM GROUP-BY	DISTINCT
Average no. of filters (where-clause)	3	4	2	2
Maximum no. of nested queries	0	1	1	0

Table 4. Statistics of queries used for the evaluation in the language SQL. The reader is referred to the website of this paper[3] to see the complete list of queries.

3.4 Evaluation Method

In order to show the differences between a single list of queries and the use case driven approach we created two different scenarios for evaluation.

Standard Benchmark Approach In this scenario each of the 39 queries is executed 10 times by the database systems. This results in 390 queries (executions) in total. Each execution of a query is followed by a restart of the database system and the clearing of all system caches (as described in 3.5). This way, we get the so-called *cold start query execution time* which we expect to be unbiased by any cache. After each execution the runtime is saved and at the end of the evaluation run the average runtime per query is calculated.

Use Case Driven Approach In this approach all queries are executed by the database systems only in the context of their use case, e.g. for UC1 Navigation. This way, a first impression for use case driven performance is obtained for all of the use cases. However, in real-world environments a mixture of queries throughout all use cases can be observed. For instance, in 80% of the time users

navigate (UC1) and 20% of the queries come from other use cases (UC2: viewing statistics or UC3: validation). To reproduce this we compute further sequences of queries to measure the difference in overall runtime. We start by using queries *which are explicitly* relevant to a use case (100/0 ratio as described above) and successively increase the occurrence of queries *which are not* relevant to the corresponding use case (non-use case queries). We denote these ratios as 90/10, 80/20, 70/30, 60/40, and 50/50. The total runtime for a given use case and ratio is the sum of all query runtimes in a sequence. Each sequence is shuffled once in preparation for the evaluation so that each database system will have to execute the same sequence of queries per ratio.

Since working with these sequences per use case and database system would require a lot of executions and would take days for computation, we calculate the total runtime using known execution times for warm and cold starts of individual queries. We calculate the total runtime T for a given database system db and use case UC with the help of this formula:

$$T_{db,uc} = \sum_{i=1}^{j} r(q_i) + \sum_{k=j+1}^{n} r(q_k) \tag{1}$$

where $i..j$ is the index of *relevant* queries (part of the given use case UC), $k..n$ the index of *not relevant* queries (all other queries). For example:

$$\begin{aligned} T_{db1,uc1} =& (r(sd1) + r(sd2) + ... + r(qd6)) + \\ & (r(vf1) + r(vf2) + ... + r(dsv1) + r(dsv2) + ...) \end{aligned} \tag{2}$$

$r(q_x)$ is the look-up function for the known runtime of query q_x in a specific database system and use case. In particular

$$r_{db,uc}(q_x) = \frac{r_{db,cold}(q_x) + r_{db,warm}(q_x) * (n_{uc} - 1)}{n_{uc}} \tag{3}$$

which respects (1) the cold start runtime, (2) the warm start runtime (with caching mechanisms) of the database, and the total number of times n_{uc} the query will be executed in that particular use case.

3.5 Execution Environment

As already mentioned we focus on centralized single-node solutions and standard hardware. Thus, operating system, database installation, test data, and client software reside all on one server. The evaluation was made with an Intel(R) Core(TM) i5 650 CPU quad core processor running 3.2Ghz each, 4MB internal cache, 1333Mhz FSB, 8GB of main memory, and 80GB main storage. The operating system is Linux, Fedora 21, kernel version 4.1.3, with the latest update of packages. Processes that are not relevant to the execution and operating system were terminated. In cold start query executions file system caches where dropped with `sync && echo 3 > /proc/sys/vm/drop_caches` right before an execution. For the purpose of automation a lightweight client software was written. It executes a given sequence of queries, restarts the database and drops caches (in cold start scenarios) and measures the query execution time.

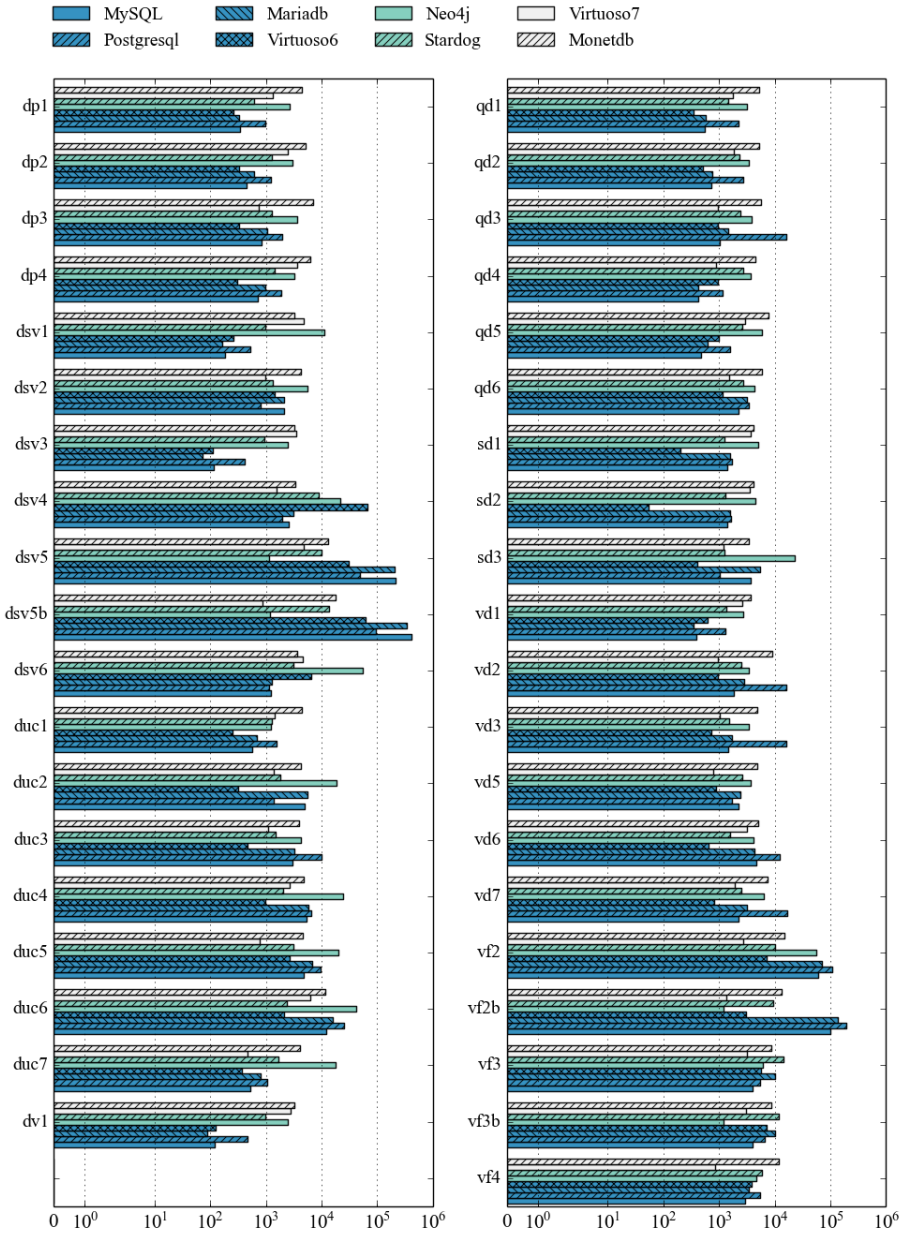

Fig. 2. Average query runtime of 10 cold start executions for 39 queries on each database system. For better comparison the x-axis with query runtimes in milliseconds is log-scaled. The three colors group similar database technologies. See Table 5 for total runtimes.

4 Evaluation Results

4.1 Standard Benchmark Approach

Figure 2 shows query runtimes for 39 different queries on all database systems based on the average of 10 cold start executions (note: x-axis is log-scaled). Virtuoso7 performs best with 85sec in total and has best times in 8 individual queries. Note the high standard deviation on cold start results (Table 5). Stardog is the second best with overall 139sec, but has zero best times in single query performance comparison. Virtuoso6 is on the third place with a total runtime of 211sec and the best performance in 20 single queries. Worst performance has MySQL with a total runtime of 862sec and being the fastest in two queries. It can be observed that in the group of relational databases (MySQL, MariaDB, PostgresQL) the overall performance is massively influenced by only a small number of queries. For example, for the queries dsv5 and dsv5b the performance is 2.5 magnitudes worse than the best-performer. Similarly for the queries vf2 and vf2b with up to 2 magnitudes (cp. Figure 2).

	Overall Query Runtime (ms)	Best Time in .. Queries	Mean Standard Deviation
MySQL	***865,566***	2	802.745
Postgresql	845,651	2	691.900
MariaDB	623,358	4	180.967
Virtuoso6	211,515	20	174.263
Neo4j	391,663	3	301.498
Stardog	139,779	0	**115.029**
Virtuoso7	**85,886**	8	***3,264.709***
Monetdb	251,826	0	336.861

Table 5. Overall query runtimes and mean standard deviation values for 10 executions of each query on each of the database systems. Highlighted are **lowest** and ***highest*** runtimes.

4.2 Use Case Driven Approach

Figure 3 let us compare total runtimes for the ratio 100/0, which contains only use case specific queries. Database systems of the same technology are grouped by color. None of the systems perform best or worst throughout all use cases. In the group of traditional relational databases (blue color) MySQL performs best in all use cases. In the group of graph databases (green color) Neo4j outperforms Stardog in all use cases except UC3. In UC2 Neo4j even has a significant shorter runtime. The group of relational technologies using column-oriented storage (grey color) seems to have a good performance in all use cases. Overall, UC3 consumes most of the time in comparison to the other use cases.

Table 6 shows a detailed overview of query runtimes for the ratios 100/0 and 50/50. It shows that for a use case driven approach the systems perform

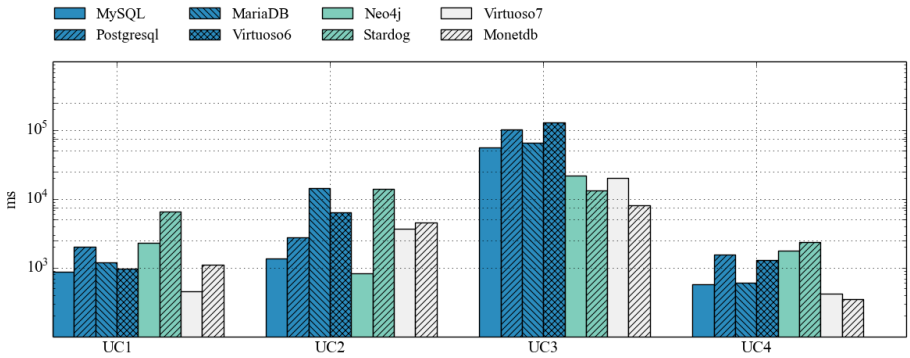

Fig. 3. Comparing total runtimes for ratio 100/0, which contains use case specific queries. Database systems of same technology are grouped by color. None of the systems perform best or worst throughout all use cases.

differently in each use case. Virtuoso7 performs best and Stardog worst in UC1 ratio 100/0. In UC2 Neo4J performs best and MariaDB worst. Monetdb performs best and Virtuoso6 worst in UC3, where in UC4 Monetdb performs best and Stardog worst. Since all queries are equally distributed in ratio 50/50, we find very similar runtimes for each of the database systems in all use cases.

	UC1		UC2		UC3		UC4	
	100/0	50/50	100/0	50/50	100/0	50/50	100/0	50/50
MySQL	876	74,317	1,356	73,988	56,003	74,730	571	93,376
Postgresql	2,048	127,512	2,802	120,311	103,595	140,538	1,571	144,170
MariaDB	1,191	100,680	***14,352***	99,121	66,135	102,760	600	127,255
Virtuoso6	977	***144,818***	6,476	***138,950***	***129,641***	***140,610***	1,303	***144,575***
Neo4j	2,328	39,831	**829**	39,731	21,917	36,654	1,778	38,621
Stardog	***6,494***	40,842	13,903	39,426	13,388	40,544	***2,356***	41,022
Virtuoso7	**452**	26,339	3,685	26,629	20,413	26,943	420	29,315
Monetdb	1,120	**15,186**	4,572	**16,666**	**8,096**	**16,038**	**350**	**15,697**

Table 6. Total query runtimes in milliseconds for ratio 100/0 and 50/50 (cf. Figure 4). Highlighted are **lowest** and ***highest*** runtimes per use case and ratio. Since all queries are equally distributed in ratio 50/50, very similar runtimes for all use cases and databases systems can be observed in this ratio.

Figure 4 shows that lowest values for runtime performance can be found in ratio 100/0 (cf. Figure 3 which depicts a detailed view on that). This changes when the number of queries increases which do not belong to a use case. Generally, there are two types of curves: a) one that starts with low value at 100/0 and increases significantly in the next ratio 90/10, following a continuous decrease in value in the next ratios; and b) one that starts with low value at 100/0 than following a logarithmic shape. For the group of traditional relational technologies (blue color) there is a high rise in runtimes in 90/10 and a following slow decrease in runtimes in the subsequent ratios until 50/50. Virtuoso6 remains rather stable

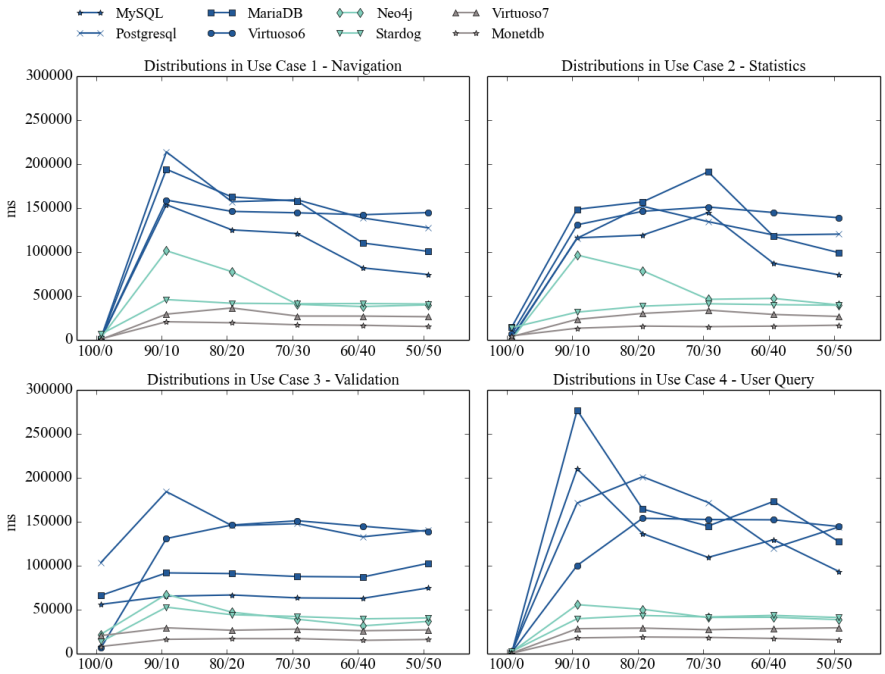

Fig. 4. Comparing total runtimes for all use cases and ratios. Database systems of same technology are grouped by color. Successively increasing the frequency of queries which do not belong to a use case results in different behaviour of the database systems.

in UC1-3, with a continuous increase in runtime in UC4 from 100/0, 90/10 to 80/20. Same can be observed for the group of graph databases (green color), although the increase in runtime is not that high. Neo4j's increase in runtime is much higher in ratio 90/10 and converges to that of Stardog until 70/30. From there it is very similar to the runtime of Stardog with mutual marginal outperforming. The increase of non-use case queries does not seem to influence overall runtimes for the group of relational column-stores. After a slight increase in 90/10, the runtimes remain quite stable. Without doubts, Virtuoso7 and Monetdb outperform the other database systems in all use cases from ratio 90/10 to 50/50.

5 Discussion

Looking at Figure 4, it seems that each plot is tripartite and that there are three groups where the overall runtimes and the shapes of curves are similar. These three groups align well with the database technologies that were used in the evaluation: (1) the traditional relational model, (2) the graph model and (3) the relational model with column-oriented storage (Virtuoso7 and Monetdb).

Except for Virtuoso6 all of the other systems in the first group implement the schema from the domain model perspective, i.e. *type-oriented* [19]. In our case, the schema consists of 50 tables for 23 entity types. The schema implementation

of Virtuoso is implemented from the implicit RDF graph model perspective, i.e. *property-oriented* [9]. Thus, different schema implementations yield to different queries and query runtimes. However, the performance of Virtuoso 6/7 is surprising, since it also has to translate all SPARQL queries into SQL during query evaluation.

We would have liked to compare this aspect for the graph-based technologies (Neo4j and Stardog) likewise, but unfortunately the internal schemata are not easily accessible. We suppose that each graph database implements a different data structure to store data in. Graph databases were developed to take advantage of specific properties of graphs, e.g. the number of steps needed to get from one node to another, or graph traversal. We did not take advantage of these features in our data set and queries, hence the systems could not show their full strengths. However, we can see that the influence on the schema for these two systems is less serious. Besides the spikes for Neo4j in 90/10 and 80/20 in all use cases, both systems have very similar runtimes.

We did not expect total query runtimes to be the highest in ratio 90/10 and to decrease in the following ratios for most of the systems. As the frequency of non-use case queries increase (from 90/10 to 50/50), we expected runtime to gradually either decrease or increase for a system. One explanation is that caching mechanisms for query evaluation applies better as queries are executed more often. Further, each system has at least one query which is slow in execution (cf. Figure 2) and the runtime for the first execution is the highest in most systems (cold start). In 90/10 there is 10% of non-use cases queries and thus, a slow query only executes approximately between 1-5 number of times. This results in a high average value for that system. Further, in all use cases the runtimes are quite similar for all systems in ratio 50/50 (cf. Table 6). This is caused by the balanced ratio of use case and non-use case queries in this sequence, which also shows that the execution results are consistent.

We expected all systems to show that 90/10-phenomenon, but the third group of database technologies (Virtuoso7 and Monetdb) seem to be immune to that and their execution times are quite stable. Even Stardog behaves similarly. On the other hand, stable runtimes suggest for a better exploitation of database caches, since rare queries are encountered more often. Also, we assume that this behaviour is because of the main-memory usage of these systems.

Comparison Standard Benchmark and Use Case Approach The results of our case study give a more distinctive impression of database performance. This would lead to a different decision regarding the choice of a database system.

The standard benchmark approach would rank 1. Virtuoso7, 2. Stardog, 3. Neo4J, 4. Monetdb and on the very last rank 8. MySQL (cp. Table 5). As already mentioned, total runtime of a database system (here: all relational databases) is massively influenced by a small number of queries with exceptional long runtimes that add to the overall runtime. This is enforced by using only mean cold start query runtimes.

In the use case driven approach with ratio 100/0 one would choose a database system depending on the use case only. That would be Virtuoso7 for queries of

UC1, Neo4j for UC2, and Monetdb for queries of UC3 and UC4 (cp. also Table 6).

By looking at mixtures of use case queries that reflects real-world user behavior Monetdb now gives a much better picture. In the group of relational technologies MySQL performs best in every use case. Looking at the plots in Figure 4) one would now choose for Monetdb and Virtuoso7 in the first place, where a very stable performance between all ratios can be observed. The performance of graph-based technologies is comparable.

This shows that a use case driven evaluation can give a more fine-grained view on database performance to obtain realistic query runtimes.

Limitations We believe that our domain model and data used in this study represent well a real-world situation in single-server environments. However, the queries may not be diverse enough for a more general statement of which database should be used in which use case and ratio. Nevertheless, this work does motivate to have a closer look at the queries and their execution frequencies in an application. Further, the volume of data and the distribution of data within the schema has a high impact on query runtime. In our case, the volume of structural and numerical data is very unbalanced (cf. Section 3.1). However, the data set and the queries are pretty common for statistical data. Increasing the number of entities residing in the graph-part of the schema (StudyGroup - Study - LogicalDataSet - Variable - Question) would probably result in a shift towards other database technologies than our results would suggest.

6 Conclusion and Future Work

This paper addresses the issue of decision making about which of the available database system types is best suited for an application model and its data. Our hypothesis is that overall runtimes are different and probably lower in a use case driven grouping of queries. We investigate this by measuring query runtimes of heterogeneous database system technologies and different schema implementations. The two evaluation methodologies differ by (1) using groups of queries according to application use cases and (2) by using a realistic model to compute query runtimes based on cold- and warm start execution times.

One can see from the evaluation results that each system performs differently according to a given use case and ratio. From that we conclude that the overall performance of an application is influenced by (1) characteristics of the schema implementation and the executed queries, (2) the ratio of use case and non-use case queries, (3) realistic query runtime modeling and (4) the internal model of the database system. In order to fully utilize the strengths of each database system a *use case-aware* application should be developed which respects the mentioned aspects.

In future work, we want to investigate various schema implementations with respect to runtime performance in RDF data management systems. We also consider implementing the proposed use case driven approach in a framework or to extend an existing benchmark suite. This framework would (a) compute sequences of queries with different distributions according to a set of use cases

and their corresponding queries, (b) estimate query runtimes according to the look-up function 1, (c) give feedback to the database engineer about the results. We also consider respecting the characteristics at schema and data level to let the results of the framework be more reliable and realistic.

References

[1] Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified Stress Testing of RDF Data Management Systems. In: ISWC (2014)
[2] Aluc, G., Özsu, M.T., Daudjee, K.: Workload Matters: Why RDF Databases Need a New Design. PVLDB, pp. 837–840 (2014)
[3] Auer, S., Dietzold, S., Lehmann, J., Hellmann, S., Aumueller, D.: Triplify: Light-weight Linked Data Publication from Relational Databases. In: Proc. of the 18th International Conference on World Wide Web. pp. 621–630. ACM (2009)
[4] Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. International Journal on Semantic Web and Information Systems (IJSWIS) pp. 1–24 (2009)
[5] Bornea, M.A., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., Bhattacharjee, B.: Building an Efficient RDF Store over a Relational Database. pp. 121–132. SIGMOD, ACM (2013)
[6] Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering SPARQL Queries over Relational Databases. Semantic Web Journal (2016)
[7] Cattell, R.: Scalable SQL and NoSQL Data Stores. SIGMOD pp. 12–27 (2011)
[8] Cudré-Mauroux, P., Enchev, I., Fundatureanu, S., Groth, P.T., Haque, A., Harth, A., Keppmann, F.L., Miranker, D.P., Sequeda, J., Wylot, M.: NoSQL Databases for RDF: An Empirical Evaluation. In: ISWC. pp. 310–325. Springer (2013)
[9] Erling, O.: Virtuoso, a Hybrid RDBMS/Graph Column Store. IEEE (2012)
[10] Gallego, M.A., Fernndez, J.D., Martnez-prieto, M.A., De La Fuente, P.: An Empirical Study of Real-world SPARQL Queries. In: 1st International Workshop on Usage Analysis and the Web of Data (USEWOD) (2011)
[11] Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics, pp. 158–182 (2005)
[12] Hartmann, T., Zapilko, B., Wackerow, J., Eckert, K.: Constraints to Validate RDF Data Quality on Common Vocabularies in the Social, Behavioral, and Economic Sciences. Computing Research Repository (CoRR), abs/1504.04479 (2015)
[13] Lee, K.K.Y., Tang, W.C., Choi, K.S.: Alternatives to Relational Database: Comparison of NoSQL and XML Approaches for Clinical Data Storage. Computer Methods and Programs in Biomedicine, pp. 99–109 (2013)
[14] Pham, M.D., Boncz, P.A.: Exploiting Emergent Schemas to make RDF Systems More Efficient. In: ISWC. pp. 463–479 (2016)
[15] Saleem, M., Mehmood, Q., Ngonga Ngomo, A.C.: FEASIBLE: A Feature-Based SPARQL Benchmark Generation Framework. In: ISWC (2015)
[16] Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP2Bench: A SPARQL Performance Benchmark. Computing Research Repository (CoRR), (2008)
[17] Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., Wilkins, D.: A Comparison of a Graph Database and a Relational Database: a Data Provenance Perspective. In: ACM Southeast Regional Conference. p. 42. ACM (2010)
[18] Vompras, J., Gregory, A., Bosch, T., Wackerow, J.: Scenarios for the DDI-RDF Discovery Vocabulary. DDI Working Paper Series (2015)
[19] Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D.: Efficient RDF Storage and Retrieval in Jena2. pp. 120–139. SWDB, CEUR-WS.org (2003)