SP2Bench: A SPARQL Performance Benchmark

Michael Schmidt, Thomas Hornung, Michael Meier, Christoph Pinkel, and Georg Lausen

Abstract A meaningful analysis and comparison of both existing storage schemes for RDF data and evaluation approaches for SPARQL queries necessitates a comprehensive and universal benchmark platform. We present SP2Bench, a publicly available, language-specific performance benchmark for the SPARQL query language. SP2Bench is settled in the DBLP scenario and comprises a data generator for creating arbitrarily large DBLP-like documents and a set of carefully designed benchmark queries. The generated documents mirror vital key characteristics and social-world distributions encountered in the original DBLP data set, while the queries implement meaningful requests on top of this data, covering a variety of SPARQL operator constellations and RDF access patterns. In this chapter, we discuss requirements and desiderata for SPARQL benchmarks and present the SP2Bench framework, including its data generator, benchmark queries and performance metrics.

16.1 Introduction

In the recent years, many proposals for the efficient evaluation of SPARQL [35] have been made. These approaches comprise a wide range of optimization techniques, including normal forms [23], graph pattern reordering based on selectivity estimations [22, 30] (similar to relational join reordering), syntactic rewriting of SPARQL queries [17, 27], RISC-style query processing [22] and Semantic Query...
Optimization [18]. In addition, there has been a corpus of research on specialized indices [13, 16] and storage schemes [1, 4, 7, 15, 31, 36] for RDF data [34], with the aim to provide efficient data access paths. Another notable line of research is the translation of SPARQL queries into established data models like SQL [9, 10] or Datalog [3, 24], thus facilitating SPARQL evaluation with traditional engines and allowing to exploit optimization techniques implemented in existing systems.

A meaningful analysis and comparison of all these optimization approaches necessitates a comprehensive and universal benchmark platform. Systematic benchmarking has been an important topic in database research from the beginning [12] and, over time, a variety of benchmarks have been proposed, covering both different aspects of data processing and different data models. Examples include the prominent TPC\(^1\) benchmark suite for Relational data, the OO7 [8] benchmark for Object-oriented databases, and the XMark [25] benchmark for the XML data model. Coming along with the proliferation of the Semantic Web, benchmarking has become an increasingly important topic in the context of Semantic Web data formats like RDF(S) [34] and OWL [33]. As a response, also in this context several benchmark platforms have been developed. These platforms address structural aspects of the data (e.g., [21]) as well as the issue of efficient data processing (e.g., [2, 6, 14]).

One well-known benchmark that falls into the latter category is the Lehigh University Benchmark (LUBM) [14]. It comes with a data generator, which allows to generate synthetic OWL documents, a set of 14 benchmark queries, implementing different reasoning tasks with varying complexity over the generated data, and several performance metrics, allowing to evaluate and compare benchmark results. The LUBM benchmark suite has a strong focus on testing the inference and reasoning capabilities of Semantic Web repositories and—although there exist SPARQL versions of the benchmark queries—is not primarily a SPARQL benchmark.

The Barton Library benchmark [2] is a more traditional database benchmark in the sense that it shifts the focus from inference and reasoning to query answering. It uses the real-world RDF Barton online catalog as underlying data. The benchmark queries implement tasks that are derived from a typical browsing session through the catalog data, so Barton is highly use-case driven. The queries are encoded in SQL and it is assumed that RDF data is stored in a Relational database (as proposed in [1]). Given the current SPARQL specification [35], not all of the Barton benchmark queries can be expressed in SPARQL, due to missing support for aggregation.

The application-oriented Berlin SPARQL Benchmark [6] (BSBM) is SPARQL-specific and tests the performance of SPARQL engines in a prototypical e-commerce scenario. Also BSBM is use-case driven and provides a set of benchmark queries that are intended to be run in a work load fashion over generated data sets of different size. This setting is particularly interesting to compare the overall performance of engines that expose SPARQL endpoints via the SPARQL protocol.

In this chapter, we present the SP\(^2\)Bench SPARQL Performance Benchmark [26, 28], which comprises a data generator and a set of queries, implementing meaningful requests on top of the generated data. SP\(^2\)Bench is freely available online in a

---

\(^1\)See http://www.tpc.org.
ready-to-use format. The focus and design goals of SP²Bench vary from the benchmark projects discussed before. In particular, SP²Bench differs in that it is neither application-oriented nor use-case driven, but falls into the class of language-specific benchmarks. This means that, compared to the other benchmarks, the document and query design in SP²Bench is not driven by a specific use-case, but instead specifically laid out to test common SPARQL constructs, operator constellations, and a variety of RDF data access patterns. In this line, SP²Bench covers a broad range of challenges that SPARQL engines might face in different contexts and constitutes a benchmark that allows for comprehensive performance evaluation, rather than performance assessment in a specific, application-driven scenario. The SP²Bench queries, which differ in their characteristics and complexity, are not intended to be evaluated in a work load setting, but rather on a one by one basis, where each query poses different challenges to the tested SPARQL engine and might help to identify deficiencies in the underlying evaluation strategy. With these design goals, SP²Bench allows to assess the generality of optimization approaches and to compare evaluation strategies in a universal, application-independent setting.

The first component of SP²Bench is the data generator, which supports the creation of DBLP-like models in RDF format. The generated documents mirror vital key characteristics and distributions found in the original DBLP database [19], making it possible to create arbitrarily large documents with realistic data that exhibits many real-world characteristics. The data mimics natural correlations between entities, such as power law distributions and limited growth curves. Complementary to the generator, SP²Bench comprises 17 meaningful queries specified over the generated documents. These queries cover important SPARQL constructs, operator constellations, and vary in their characteristics, such as complexity and result size. The detailed knowledge of data characteristics, which is established by an elaborate analysis of the original DBLP database, makes it possible to predict the challenges that the queries impose on SPARQL engines. This, in turn, contributes to the understanding of the benchmark and facilitates the interpretation of benchmark results.

Structure We start with a discussion of general and SPARQL-specific desiderata for benchmarks in Sect. 16.2, including a summary of design decisions made in the SP²Bench framework. Subsequently, in Sect. 16.3 we turn towards a study of the SP²Bench data generator. This discussion includes an analysis of key characteristics of the DBLP data set, which forms the basis for the implementation of the data generator. The profound knowledge of the DBLP main characteristics then helps to understand the key challenges of the SP²Bench queries, which are presented in Sect. 16.4. The chapter ends with a discussion of possible performance metrics for the SP²Bench suite (Sect. 16.5) and a short conclusion (Sect. 16.6).

2See http://dbis.informatik.uni-freiburg.de/index.php?project=SP2B.
3DBLP [19] is a well-known bibliographic library that contains publications made in the area of databases and, more generally, Computer Science.
16.2 Benchmark Design Decisions

A central aspect in the design of a benchmark is the choice of an appropriate domain. Clearly, the domain of a language-specific benchmark should not only constitute a representative scenario that captures the philosophy behind the data format, but also leave room for challenging benchmark queries. With the choice of the DBLP [19] library, a bibliographic database that contains a large collection of publications in the area of Computer Science, SP²Bench satisfies both desiderata. First, the RDF data format has been particularly designed to encode metadata (cf. [34]), which makes DBLP an excellent candidate for an RDF scenario. Further, as shown in [11], DBLP reflects interesting social-world distributions. One might expect that such distributions are frequently encountered in the Semantic Web, which integrates a great many of individual databases into one global database and therefore can be seen as a large social network. As an example, it has been shown in [32] that power-law distributions are naturally contained in large RDF Schema specifications. These observations justify the choice of DBLP as the underlying scenario.

Rather than using an RDF version of the existing DBLP data set (such as [5]), SP²Bench comes with a generator that supports the creation of arbitrarily large DBLP-like documents in RDF format, hence overcoming an upper limit on the size of benchmark documents. The generator itself relies on an in-depth study of characteristics and relationships between entities found in the original DBLP database, comprising the analysis of data entities (such as articles and authors), their properties, frequency and also their interaction. Consequently, the generated documents mimic a broad range of natural, social-world distributions such as power laws (found in the citation system or the distribution of publications among authors) and limited growth curves (e.g., the increasing number of venues and publications over time).

Requirements for Domain-specific Benchmarks In the Benchmark Handbook [12], four key requirements for domain specific benchmarks are postulated. First, a domain-specific benchmark should be (1) relevant, thus testing typical operations within the specific domain. Second, the benchmark should be (2) portable, i.e., should be executable on different platforms. Third, such a benchmark should be (3) scalable, which particularly this means that it should be possible to run the benchmark on both small and very large data sets. Last but not least, a benchmark must be (4) understandable, since otherwise it will not be adopted in practice.

For a language-specific benchmark, the relevance requirement (1) suggests that queries implement realistic requests on top of the data. Further, the queries should not focus on verifying the correctness of the tested engine, but on common operator constellations that impose particular challenges. To give a concrete example for the manifestation of these ideas in SP²Bench, two benchmark queries (i.e., $Q_6$ and $Q_7$) test negation, which (under closed-world assumption) can be expressed in SPARQL through a combination of operators \texttt{OPTIONAL}, \texttt{FILTER}, and \texttt{BOUND} (cf. [3]).

Requirements (2) portability and (3) scalability mainly bring along technical challenges concerning the implementation of the data generator. Addressing those, the SP²Bench data generator is deterministic, platform independent and accurate.
w.r.t. the desired size of generated documents. Furthermore, the C++ implementation is both efficient and effective, i.e., it gets by with a constant amount of main memory, making it possible to generate arbitrarily large RDF documents.

Finally, from the viewpoint of an engine developer, a benchmark should give hints on deficiencies in the design and implementation of the respective engine. This is where the (4) understandability requirement comes into play, i.e., it is important to keep queries simple and understandable. At the same time, they should still be challenging and leave room for diverse optimizations. In this regard, the SP²Bench queries are carefully designed in such a way that they are amenable to a wide range of optimization strategies. Ultimately, also the fact that these queries operate on top of data with realistic, natural distributions contributes to the understanding of the benchmark queries and permits to predict the challenges that the queries impose to SPARQL engines, allowing to better interpret the benchmark results.

16.3 The SP²Bench Data Generator

Having discussed general aspects of benchmarking and the SP²Bench design decisions, we now turn towards a study of the DBLP database [19], which lays the foundations for the implementation of the SP²Bench data generator. We analyze properties, relations and distributions of bibliographic entities and persons in the DBLP data. The study of correlations in scientific production is not new and has first been performed in [20]. A previous study of characteristics of DBLP has been presented in [11], showing that DBLP—restricted to publications and authors in the database area—reflects many distributions typically encountered in social networks and, with this regard, it forms a “small world” on its own. While the latter analysis forms valuable groundwork for the study in this section, our approach here is more pragmatic: we pursue the goal to approximate distributions by concrete functions that can be used to implement the SP²Bench data generator. With this goal in mind, we approximate distributions found in the original database by function families that naturally reflect the characteristics of interest, such as logistics curves for limited growth scenarios or power equations for power law distributions.

16.3.1 Characteristics of DBLP Data

In the following, we sketch different aspects of the DBLP analysis that were considered in the design of the SP²Bench data generator. We leave out technical details, referring the interested reader to [28] for a deeper, more technical discussion.

16.3.1.1 Structure of Document Classes

The starting point for our study is the XML version of the DBLP database [19]. Abstracting from the details, this database contains nine different types of bibliographic entities, namely ARTICLE, INPROCEEDINGS, JOURNAL, PROCEEDINGS,
Table 16.1  Probability distribution for selected attributes and document classes

<table>
<thead>
<tr>
<th></th>
<th>Article</th>
<th>Inproc.</th>
<th>Proc.</th>
<th>Book</th>
<th>WWW</th>
<th>PhDTh.</th>
<th>MastTh.</th>
<th>Incoll</th>
</tr>
</thead>
<tbody>
<tr>
<td>author</td>
<td>0.9895</td>
<td>0.9970</td>
<td>0.0001</td>
<td>0.8937</td>
<td>0.9973</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.8459</td>
</tr>
<tr>
<td>cite</td>
<td>0.0048</td>
<td>0.0104</td>
<td>0.0001</td>
<td>0.0079</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0047</td>
</tr>
<tr>
<td>editor</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.7992</td>
<td>0.1040</td>
<td>0.0004</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>isbn</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.8592</td>
<td>0.9294</td>
<td>0.0000</td>
<td>0.0222</td>
<td>0.0000</td>
<td>0.0073</td>
</tr>
<tr>
<td>journal</td>
<td>0.9994</td>
<td>0.0000</td>
<td>0.0004</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>month</td>
<td>0.0065</td>
<td>0.0000</td>
<td>0.0001</td>
<td>0.0008</td>
<td>0.0000</td>
<td>0.0333</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>pages</td>
<td>0.9261</td>
<td>0.9489</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.6849</td>
</tr>
<tr>
<td>title</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

**BOOK, INCOLLECTION, PHDTHESIS, MASTERSTHESIS and WWW documents.** We call these entities *document classes* and instances thereof *documents*.

Each document in the DBLP XML database is described by a set of *attributes*, such as *author*, *cite*, *editor*, *title* or *pages*. As one might expect, the document classes differ in their structure; even instances of the same document class do not necessarily have the same set of describing attributes. For instance, attribute *pages* is never associated with WWW documents, but typically (though not necessarily) used in combination with INPROCEEDINGS documents.

A first and very basic step in the analysis of the DBLP data therefore is an investigation of the structure of document classes. We illustrate our analysis in Table 16.1, which—for selected attribute/document class pairs—shows the probability that the attribute describes a document of the respective class. To give an example, about 92.61% of all ARTICLE documents are described by attribute *pages*, but none of them has an *editor* associated. In contrast, about 79.92% of all PROCEEDINGS documents are described by one or more *editor* attributes. This probability distribution forms the basis for generating document class instances of any type.

We note that the latter distribution does not yet consider the issue of attributes that are repeatedly used to describe a single document, e.g., article documents are typically described by several *author* attributes (i.e., one *author* attribute for each article author). Tackling this issue, we next turn towards an investigation of repeated attributes. We will exemplarily discuss our analysis for attribute *cite*, which is used to model outgoing citations of bibliographic entities.

Figure 16.1(a) shows, for each document that is described by at least one *cite* attribute, the probability (y-axis) that the document has exactly *n* cite attributes (x-axis). We observe that, given a publication with at least one outgoing citation, the average number of outgoing citations is about fifteen. Recalling the goal to approximate the distributions encountered in DBLP by natural function families, we decided to use bell-shaped Gaussian curves for approximating the distribution of repeated *cite* attributes, i.e., functions of the form

\[ p^{(\mu, \sigma)}_{\text{gauss}}(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-0.5\left(\frac{x-\mu}{\sigma}\right)^2} \]  

(16.1)
where parameter $\mu \in \mathbb{R}$ fixes the $x$-position of the peak and $\sigma \in \mathbb{R}_{>0}$ specifies the statistical spread. Such functions are typically used to model normal distributions.\(^4\)

The approximation function for the distribution of outgoing citations that is implemented in our data generator hence is an instance of the Gaussian curve equation above, obtained by choosing appropriate values for parameters $\mu$ and $\sigma$.

The analysis for other repeated attributes in DBLP, such as `editor` and `author`, is very similar and therefore we omit the details. A notable observation in the context of the `author` attribute was that the average number of authors per publication has steadily increased over time. Similar observations were made in [11] and explained by the increasing pressure to publish and the proliferation of new communication platforms, like the Internet, which facilitates the collaboration among authors. Due to the prominent role of authors in DBLP, this increase in the average number of authors is also implemented in the SP\(^2\)Bench data generator.

### 16.3.1.2 Development of Document Classes over Time

Another important subject of our DBLP study is the development of document class instances over time. Figure 16.1(b) exemplarily plots the number of `PROCEEDINGS`, `JOURNAL`, `INPROCEEDINGS` and `ARTICLE` documents as a function of time. Note that the $y$-axis is in log scale. It can be seen that inproceedings and articles are closely coupled to the proceedings and journals, respectively. For instance, there are always about 50–60 times more inproceedings than proceedings. This indicates that the average number of inproceedings per proceeding is stable over time. Similar observations hold with respect to the articles and the journals they appeared in.

Figure 16.1(b) clearly indicates exponential growth for all four document classes and we observe that the growth rate of `JOURNAL` and `ARTICLE` documents de-

\(^4\)Note that, strictly speaking, our data is not normally distributed, due to the left limit at position $x = 1$. Still, the Gaussian curve shown in Fig. 16.1(a) nicely fits the data.
creases in the final years. This observation strongly suggests a limited growth scenario, which is captured by limited growth curves, i.e., functions of the form

\[ f_{\text{logistic}}(x) = \frac{a}{1 + be^{-cx}} \]  

(16.2)

where \( a, b, c \in \mathbb{R}_{>0} \). For this parameter setting, \( a \) constitutes the upper asymptote and the \( x \)-axis forms the lower asymptote; the curve is “caught” in-between its asymptotes and increases continuously, i.e. it is \( S \)-shaped. Fitting this function type, the developments of \textit{JOURNAL}, \textit{PROCEEDINGS}, \textit{INPROCEEDINGS} and \textit{ARTICLE} documents over time are modeled as instances of (16.2), by fixing parameters \( a, b, \) and \( c \), appropriately. The resulting approximations are plotted in Fig. 16.1(b).

A similar analysis leads to approximation functions for the document classes \textit{BOOK}, \textit{INCOLLECTION}. As for \textit{PHDTHESIS}, \textit{MASTERS THESIS}, and \textit{WWW} documents we found that they were distributed unsteadily; they are implemented in \textsc{SP2Bench} as random functions that reflect their distribution in the original data set.

16.3.1.3 Other Characteristics

In the following, we shortly sketch selected other characteristics that were considered in the \textsc{SP2Bench} data generator design. A complete discussion can be found in [28].

- **Authors**: special care in the analysis of DBLP was spent in the analysis of authors. The study includes the total number of authors per year (i.e., the sum of \textit{author} attributes), the number of distinct authors per year (obtained from the total authors by eliminating duplicates), as well as the number of new authors in each year. Among others, this analysis shows that the total number of authors has increased steadily over time [accounting for the increasing number of publications over time sketched in Fig. 16.1(b)] and that the number of distinct authors relative to the number of total authors decreases over time, which reflects an increasing productivity of authors, as discussed before in Sect. 16.3.1.1.

- **Editors**: as one might expect, an analysis of connections between authors and editors in DBLP reveals that editors are typically persons that have published before, i.e., persons that are well known in the community.

- **Publications**: the distribution of publications among authors follows a prototypical power law distribution: there are only few authors with many publications, whereas many authors have few publications. Power law distributions are natural distributions that can be modeled as power equations (cf. [28]).

- **Incoming Citations**: in addition to the study of outgoing citations discussed in Sect. 16.3.1.1 [Fig. 16.1(a)], we consider the number of incoming citations for publications. Like for the distribution of publications among authors, we observe a power law distributions, i.e., few publications have many incoming citations, but many of them have few citations. Consequently, we use power equations to implement this distribution in the \textsc{SP2Bench} data generator.
• **Completeness of the Citation System**: it is worth mentioning that, according to Table 16.1, only 0.5% of all article and only about 1% of inproceedings have outgoing citations. Arguably, this value should be close to 100% in a complete scenario, i.e., DBLP contains only a fraction of all existing citations. In addition, we found that in DBLP the number of incoming citations is considerably smaller than the number of outgoing citations. At first glance, this might seem paradoxically, but it is simply explained by the fact that DBLP contains many untargeted citations. Combined with the previous observation that only a fraction of all publications have outgoing citations, we conclude that the DBLP citation system is very incomplete, although in some sense natural in that it follows natural distributions such as power law distribution (w.r.t. incoming citations) or the Gaussian distribution (w.r.t. outgoing references, cf. Sect. 16.3.1.1).

### 16.3.2 Data Generator Implementation and RDF Scheme

All the characteristics discussed throughout Sect. 16.3.1 are implemented in the SP²Bench data generator. The implementation is written in C++ and offers two parameters, to fix either a triple count limit or the year up to which data will be generated. The generation process is simulation-based, which, among others, means that we assign life times to authors and individually estimate their future behavior, taking into account global publication and coauthor constraints and characteristics, as well as the number of distinct and new authors per year (cf. Sect. 16.3.1.3).

All random functions (which, for example, are used to assign the attributes according to Table 16.1, or to sample data according to the approximation functions in Fig. 16.1) base on a fixed seed. This makes data generation deterministic, i.e., the parameter setting uniquely identifies the outcome. In addition, the generator is implemented in plain ANSI C++, which asserts platform-independence. The fixed random seed and the platform-independent implementation ensure that documents generated under different platforms are always identical, so experimental results from different platforms remain (at least to a certain degree) comparable.

We next survey selected data generator and output document characteristics for documents containing up to 25M RDF triples that have been generated with the SP²Bench data generator implementation. Table 16.2 lists the size of the output file, the year up to which data was generated, the counts of the document class instances (cf. Sect. 16.3.1.2), and the number of total authors and distinct authors contained in the data set (cf. Sect. 16.3.1.3). One can observe superlinear growth for the number of authors relative to the number of triples in the data set, which is primarily caused by the increasing average number of authors per publication, as discussed in Sect. 16.3.1.1. The growth rate of proceedings and inproceedings is also superlinear, while the number of journals and articles increases sublinear. These observations reflect the development of proceedings, inproceedings, journals, and articles sketched in Fig. 16.1(b). Note that the number of inproceedings and articles in the data set clearly dominates the remaining document classes. Finally,
Table 16.2 Characteristics of generated documents

<table>
<thead>
<tr>
<th>#Triples</th>
<th>10k</th>
<th>50k</th>
<th>250k</th>
<th>1M</th>
<th>5M</th>
<th>25M</th>
</tr>
</thead>
<tbody>
<tr>
<td>file size [MB]</td>
<td>1.05</td>
<td>2.6</td>
<td>106</td>
<td>533</td>
<td>2694</td>
<td></td>
</tr>
<tr>
<td>#Tot.Auth.</td>
<td>1.5k</td>
<td>6.8k</td>
<td>34.5k</td>
<td>151.0k</td>
<td>898.0k</td>
<td>5.4M</td>
</tr>
<tr>
<td>#Dist.Auth.</td>
<td>0.9k</td>
<td>4.1k</td>
<td>20.0k</td>
<td>82.1k</td>
<td>429.6k</td>
<td>2.1M</td>
</tr>
<tr>
<td>#Journals</td>
<td>25</td>
<td>104</td>
<td>439</td>
<td>1.4k</td>
<td>4.6k</td>
<td>11.7k</td>
</tr>
<tr>
<td>#Articles</td>
<td>916</td>
<td>4.0k</td>
<td>17.1k</td>
<td>56.9k</td>
<td>207.8k</td>
<td>642.8k</td>
</tr>
<tr>
<td>#Proc.</td>
<td>6</td>
<td>37</td>
<td>213</td>
<td>903</td>
<td>4.7k</td>
<td>24.4k</td>
</tr>
<tr>
<td>#Inproc.</td>
<td>169</td>
<td>1.4k</td>
<td>9.2k</td>
<td>43.5k</td>
<td>255.2k</td>
<td>1.5M</td>
</tr>
<tr>
<td>#Incoll.</td>
<td>18</td>
<td>56</td>
<td>173</td>
<td>442</td>
<td>1.4k</td>
<td>4.5k</td>
</tr>
<tr>
<td>#Books</td>
<td>0</td>
<td>0</td>
<td>39</td>
<td>356</td>
<td>973</td>
<td>1.7k</td>
</tr>
<tr>
<td>#PhD Th.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>101</td>
<td>237</td>
<td>365</td>
</tr>
<tr>
<td>#Mast.Th.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>95</td>
<td>169</td>
</tr>
<tr>
<td>#WWWs</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>35</td>
<td>92</td>
<td>168</td>
</tr>
</tbody>
</table>

we remark that—like in the original DBLP database—in the early years instances of some document classes are missing, e.g., there are no BOOK or WWW documents for these years.

The SP$^2$Bench RDF Scheme for DBLP  Having established the basic knowledge of structural aspects of DBLP, we are now ready to describe the RDF scheme for the DBLP data set that is implemented in the SP$^2$Bench data generator. The scheme basically follows the approach in [5], an existing XML-to-RDF mapping of the original DBLP database. However, with the goal to generate arbitrarily-sized documents, the SP$^2$Bench data generator uses lists of first and last names, publishers, and random words, rather than real author, publications and conference names. The generated SP$^2$Bench conference and journal names are always strings of the form “Conference $i$ ($year$)” and “Journal $i$ ($year$)”, where $i$ is a unique conference (respectively journal) number in year $year$. Furthermore, predefined lists of random words are used to generate string content, such as titles or abstracts. Concerning person names, the generator relies on lists of first and last names to create fresh random names. Finally, domain-specific string-content, such as the pages specification or the ISBN number of documents are filled with random strings of a reasonable domain (for instance, ISBN-like random strings).

Following [5], SP$^2$Bench uses existing RDF vocabularies to describe resources in a uniform way. It borrows vocabulary from FOAF (namespace foaf) for describing persons, and from SWRC (namespace swrc) and Dublin Core (namespace dc) for
Fig. 16.2 Sample RDF database that illustrates the structure of generated SP²Bench data describing scientific resources.⁵ Additionally, the fresh namespace bench defines DBLP-specific document classes, such as bench:Book and bench:WWW.

Unfortunately, the XML-to-RDF translation of DBLP presented in [5] neither contains blank nodes nor RDF containers. With the goal to contribute a comprehensive benchmark, though, SP²Bench requires such RDF-specific constructs. For this reason, persons in the data set are modeled as (unique) blank nodes “_:firstname_lastname” (using the lists of first- and lastnames mentioned above), instead of URIs. Moreover, to have RDF containers available, the SP²Bench data generator models outgoing citations of documents using standard rdf:Bag containers. As a further modification, we enrich a small fraction of ARTICLE and INPROCEEDINGS documents with the property bench:abstract, which carries comparably large string content (the original DBLP data set does not contain abstracts).

Figure 16.2 illustrates a sample DBLP instance in RDF format, as it might be generated by the SP²Bench data generator. Dashed edges are typing edges (i.e., rdf:type) and sc stands as an abbreviation for rdfs:subClassOf. On the logical level, we distinguish between the schema layer (gray) and the instance layer (white). As discussed before, reference lists are modeled as blank nodes of type rdf:Bag (see e.g., node _:references1), while both authors and editors are modeled as blank nodes of type foaf:Person. On the schema level, class foaf:Document splits up into the DBLP document classes bench:Journal, bench:Article, and so on. In summary, the sample graph defines three per-

sons, one proceeding, two inproceedings, one journal and one article. For readability reasons, we plot only selected properties of these entities. As also illustrated, property \texttt{dcterms:partOf} links inproceedings and proceedings together, while \texttt{swrc:journal} connects articles to the journals they appeared in.

In order to provide an entry point for queries that access authors and to provide a person with fixed characteristics, the generated data contains a special author, named after the famous mathematician Paul Erdős (it is not shown in Fig. 16.2). Per year, the generator assigns exactly ten publications and two editor activities to this prominent person, starting from year 1940 up to 1996. For the ease of access, Paul Erdős is modeled as a fixed URI. As an example query that uses this person, consider \textit{Q8} in Sect. 16.4.3, which extracts all persons with \textit{Erdős Number}\textsuperscript{6} 1 or 2.

### 16.4 The SP\textsuperscript{2}Bench Benchmark Queries

We now come to the discussion of the SP\textsuperscript{2}Bench queries. Before presenting the queries one by one in Sect. 16.4.3, we survey characteristics of RDF (Sect. 16.4.1) and SPARQL (Sect. 16.4.2) that were of particular interest for query design.

#### 16.4.1 RDF Characteristics

Decisions on how RDF data is stored and accessed by SPARQL engines might heavily influence the performance of the engine (see for instance the discussions in [1, 26, 36]). Consequently, any reasonable SPARQL benchmark should consider the specifics of the underlying RDF data representation language.

The first aspect that is interesting with respect to the RDF data format is that RDF constitutes elements from three different domains, namely URIs, blank nodes, and literals. SPARQL engines might represent elements from these domains differently (e.g., it might make sense to have a special index for literals to accelerate text search). The SP\textsuperscript{2}Bench queries therefore access all three entities and different combinations thereof; line 1 in Table 16.3 surveys this characteristic for the SP\textsuperscript{2}Bench queries, where abbreviations are indicated by bold font, e.g., using \texttt{B} as a shortcut for \texttt{BLANK NODES}.

Line 1 also indicates queries that access comparably large literals (namely the abstracts of documents) and RDF containers (i.e., the outgoing references, which are of type \texttt{rdf:Bag}). Containers are of particular interest due to their special semantics and the fact that they induce a (possibly large) set of membership properties \texttt{rdf:}_1, \texttt{rdf:}_2, \texttt{rdf:}_3, \ldots. As argued in [26], this may induce problems for some RDF storage schemes like Vertical Partitioning [1].

---

\textsuperscript{6}See http://www.oakland.edu/enp/.

\textsuperscript{7}As we shall see in Sect. 16.4.3, \textit{Q12a} and \textit{Q12b} are ASK-counterparts of the SELECT queries \textit{Q5a} and \textit{Q8}, respectively. The ASK versions of these queries are not explicitly listed in the table.
Table 16.3  Selected properties of the SP2Bench queries

<table>
<thead>
<tr>
<th>Query</th>
<th>1</th>
<th>2</th>
<th>3abc</th>
<th>4</th>
<th>5ab</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Access Pattern: <strong>SUBJ., PRED., OBJ., NONE</strong></td>
<td>P, PO</td>
<td>P, PO</td>
<td>N, PO</td>
<td>P, PO</td>
<td>P, PO</td>
<td>P, PO</td>
</tr>
<tr>
<td>3 Operators: <strong>AND, FILTER, UNION, OPTIONAL</strong></td>
<td>A</td>
<td>A, O</td>
<td>A, F</td>
<td>A, F</td>
<td>A, F</td>
<td>A, F, O</td>
</tr>
<tr>
<td>4 Modifiers: <strong>DISTINCT, LIMIT, OFFSET, ORDER by</strong></td>
<td>–</td>
<td>Ob</td>
<td>–</td>
<td>D</td>
<td>D</td>
<td>–</td>
</tr>
<tr>
<td>5 Filter Pushing Possible?</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>✓/–</td>
<td>✓</td>
</tr>
<tr>
<td>6 Reusing of Graph Patterns Possible?</td>
<td>–</td>
<td>–</td>
<td>✓</td>
<td>–</td>
<td>–</td>
<td>✓</td>
</tr>
</tbody>
</table>

A second important requirement in the context of RDF data is to test different data access paths at triple pattern level. Triple patterns in SPARQL queries may contain variables in any position and the efficient evaluation of single triple patterns forms the basis for the fast evaluation of more complex queries. On the one hand, it seems reasonable to assume that in most triple patterns the predicate is fixed, such as is the case in patterns like 

\[(?book, dc:creator, ?name)\]

or

\[(bench:Book1, dc:creator, ?name)\]

asking for authors of all or a fixed book, respectively. Such patterns are natural counterparts of SQL queries, which select a fixed set of properties from entities, by accessing exactly those table attributes that contain the desired properties. On the other hand, as also argued in [29, 36], one strength of RDF querying is the flexibility of having variables in predicate position, which allows for patterns like

\[(Book1, ?prop, ?val)\]

or

\[(?subj, ?prop, Person1)\]

to extract all entities that stand in direct relation to Person1. We survey the use of data access patterns in the SP2Bench queries in Table 16.3, Line 2, where e.g. the shortcut PO denotes that the query contains triple patterns with fixed predicate and
object position and a variable in subject position. While, in most triple patterns the predicate position is fixed, the SP²Bench queries cover also other patterns, such as triple patterns containing only variables ($Q^{3a}$, $Q^{3b}$, $Q^{3c}$, and $Q^9$), patterns where only the object is bound ($Q^{10}$), and variable-free triple patterns ($Q^{12c}$).

### 16.4.2 SPARQL Characteristics

Next, we turn towards a discussion of SPARQL characteristics that were of particular interest in query design. Rows 3 and 4 in Table 16.3 survey the operators and solution modifiers that are used in the benchmark queries. As can be seen, the queries cover various operator constellations, combined with selected solution modifier combinations. Special care in query design was taken in operator OPTIONAL, arguably the most complex operator in the SPARQL query language [23, 27], which can be used to encode (closed-world) negation in SPARQL [3] (cf. $Q^6$ and $Q^7$).

Another important objective of SP²Bench was to design queries that are amenable to a wide range of SPARQL optimization approaches. One promising approach to SPARQL optimization is the reordering of triple patterns based on selectivity estimation [22, 30], similar by idea to join reordering in Relational Algebra optimization. A beneficial ordering of triple patterns depends on both the selectivity of triple patterns and data access paths that are provided by the engine. Actually, most SP²Bench queries might benefit from such an optimization, because most of them contain large AND-connected blocks and require series of joins. Closely related to triple reordering is filter pushing, which aims at an early evaluation of filter conditions (see e.g., [27]). Like triple pattern reordering, filter pushing might speed up the evaluation by decreasing the size of intermediate results. Row 5 in Table 16.3 identifies the SP²Bench queries that are amenable to filter pushing techniques.

Another reasonable idea is to reuse evaluation results of graph pattern evaluation (or even of whole subqueries). This strategy is applicable whenever the same pattern or subquery is used multiple times in the same query. As a simple example, consider $Q^4$ in Sect. 16.4.3. In that query, $?article1$ and $?article2$ in the first and second triple pattern will be bound to exactly the same nodes of the input RDF graph, so it suffices to evaluate this pattern only once and use this result for both triple patterns. We survey the applicability of graph pattern reusing in Table 16.3, row 6. We will sketch more optimization approaches, such as Semantic Query Optimization (cf. [27]), when discussing the individual queries in the following section.

### 16.4.3 Discussion of Benchmark Queries

We now come to a one by one presentation of the SP²Bench queries, covering the challenges and the characteristics of the individual queries. In the following discussion, we distinguish between in-memory engines, which load the document from file and process queries in main memory, and native engines, which rely on a physical
Table 16.4 Number of query results for SELECT queries on documents up to 25 million RDF triples

<table>
<thead>
<tr>
<th></th>
<th>Q1</th>
<th>Q2</th>
<th>Q3a</th>
<th>Q3b</th>
<th>Q3c</th>
<th>Q4</th>
<th>Q5a</th>
<th>Q5b</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
<th>Q11</th>
</tr>
</thead>
<tbody>
<tr>
<td>10k</td>
<td>1</td>
<td>147</td>
<td>846</td>
<td>9</td>
<td>0</td>
<td>23226</td>
<td>155</td>
<td>155</td>
<td>229</td>
<td>184</td>
<td>4</td>
<td>166</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>50k</td>
<td>1</td>
<td>965</td>
<td>3647</td>
<td>25</td>
<td>0</td>
<td>104746</td>
<td>1085</td>
<td>1085</td>
<td>1769</td>
<td>2</td>
<td>264</td>
<td>4</td>
<td>307</td>
<td>10</td>
</tr>
<tr>
<td>250k</td>
<td>1</td>
<td>6197</td>
<td>15853</td>
<td>127</td>
<td>0</td>
<td>542801</td>
<td>6904</td>
<td>6904</td>
<td>12093</td>
<td>62</td>
<td>332</td>
<td>4</td>
<td>452</td>
<td>10</td>
</tr>
<tr>
<td>1M</td>
<td>1</td>
<td>32770</td>
<td>52676</td>
<td>379</td>
<td>0</td>
<td>2586733</td>
<td>35241</td>
<td>35241</td>
<td>62795</td>
<td>292</td>
<td>400</td>
<td>4</td>
<td>572</td>
<td>10</td>
</tr>
<tr>
<td>5M</td>
<td>1</td>
<td>248738</td>
<td>192373</td>
<td>1317</td>
<td>0</td>
<td>18362955</td>
<td>210662</td>
<td>210662</td>
<td>417625</td>
<td>1200</td>
<td>493</td>
<td>4</td>
<td>656</td>
<td>10</td>
</tr>
<tr>
<td>25M</td>
<td>1</td>
<td>1876999</td>
<td>594890</td>
<td>4075</td>
<td>0</td>
<td>n/a</td>
<td>696681</td>
<td>696681</td>
<td>1945167</td>
<td>5099</td>
<td>493</td>
<td>4</td>
<td>656</td>
<td>10</td>
</tr>
</tbody>
</table>

database system. When discussing the challenges for native engines, we always assume that the document has been loaded into the database prior to query processing.

Before starting our discussion, we survey the result sizes of the individual queries on RDF documents of different size in Table 16.4. The overview shows that the queries vary in their result size (e.g., we have queries with increasing, constant and empty result size). The survey forms the basis for the subsequent discussion.

**Benchmark Query Q1:** Return the year of publication of “Journal 1 (1940)”.

```sparql
SELECT ?yr
WHERE {
  ?journal dcterms:issued ?yr
}
```

Benchmark query Q1 returns exactly one result (for arbitrarily large documents). Native engines might use index lookups to answer this query in (almost) constant time, i.e., execution time should be independent from document size. In-memory engines must scan the whole document and should scale linearly to document size.

**Benchmark Query Q2:** Extract all inproceedings with properties dc:creator, bench:booktitle, dcterms:issued, dcterms:partOf, rdfs:seeAlso, dc:title, swrc:pages, foaf:homepage, and optionally bench:abstract, including the respective values.

```sparql
WHERE {
  ?inproc rdfs:seeAlso ?ee.
  ?inproc dcterms:issued ?yr
  OPTIONAL { ?inproc bench:abstract ?abstract }
} ORDER BY ?yr
```
This query implements a large star-join pattern, where different properties of inproceedings (variable ?inproc) are requested. It contains a simple OPTIONAL clause, and accesses large strings (i.e., the abstracts). Result size grows with database size, and a final result ordering is necessary due to operator ORDER BY. Both native and in-memory engines might reach evaluation times almost linear to the document size.

**Benchmark Queries Q3abc:** *Select all articles with property (a) `swrc:pages`, (b) `swrc:month`, or (c) `swrc:isbn`.*

<table>
<thead>
<tr>
<th>Query</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) SELECT ?article WHERE { ?article rdf:type bench:Article. ?article ?property ?value FILTER (?property=swrc:pages) }</td>
<td>(a) Q3a, but &quot;swrc:month&quot; instead of &quot;swrc:pages&quot;</td>
</tr>
<tr>
<td>(b) Q3a, but &quot;swrc:month&quot; instead of &quot;swrc:pages&quot;</td>
<td>(c) Q3a, but &quot;swrc:isbn&quot; instead of &quot;swrc:pages&quot;</td>
</tr>
</tbody>
</table>

These three queries test FILTER expressions with varying selectivity. According to Table 16.1, the FILTER expression in Q3a is not very selective (i.e., retains about 92.61% of all articles). Data access through a secondary index for Q3a is probably not very efficient, but might work well for Q3b, which selects only 0.65% of all articles. The filter condition in Q3c is never satisfied, because no articles have `swrc:isbn` predicates. Native engines might use statistics to answer Q3c in constant time.

**Benchmark Query Q4:** *Select all distinct pairs of article author names for authors that have published in the same journal.*

<table>
<thead>
<tr>
<th>Query</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT DISTINCT ?name1 ?name2 WHERE { ?article1 rdf:type bench:Article. ?article2 rdf:type bench:Article. ?article1 dc:creator ?author1. ?author1 foaf:name ?name1. ?article2 dc:creator ?author2. ?author2 foaf:name ?name2. ?article1 swrc:journal ?journal. ?article2 swrc:journal ?journal FILTER (?name1&lt;?name2) }</td>
<td>Q4 contains a rather long graph chain, i.e. variables ?name1, and ?name2 are linked through the articles that different authors have published in the same journal. As one might expect, the result is very large (cf. Table 16.4). Instead of evaluating the outer pattern block and applying the FILTER afterwards, engines might embed the FILTER expression in the computation of the block, e.g., by exploiting indices on author names. The DISTINCT modifier further complicates the query. We shall expect superlinear behavior for this query, even for native engines.</td>
</tr>
</tbody>
</table>
Benchmark Queries Q5ab: Return the names of all persons that occur as author of at least one inproceeding and at least one article.

(a) SELECT DISTINCT ?person ?name
    WHERE { ?article rdf:type bench:Article.
              ?article dc:creator ?person.
              ?person foaf:name ?name.
              FILTER(?name=?name2) }

(b) SELECT DISTINCT ?person ?name
    WHERE { ?article rdf:type bench:Article.
              ?article dc:creator ?person.
              ?person foaf:name ?name }

Q5a and Q5b test different join variants: Q5a implements an implicit join on author names (encoded in the FILTER condition), while Q5b explicitly joins the authors on variable ?name. Although in general the queries are not equivalent, the one-to-one mapping between authors and their names (i.e., author names constitute primary keys) in the SP2Bench scenario implies equivalence. In [18, 27], semantic optimization using such keys for RDF has been proposed. Such approaches might detect the equivalence of Q5a and Q5b in and choose the more efficient variant.

Benchmark Query Q6: Return, for each year, the set of all publications authored by persons that have not published in years before.

SELECT ?yr ?name ?doc
WHERE { ?class rdfs:subClassOf foaf:Document.
              ?author foaf:name ?name.
              OPTIONAL { ?class2 rdfs:subClassOf foaf:Document.
              FILTER (!bound(?author2))
              FILTER (?author2 && ?yr2<?yr) } }

This query implements closed world negation (CWN), expressed through a combination of operators OPTIONAL, FILTER, and BOUND. The idea of the construction is that the block outside the OPTIONAL expression computes all publications, while the inner one constitutes earlier publications from authors that appear outside. The outer FILTER expression then retains publications for which ?author2 is unbound, i.e., those publications of authors without publications in earlier years. For this query, SPARQL-specific optimization like the algebraic rewriting of closed-world negation proposed in [27], might be beneficial.
**Benchmark Query Q7**: Return the titles of all publications that have been cited at least once, but not by any paper that has not been cited itself.

```
SELECT DISTINCT ?title
WHERE {
  ?class rdfs:subClassOf foaf:Document.
OPTIONAL {
  ?class3 rdfs:subClassOf foaf:Document.
OPTIONAL {
  ?class4 rdfs:subClassOf foaf:Document.
} FILTER (!bound(?doc4))
} FILTER (!bound(?doc3))
}
```

This query tests double negation, which requires the encoding of nested closed-world negation. Recalling that the citation system of DBLP is rather incomplete (cf. Sect. 16.3.1.3), this query returns only few results (cf. Table 16.4). However, the query is challenging due to the double negation. Engines might reuse graph pattern evaluation results. For instance, the AND-connected triple pattern block $1$ rdfs:subClassOf foaf:Document. $2$ rdf:type $1$ occurs three times, for (i) $1:=?class, \$2:=?doc$, (ii) $1:=?class3, \$2:=?doc3$, and (iii) $1:=?class4, \$2:=?doc4$.

**Benchmark Query Q8**: Compute authors that have published with Paul Erdös or with an author that has published with Paul Erdös.

```
SELECT DISTINCT ?name
WHERE {
  ?erdoes foaf:name "Paul Erdös"^^xsd:string.
  ?author2 foaf:name ?name
FILTER (?author!=?erdoes &&
  ?doc2!=?doc &&
  ?author2!=?erdoes &&
  ?author2!=?author)
) UNION {
  ?author foaf:name ?name
FILTER (?author!=?erdoes) }
}
```
Here, the evaluation of the second \texttt{UNION} part is basically “contained” in the evaluation of the first part. Hence, techniques like graph pattern (or subexpression) reusing are applicable. Another promising strategy is to decompose the filter expressions and push its components down in the operator tree (cf. \cite{27}), in order to apply filter subexpressions early and to decrease the size of intermediate results.

**Benchmark Query Q9:** Return incoming and outgoing properties of persons.

```sql
1  SELECT DISTINCT ?predicate
2  WHERE {
3    ( ?person rdf:type foaf:Person.
4     ?subject ?predicate ?person ) UNION
6     ?person ?predicate ?object )}
```

Q9 has been primarily designed to test nonstandard data access patterns. Naive implementations would compute the triple patterns of the \texttt{UNION} subexpressions separately, thus evaluating patterns where no component is bound. Then, pattern \texttt{?subject ?predicate ?person} selects all graph triples, which is rather inefficient. Another idea is to evaluate the first triple in each \texttt{UNION} subexpression, afterwards using the bindings for variable \texttt{?person} to evaluate the second pattern efficiently. Note that the result size is exactly 4 for sufficiently large documents (see Table 16.4). RDF-specific statistics about incoming and outgoing properties of Person-typed objects (in native engines) might help to answer this query in constant time, even without data access. In-memory engines, however, must always load the whole document and therefore should scale linearly to document size.

**Benchmark Query Q10:** Return all subjects that stand in some relation to person “Paul Erdös”, including the type of their relation.

```sql
1  SELECT ?subj ?pred
2  WHERE { ?subj ?pred person:Paul_Erdoes }
```

In the SP\textsuperscript{2}Bench scenario, this query can be reformulated as: Return publications and venues in which “Paul Erdös” is involved as author or as editor. It implements an object bound-only access pattern. In contrast to Q9, statistics are not immediately useful, because the result includes the subjects (which must be extracted from the data set). Recalling that “Paul Erdös” is active only between 1940 and 1996 (cf. the discussion in Sect. 16.3.2), the result size stabilizes for large documents. Native engines that exploit indices might reach (almost) constant execution time.

**Benchmark Query Q11:** Return (up to) 10 electronic edition URLs starting from the 51th publication, in lexicographical order.
The focus of this query lies on the combination of solution modifiers ORDER BY, LIMIT and OFFSET. In-memory engines have to read, process and sort electronic editions prior to application of the LIMIT and OFFSET modifiers. In contrast, native engines might exploit indices to access only a fraction of all electronic editions and, as the result size is limited to 10 due to the LIMIT modifier, would optimally reach constant runtime, independent from the size of the input document.

**Benchmark Query Q12**: (a) Return yes if a person occurs as author of at least one inproceeding and article, no otherwise; (b) Return yes if an author has published with Paul Erdős or with an author that has published with “Paul Erdős”, and no otherwise; (c) Return yes if person “John Q. Public” is present in the database.

All three queries are boolean queries, designed to test the efficient implementation of the SPARQL ASK query form. Q12a and Q12b share the properties of their SELECT counterparts Q5a and Q8, respectively. They always return yes for sufficiently large documents. When evaluating ASK queries, engines should stop the evaluation process as soon as a solution has been found. A reasonable optimization approach would be to adapt the query execution plan, trying to efficiently locate a witness. For instance, based on execution time estimations it might be favorable to evaluate the (simpler) second part of the UNION in Q12b first. Q12c asks for a single triple that is not present in the database. With indices, native engines might answer Q12c in constant time. Again, in-memory engines must read and scan the whole document.

### 16.5 Benchmark Metrics

In this section, we propose several benchmark metrics that cover different aspects of the evaluation process. These metrics reflect the scope and design decisions of SP2Bench and can be used to systematically evaluate benchmark results.

We propose to perform three runs over documents comprising 10k, 50k, 250k, 1M, 5M and 25M RDF triples, using a fixed timeout of 30 min per query and document. The reported time should include the average value over all three runs and, if significant, the errors within these runs. This setting was tested in [26, 28] and,
for state-of-the-art engines, can be evaluated in reasonable time (typically within few days). If the tested engine is fast enough, nothing prevents the user from adding larger documents. In the following, we shortly describe a set of interesting metrics.

1. **SUCCESS RATE**: As a first indicator, we propose to survey the success rates for the engine on top of all document sizes, distinguishing between Success, Timeout (e.g., an execution time > 30 min), Memory Exhaustion (if an additional memory limit was set) and general Errors. This metric gives a good survey over scaling properties and might give first insights into the engine’s overall behavior.

2. **LOADING TIME**: The loading time for documents of different sizes is interesting to get insights into the efficiency and, particularly, to see how document loading scales with document size. This metric primarily applies to engines with a physical database backend and might be irrelevant for in-memory engines, where document loading is usually part of the query evaluation process.

3. **PER-QUERY PERFORMANCE**: Individual performance results for all queries over all document sizes give more detailed insights into the behavior than the SUCCESS RATE metric discussed before. The PER-QUERY PERFORMANCE metric forms the basis for a deep study of the results and allows to investigate strengths, weaknesses and scaling of the tested implementation based on a one by one discussion of the engine result for the individual benchmark queries.

4. **GLOBAL PERFORMANCE**: This metric integrates the individual per-query results into a global performance measure. It contains, for each tested document size, the arithmetic and the geometric mean of the engine’s average execution time over all queries. We propose to penalize timeouts and other errors with 3600 s (i.e., twice the evaluation time limit). GLOBAL PERFORMANCE is well-suited to investigate the overall scaling properties of engines and to compare the performance of different evaluation approaches. Note that the arithmetic mean imposes a stronger penalty on outlier queries than the geometric mean. Generally speaking, the geometric mean better reflects the average behavior, while the arithmetic mean gives good insights in the worst-case behavior.

5. **MEMORY CONSUMPTION**: In particular, for engines with a physical backend, the main memory consumption for the individual queries and also the average memory consumption over all queries might be of interest. Optimally, physical backend database engines should get by with a constant main memory consumption, independent from the size of the input document.

We refer the interested reader to [26, 28] for example discussions of SP²Bench evaluation results based on the metrics presented above.

### 16.6 Conclusion

In this chapter, we presented the SP²Bench performance benchmark for SPARQL, which constitutes a methodical approach for testing the performance of engines

---

8The geometric mean is defined as the \( n \)th root of the product over \( n \) numbers.
w.r.t. different operator constellations, RDF access paths, typical RDF constructs and a variety of possible optimization approaches. SP²Bench allows to assess the performance of SPARQL engines in a general, comprehensive way and to identify weaknesses and strengths of evaluation approaches. In this line, the SP²Bench framework is a useful tool for both industrial and research quality assessment.

The SP²Bench data generator supports the creation of arbitrarily large DBLP-style documents, which implement various real-world distributions, accounting for the social-world character of the Semantic Web. While, in the context of SP²Bench, this data forms the basis for the design of challenging and predictable benchmark queries, the data generator might also be useful in other Semantic Web projects, i.e., whenever large amounts of test data with natural distributions are needed.

References

34. W3C: Resource Description Framework (RDF). http://www.w3.org/RDF/