Benchmark for Performance Evaluation of SHACL Implementations in Graph Databases

Robert Schaffenrath, Daniel Proksch, Markus Kopp, Iacopo Albasini, Oleksandra Panasiuk, and Anna Fensel

Department of Computer Science, University of Innsbruck, Austria
robert.schaffenrath@student.uibk.ac.at,
daniel.proksch@student.uibk.ac.at, markus.kopp@student.uibk.ac.at,
iacopo.albasini@student.uibk.ac.at,
oleksandra.panasiuk@sti2.at, anna.fensel@sti2.at

Abstract. Due to the rise in the commercial usage of knowledge graphs, the validation of graph-based data has gained importance over the past few years in the field of Semantic Web. In spite of this trend, the number of graph databases that support W3C's validation specification Shapes Constraint Language (SHACL) can still be regarded as low, and best practices for their SHACL implementations performance evaluation are lacking. In this paper, we propose a benchmark for performance evaluation of SHACL implementations and present an evaluation of five common graph databases using the benchmark.

Keywords: Benchmark, SHACL, Graph Database

1 Introduction

The Semantic Web is a concept for an extension of the World Wide Web that was originally popularized through a 2001 article in the Scientific American by Berners-Lee, Hendler and Lassila [3]. The goal of the concept was to provide a new form of web content focusing on machine interpretability. This would enable the usage of autonomous task handling by software agents and by that enhance the user experience on the web.

Since the initial introduction of the concept, many developments have taken place. There has been a substantial increase in the amount of publicly available linked data on the web. Companies like Google and Facebook started to use semantic data structures and knowledge graphs to improve their services for the user [23] [26]. The establishment of intelligent assistants, like Cortana by Microsoft or Siri by Apple that can act comparably to the in 2001 envisioned software agents to autonomously solve tasks basing on the semantic annotations [3], can be seen as a sign of the increasing importance of the Semantic Web.

With the increase in utilization of graph-based data on the web and the issues with the published data quality [13], the need for validation of this type of data has increased as well. To accommodate this need, the World Wide Web

Consortium (W3C) has provided a standardized Shapes Constraint Language (SHACL) [15] in 2017. The W3C recommendation represents a language for the validation of RDF graphs against a set of conditions. Moreover, with the increased velocity needs of applications, time is becoming an important factor for the data processing with SHACL.

In spite of the need for the type of validations SHACL provides, most popular graph databases currently do not offer an implementation of the standard or a different validation language. Depending on the size of the data graph and shapes graph, validation can entail substantial issues in terms of the memory consumption versus computation time. There are currently no benchmarks for SHACL validations in graph databases, which warrant an empirical evaluation of the available implementations. In this paper, we provide a SHACL performance evaluation benchmark and conduct the performance evaluation of SHACL implementations of different graph databases employing it.

This paper is structured as follows. Section 2 presents related work, which focuses on benchmarking of graph databases. Section 3 describes the methodology and construction of our benchmark. Section 4 presents the performance evaluation of five common graph databases we conducted using the benchmark. A detailed explanation for the results is provided in Section 5. Section 6 concludes the paper.

2 Related Work

There are multiple available evaluations of graph databases for various benchmark and instruction types. Jouili and Vansteenberghe [12] performed an empirical comparison of the graph databases Neo4j, Titan, OrientDB and DEX using the specifically developed benchmarking framework GDB. The framework focuses on benchmarking typical graph operations. The database management system Neo4j generally yielded the best result for the performed instructions.

McColl et al. [16] conducted a performance comparison of 12 different graph databases using the four fundamental graph algorithms SSSP, Connected Components, PageRank and Update Rate. The evaluation was performed on a network containing up to 256 million edges.

Dominguez-Sal et al. [6] evaluated the performance of the graph databases Neo4j, Apache Jena, HypergraphDB and DEX using the HPC Scalable Graph Analysis Benchmark. From the results, they concluded that DEX and Neo4j were the most efficient of the four tested graph databases.

Capotă et al. [5] developed a big data benchmark for graph-processing platforms. Using this benchmark, they evaluated the platforms Apache Giraph, MapReduce, GraphX and Neo4j for five different algorithms on three different datasets.

While these evaluations focused on ordinary graph operations, multiple evaluations based on benchmarks for SPARQL queries were conducted as well. Bizer and Schulz [4] developed the Berlin SPARQL Benchmark, which focuses on search and navigation patterns in a fictional use-case. Using this benchmark,

they conducted a comparative evaluation of the RDF stores Sesame, Virtuoso, Jena TDB and Jena SDB to compare them to the performance of the SPARQL-to-SQL writers D2R Server and Virtuoso RDF.

Schmidt et al. [21] created another SPARQL performance benchmark based on generated data using the scenario of the DBLP dataset. Compared to the Berlin Benchmark, SP2Bench is less use-case driven and tries to address language-specific issues. The benchmark was conducted for the RDF stores Virtuoso, Sesame, Redland, ARQ and SDB.

The presented papers portray the availability of evaluations and benchmarks for multiple aspects of graph databases. These aspects however do not include SHACL validations at the current moment, which suggests the need for more research in this specific area.

3 The Benchmark

We propose a benchmark for performance evaluation of SHACL implementations in graph databases. The benchmark focuses on evaluating the performance of full SHACL validations on a dataset using the implementations and storages provided by the graph database management system. Using the benchmark, a comparative analysis of five common graph databases is performed. The validity of the validation processes is not part of our project scope.

As our dataset, we use a subset consisting of one million quads from the Tyrolean Knowledge Graph, which contains currently around eight billion triples, set up by Kärle et al. [14]. This dataset is very representative for the real-life data found currently on the Web, and is therefore suitable for a benchmark construction. Based on this dataset and domain specifications[1] defined by Panasiuk et al. [19] [18], we constructed 58 different SHACL shapes that are suitable to be used as valid shapes to clean the data of this specific knowledge graph. The dataset combined with the defined shapes form the benchmark. These two inputs are used for the validation process to test the performance of SHACL implementations in graph databases. We have published the benchmark dataset, as well as the developed SHACL shapes as open research data[2], and discuss them further in detail.

3.1 The Benchmark Dataset

The Tyrolean Knowledge Graph is a knowledge graph that gathers and publicly provides tourism data in the Austrian region of Tyrol [14]. The knowledge graph currently contains more than eight billion triples based on schema.org annotations collected from different sources such as destination management

[1] A domain specification is a domain specific pattern that restricts and extends schema.org for domain and task specific needs. Currently, 84 domain specifications are available that focus on providing representations for tourism related data and can be used for validation purposes [22].

[2] Benchmark for SHACL Performance in Knowledge Bases, doi: 10.17632/jfrdpnb945.1

organizations and geographical information systems. Apart from service types relevant in tourism, like hotels, food and beverage establishments or events, the knowledge graph contains information on additional attributes, like geographic data. The vocabulary of the knowledge graph is schema.org [10], a de-facto standard for semantically annotating data, content, and services on the web.

The SHACL shapes of our benchmark were designed for a subset of the Tyrolean Knowledge Graph containing 30 million quads. Due to resource and software limitations that occurred throughout the evaluation process (described in Section 4.3), the size of the dataset had to be reduced to one million quads. This allows the usage of any subset of the 30 million quads dataset as valid input with a similar validation coverage. For different subsets of the Tyrolean Knowledge Graph or other datasets based on the schema.org ontology, the number of validated classes and properties can vary significantly. Especially the usage of other datasets can lead to an increase of violations, since many SHACL shapes are explicitly designed for properties in Tyrol, e.g. the `latitude` property of `https://schema.org/GeoCoordinates` has to be between the values 46.3 and 47.5.

3.2 The SHACL Shapes

In order to validate data, SHACL requires two different inputs. First, a data graph has to be available that may contain invalid or malformed entries. In our benchmark, this data graph is represented by the subset of the Tyrolean Knowledge Graph. The second required input is the shapes graph that declares constraints upon the given data from the data graph. It is worth mentioning that combining the data and shape graphs together into a single graph is possible, but this has a negative impact on the readability and compatibility, due to the limited support of SHACL by many databases.

There exist two types of shapes: node shapes and property shapes. A node shape specifies constraints that need to be fulfilled by focus nodes. A focus node is an RDF term from the data graph that is validated against a shape. This means that a node shape specifies a focus node for which the remaining constraints of the shape has to apply. A property shape specifies constraints on nodes that are reachable from a given focus node by following a property path. This allows to define clear restrictions on possible values for subjects of a triple or quad, e.g. cardinality constraints, value types and value ranges.

The design philosophy of the SHACL shapes for our benchmark was to validate most of the data in our dataset to make the evaluation accurate and computationally demanding. With that in mind we tried to cover a majority of the properties for each class that appears in the initial 30 million quads dataset, prioritising the most frequent appearing properties. In addition we specified the cardinality for each property according to the domain specifications of *Schema Tourism* [20].

The constructed by us 58 different SHACL shapes can be used as valid shapes to clean the data of this specific knowledge graph. Apart from the instrumental shape-based constraint components, the shapes include the cardinality constraint

components `sh:maxCount` and `sh:minCount`, value type constraint components like `sh:datatype`, and string-based constraint components like `sh:pattern` and `sh:maxLength`. We also utilize the logical constraint component `sh:or`, which led to different results in the validation reports depending on the graph database. The goal of these constraints is to mimic a non-specific validation of a knowledge graph based on its vocabulary.

In addition to the domain related constraints, we approached a design of specific constraints for regional attributes of the Tyrolean Knowledge Graph. Since the dataset should only contain data about businesses and events from Austria and more specifically from Tyrol, values of many properties of that schema can be evaluated precisely. For instance, the property **addressCountry** must have the string "AT", which is the ISO 3166 code for Austria [11]. The shape for this specific example is depicted in Listing 1.1.

Listing 1.1. Shape for schema **PostalAddress**.

```
@prefix schema: <http://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:AddressCountryShape a sh:NodeShape;
    sh:targetClass schema:PostalAddress;
        sh:property [
            sh:path schema:addressCountry;
            sh:datatype xsd:string;
            sh:pattern "AT";
            sh:maxCount 1;
        ].
```

Following the principles described above, the SHACL shapes can be constructed in a similar manner for different datasets. This ensures that our approach is sufficiently representative and generic.

4 Performance Evaluation

In this section, we explain the utilization of the evaluated graph databases for the validation of our dataset. We describe the ease of use, potential pitfalls of the current implementations and results of the overall evaluation.

4.1 Database Selection

To determine which graph databases offer SHACL implementations we performed keyword searches in the respective software documentations and in the source code of open source projects. We also searched for indications on the official database websites, in various forums and contacted some developers of the respective databases. Using this process, we were able to identify a number of graph databases that provide a SHACL implementation. For the project, we decided to use the free-to-use versions of the graph databases AllegroGraph by Franz Inc, Apache Jena by Apache, GraphDB by Ontotext, RDF4J, Stardog by Stardog Union Inc. We should point out that the most popular graph database according to DB-Engines [24], Neo4j, did not have native SHACL support at the

time we conducted our evaluation. Furthermore, we determined the absence of SHACL support for more graph databases.

AllegroGraph [9] is a closed-source persistent RDF store with an additional graph database management system. It is developed and published by Franz Inc and solely supports RDFS as its data scheme. AllegroGraph uses SPARQL as its query language and is available for Linux, OS X and Windows. The problem with the usage lies in the limitations of the free-to-use community edition of the commercial product. This version is only able to store up to five million triples in its triplestore, which is not enough to work with our initial 30 million quads dataset. Since we had to reduce our dataset to one million quads, this restriction did not pose a problem.

Apache Jena is an open-source Java framework by Apache that offers the component TDB [2], which acts as an RDF store and query database management system. Jena is implemented in Java and available on every operating system with a Java virtual machine. SPARQL is supported as a query language.

GraphDB [17] is a commercial, closed-source graph database management system and RDF store by Ontotext. It is implemented in Java and supports OWL/RDFS-schema as an optional feature. SPARQL is used as the query language and it is available for all operating systems that support a Java virtual machine. While GraphDB is a commercial product, a free version with limited functionality exists and was deemed usable for this project. SHACL is not supported natively, instead it supports the validation language using the RDF4J API.

RDF4J [7] is an open-source Java framework and part of the Eclipse project. The functionality of the framework is focused on processing RDF data. It is implemented in Java and available for Linux, OS X, Unix and Windows. SPARQL is supported as a query language.

Stardog [25] is a closed-source graph database management system and RDF store by Stardog Union Inc. It is implemented in Java and supports OWL/RDFS-schema as an optional feature. It uses SPARQL as its query language and is available for Linux, macOS and Windows. Stardog is a commercial product, but its one-year fully-featured non-commercial use license for students renders it usable for our project.

4.2 Evaluation Methodology

The study was conducted by installing the graph databases on a single machine with the following specifications: Intel Core i7-6700K @ 4.00GHz processor, 16 GB PC 3200 DDR4 RAM and a Kingston A400 240 GB SSD. Manjaro Linux 18.1.5 was used as the operating system for the evaluation. The criterion to assess performance is measuring the time it takes to generate a SHACL report. To measure the time the bash command `time` was used for all databases, since the entire validation process should be captured. The same holds for the tested Java frameworks, where our Java implementation could have an impact on the overall time. Therefore, these implementations were kept relatively simple. Since the structure and provided information of the generated reports differ between

databases and the average size of the reports was 1.5 GB, we were not able to compare them in detail except from size and number of violations.

The method used in the evaluation was to measure the SHACL implementation eight times on the same database under the same conditions to increase the accuracy of the results. One requirement is that the dataset is stored in a native, indexed triplestore. The generated reports were piped to the null device (`/dev/null`) for the measurement to reduce the impact of system calls. In addition we provide information regarding the difficulty of using SHACL, the usefulness of the documentation and the quality of the SHACL report produced by the database. Unlike previous measurements, these are of a qualitative nature and consist of a textual description.

4.3 Evaluation Experience

In this subsection, we highlight our experiences in the performance evaluation of each individual database. We explain the usage of the respective implementations of SHACL validation and present certain challenges and pitfalls we encountered. In addition, an extract of the respective validation reports is presented to display the discrepancy of information in the reports. The presented violations occur through the defined property path from Listing 1.2.

Listing 1.2. The property path leading to the violations in this subsection.

```
@prefix schema: <http://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:imagePath sh:path schema:image;
    sh:nodeKind sh:BlankNodeOrIRI;
    sh:maxCount 1;
    sh:or(
        [ sh:datatype schema:ImageObject;]
        [ sh:datatype xsd:string;
        sh:pattern "((?:http|https)://)?(?:www\\.)?
            [\\w\\d\\-_]+\\.\\w{2,3}(\\.\\w{2})?(/(?<=/)
            (?:[\\w\\d\\-./_]+)?)?+[.(jpg|jpeg|gif|tif|png|bmp)]?";]
    )
```

AllegroGraph AllegroGraph provides two different usage options: command-line interface (CLI) and a graphical user interface called `AGWebView`. The graphical user interface has limited functionality and does not support SHACL validation. Neither the import of the dataset represented in the N-Quads format nor the SHACL shapes written in Turtle yielded any problems.

To use SHACL in AllegroGraph, the command-line program `agtool` has to be utilized, which offers a variety of utilities. We used `agtool` to create a new database, import our dataset and shapes and perform the SHACL validation to obtain the report. It is important to note that the AllegroGraph's SHACL implementation necessitates the explicit indication of all data graphs for the evaluation. The tool also allows to specify the output format of choice. We piped the output to the null device for a more precise time measurement. In Listing

1.3 the two generated violations can be seen. The only part missing from the generated violation is a `sh:resultMessage`, which could describe the violation in more detail.

Listing 1.3. Extract of the validation report from **AllegroGraph**.

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix sh: <http://www.w3.org/ns/shacl#>

_:bCDDA3ADEx4018736 sh:sourceConstraintComponent sh:OrConstraintComponent ;
  sh:value "asdasdasdas" ;
  sh:resultPath <http://schema.org/image> ;
  sh:resultSeverity sh:Violation ;
  sh:focusNode <https://smtfy.it/Skxm5Vi17> ;
  rdf:type sh:ValidationResult ;
  sh:sourceShape <http://gdb.benchmark.com/imagePath> .

_:bCDDA3ADEx4018737 sh:sourceConstraintComponent sh:NodeKindConstraintComponent ;
  sh:value "asdasdasdas" ;
  sh:resultPath <http://schema.org/image> ;
  sh:resultSeverity sh:Violation ;
  sh:focusNode <https://smtfy.it/Skxm5Vi17> ;
  rdf:type sh:ValidationResult ;
  sh:sourceShape <http://gdb.benchmark.com/imagePath> .
```

Apache Jena Apache Jena is a Java framework with two available versions. The basic version contains command line interface (CLI) tools for the creation and manipulation of a database. Apache Jena Fuseki offers the same tools as the basic version with additional server functionalities, including a web user interface. We used the CLI, due to the lack of SHACL support in the web interface and the fixed method of measuring time. The CLI tool allows the validation of datasets against SHACL shapes, if both inputs are available as files, but it is not possible to validate data stored in triplestores. To measure the time of validating data stored in the provided triplestore (TDB), we wrote a Java application that uses the Jena API. The documentation was lackluster [1] and only provided an example for validating datasets provided as files. We piped the output to the null device to have a more accurate time measurement. The report itself complies to the W3C standard and the extract for our example is shown in Listing 1.4.

Listing 1.4. Extract of the validation report from **Apache Jena**.

```
@prefix schema: <http://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .

sh:result    [
  a                             sh:ValidationResult ;
  sh:focusNode                  <https://smtfy.it/Skxm5Vi17> ;
  sh:resultMessage              "NodeKind[BlankNodeOrIRI] : Expected
                                BlankNodeOrIRI for \"asdasdasdas\"" ;
  sh:resultPath                 schema:image ;
  sh:resultSeverity             sh:Violation ;
  sh:sourceConstraintComponent  sh:NodeKindConstraintComponent ;
  sh:sourceShape                :imagePath ;
  sh:value                      "asdasdasdas"
] ;

sh:result    [
  a                             sh:ValidationResult ;
```

```
  sh:focusNode                    <https://smtfy.it/Skxm5Vi17> ;
  sh:resultMessage                "Or[NodeShape
                                  [4acc5427c0d3a31145d9a2b6b7c5cb74],
                                  NodeShape[24c96f02c540f8563a4b2914d2af61b4]]
                                  at focusNode \"asdasdasdas\"" ;
  sh:resultPath                   schema:image ;
  sh:resultSeverity               sh:Violation ;
  sh:sourceConstraintComponent    sh:OrConstraintComponent ;
  sh:sourceShape                  :imagePath ;
  sh:value                        "asdasdasdas"
] ;
```

GraphDB GraphDB is a graph database management system built upon the Java framework RDF4J. Due to this, the implementation of SHACL validation is internally equivalent to the one of RDF4J. The implementation of SHACL into GraphDB 9.1 focuses on the specific use case of validation as part of data imports. Validations of already existing repositories are not supported by the workbench. The sole available application of SHACL is restricted by file limits and has the issue that the validation process stops on a single violation and does not yield a validation report, rendering it unusable for our comparative analysis.

To circumvent this problem we tried to implement a validation for GraphDB repositories using the RDF4J API in Java. Even though this appears possible from reading the documentation, the current implementation of SHACL in RDF4J utilizes a specifically defined repository type that requires its input data in form of a local repository or RDF file. An option to utilize an existing repository as provided by GraphDB is currently not available. In contrary to our implementation of the RDF4J evaluation it is also not possible to import the raw storage of the database as a local storage, as GraphDB's file structure differs from the triplestore provided by RDF4J. The only possibility to perform the validation for an entire repository, as we need for our comparative analysis, would be to export the data from the GraphDB repository and import it again into an empty `ShaclRepository` generated by RDF4J. This was not an option for our comparative analysis, as the only relevant information extracted from it would be the export performance of GraphDB's triplestore.

Due to the problems and limitations described in this subsection we decided to not include GraphDB in our quantitative performance evaluation, as its current SHACL implementation is too limited to be meaningfully compared to the implementations of the other graph databases.

RDF4J Similar to Apache Jena, RDF4J is a framework for processing RDF data. Therefore, many features of a complete graph database are missing. The basic RDF4J installation allows the user to store and manipulate data in the provided triplestore, which is capable of handling around 100 million triples [7]. There exists an out-of-the-box approach to validate data via SHACL with the restriction that only RDF files can be used. Even though the entirety of our dataset has to be validated, reading the data from a file instead of an indexed database could have a noticeable impact on our measurements and is therefore not an acceptable method for our evaluation.

To validate data from the provided triplestore, implementing an application that uses the RDF4J framework is required. The application connects to the triplestore, loads the SHACL shapes from a file and performs the validation. Although the described structure seems relatively simple, the outdated documentation [7] was a hurdle and required us to search through the Javadoc [8] in order to achieve the desired outcome.

If a violation of the SHACL constraints occurs, a `RepositoryException` is thrown after the entire data is checked. From this exception the validation report can be generated and written to a file. A downside of this specific implementation is the large amount of main memory necessary. These high resource demands forced us to reduce the original dataset to a subset of one million quads, since larger data exceeded our available resources. In our case 18 GB of heap memory had to be allocated for the Java Virtual Machine (JVM) to successfully generate the validation report. We were not able to determine all reasons for the large memory requirement, because even though our generated report has a size of around 1.1 GB, the used dataset consists of only 160 MB.

The validation report for RDF4J contains the most violations for our specific example. RDF4J interprets any constraint in the `sh:or` construct, and `sh:or` itself as a violation, if none of the constraints are fulfilled. This leads to a total number of four violations, which can be seen in Listing 1.5. The report complies to the W3C standard, but is missing two components. The fields `sh:value` as well as the `sh:resultMessage` are not included in the violation, which makes the identification of the cause of the violation more difficult.

Listing 1.5. Extract of the validation report from **RDF4J**.

```
@prefix sh: <http://www.w3.org/ns/shacl#> .

_:node1dvlsg0iux1660 sh:result _:node1dvlsg0iux3491406 .
_:node1dvlsg0iux3491406 sh:resultPath <http://schema.org/image>;
   sh:detail _:node1dvlsg0iux3491407 .

_:node1dvlsg0iux3491407 a sh:ValidationResult;
   sh:focusNode <https://smtfy.it/Skxm5Vi17>;
   sh:sourceConstraintComponent sh:DatatypeConstraintComponent;
   sh:sourceShape _:node1dvlsg0gqx20;
   sh:resultPath <http://schema.org/image>;
   sh:detail _:node1dvlsg0iux3491408 .

_:node1dvlsg0iux3491408 a sh:ValidationResult;
   sh:focusNode <https://smtfy.it/Skxm5Vi17>;
   sh:sourceConstraintComponent sh:PatternConstraintComponent;
   sh:sourceShape _:node1dvlsg0gqx22;
   sh:resultPath <http://schema.org/image> .

_:node1dvlsg0iux3491406 a sh:ValidationResult;
   sh:focusNode <https://smtfy.it/Skxm5Vi17>;
   sh:sourceConstraintComponent sh:OrConstraintComponent;
   sh:sourceShape <http://gdb.benchmark.com/imagePath> .

_:node1dvlsg0iux1660 sh:result _:node1dvlsg0iux3494319 .

_:node1dvlsg0iux3494319 sh:resultPath <http://schema.org/image>;
   a sh:ValidationResult;
   sh:focusNode <https://smtfy.it/Skxm5Vi17>;
   sh:sourceConstraintComponent sh:NodeKindConstraintComponent;
   sh:sourceShape <http://gdb.benchmark.com/imagePath> .
```

Stardog Stardog provides CLI tools for creating and interacting with the database. There exists also an integrated development environment (IDE) called Stardog Studio that provides a user interface and many features from the CLI tools. It is the only graphical user interface we encountered that allows users to create and validate SHACL shapes for a database. For our evaluation we used the CLI tool, due to the fixed method of time measurement.

While performing the validation process using our benchmark, it is important to increase the limit of returned validation results with the `-l` flag. The default value is 100 and Stardog will stop to validate the rest of the graph after that limit is reached. We piped the output to the null device to have valid time measurement without additional time consumed by writing the report to the command line or a file. The output of the report is in Turtle format and complies with the specification made by W3C. In contrast to the other databases, Stardog generates only a single violation in the report, which still provides enough information about violation itself. This can be seen in Listing 1.6.

Listing 1.6. Extract of the validation report from **Stardog**.

```
@prefix sh: <http://www.w3.org/ns/shacl#> .

_:bnode_2394dca2_2e9a_4242_af3d_e81fc0951ac8_6965339 a sh:ValidationResult ;
   sh:resultSeverity sh:Violation ;
   sh:sourceShape <http://gdb.benchmark.com/imagePath> ;
   sh:sourceConstraintComponent sh:NodeKindConstraintComponent ;
   sh:focusNode <https://smtfy.it/Skxm5Vi17> ;
   sh:resultPath <http://schema.org/image> ;
   sh:value "asdasdasdas" ;
   sh:resultMessage "Image is neither a schema:ImageObject nor a valid URL" .
 _:bnode_2394dca2_2e9a_4242_af3d_e81fc0951ac8_3598217 sh:result
    _:bnode_2394dca2_2e9a_4242_af3d_e81fc0951ac8_6965339 .
```

4.4 Evaluation Results

The focus of our work was to evaluate the performance of the SHACL implementation from each graph database by measuring the required time to validate our data according to the defined SHACL shapes and generate a SHACL validation report. The obtained results are illustrated in Figure 1 and the evaluation specifics are summarized in Table 1.

Based on the measurements a large discrepancy between some graph databases can be observed. Our results show that AllegroGraph requires around 11,6 times longer to validate the data and produce the validation report compared to the fastest graph database Stardog, which only took ~2 minutes for the entire dataset. A noticeable difference is also visible regarding both frameworks. RDF4J requires around 2,4 times longer than Apache Jena.

5 Discussion

The results of our performance evaluation using the benchmark suggest Stardog's implementation as the currently best performing option for SHACL validation, with Apache Jena following at a close second place. The validation in RDF4J

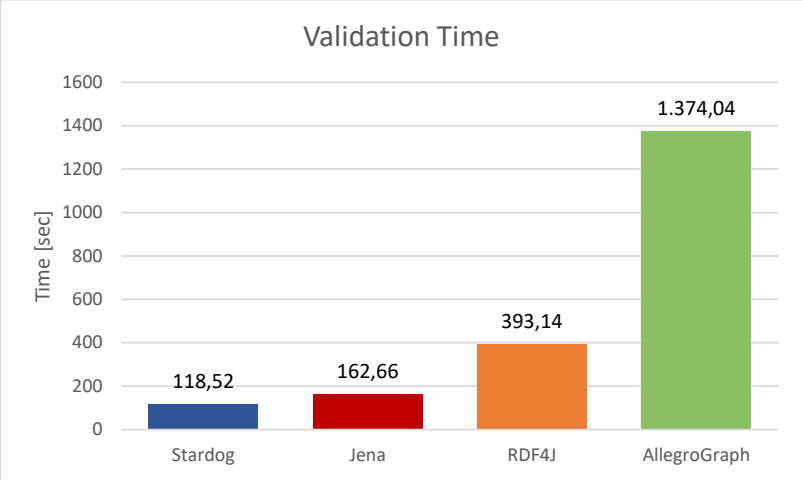

Fig. 1. Average validation time out of 8 measurements.

Table 1. Evaluation results of one million N-Quads based on the 58 SHACL shapes.

Graph database	**Validation time (sec)**	**Number of violations**	**Report size**	**W3C compliant report**	**Require-ments**	**Limitations**
Allegro Graph	1.374,04	3.567.930	1,24 GB	Yes	Validation performed by agtool	Free version supports only 5M triples
Apache Jena	162,66	3.615.753	1,82 GB	Yes	Addition application that uses Jena API was developed	Lack of documentation
RDF4J	393,14	3.676.105	1,16GB	Yes, but no specifics on errors	Addition implementation is needed	Requires large amount of memory
Stardog	118,52	3.597.171	1,78 GB	Yes	Validation limit adjustments	Not defined

performed significantly worse, tripling the runtime for the equivalent task in Stardog. AllegroGraph's current implementation of SHACL validation performed the worst out of the four evaluated graph databases.

In addition to performance, we decided to include ease of use and resource demand into our evaluation. The evaluation of those factors displays a disparity between the two frameworks (RDF4J and Apache Jena) and the two complete database management systems. Concerning ease of use, the frameworks only provide out-of-the-box validation implementations for data provided as files. Validation for additional use cases like the included triplestores have to be implemented by the users of the framework, rendering the barrier of entry for the usage of those validation options higher than for the database management systems.

Another disparity between the database management systems and the frameworks can be found by looking at the differences in resource requirements for equivalent SHACL validations. For the specific case of our evaluation the validation of our data (160 MB) in Stardog utilized about 1.2 GB of RAM, which is easily manageable by even older personal machines. The resource demand of AllegroGraph amounts to about 3.7 GB of RAM, which is more than three times the amount of memory required by Stardog. Both frameworks demand a significantly higher amount of resources, with Jena requiring about 14 GB and RDF4J requiring about 18 GB of RAM for the evaluation of the same dataset using the same SHACL shapes. This exceeds the power of many personal machines and can put a strain on servers.

We conclude from our evaluation that Stardog's implementation of SHACL validation seems to be the best option at the moment, as it offers the best performing validation operation with the lowest resource demand for task like our benchmark. Additionally, no programming is required from the user, rendering the database management system significantly easier to use than the other well performing candidates. Our evaluation also uncovered some pitfalls concerning the state-of-the-art in SHACL implementations.

As SHACL can be seen as relatively new compared to other standards of the Semantic Web, only a few graph databases currently offer an implementation. For most databases that offer validation using the recommendation, the option does not seem like a focus of development. During our evaluation we encountered a significant number of lacking or inaccurate documentation. Some current implementations are unfinished, missing core functionality of the W3C standard or only adhere to specific use cases.

The large amount of violations prevents us from having a ground truth, thus we can not be sure to find all violations. Even comparing results between the different databases is difficult, because of the way some outputs are structured and messages are generated for intermediate nodes down to the node with the violation. Out of the four evaluated graph databases only AllegroGraph generated a validation report that deviated from the SHACL W3C standard, presenting the violations in a different format. This resulted in differences of information portrayed by the validation outputs, rendering them hard to compare or validate.

In addition to this problem the implementations of SHACL validation differ in their interpretation of certain shapes. An example for this behaviour is the `sh:or` shape, which depending on the implementation can either report all violations or ignore the other arguments after a first violation is found. Another example for inconsistencies is the treatment of subclasses, which can lead to single violations being thrown multiple times.

Due to those pitfalls we conclude that SHACL is currently still very new as a de-facto standard to perform full-scale complete performance evaluations, but our work is a first step in this direction. The still low number of graph databases offering implementations in combination with the displayed lack of adherence to the standard currently results in too many factors reducing the informative value of performance evaluations and benchmarks significantly.

6 Conclusions

As contributions for our work, we have developed a benchmark for the evaluation of the performance of the SHACL processing in graph databases and have applied it to the evaluation of four graph databases. The benchmark is published as open research data and is available for further experimentation and development.

As described in Section 3, our benchmark is designed to maximize utility for the current state of the art and other specific constraints (such as being restricted to the use of the free versions of the graph databases and having limited hardware capacities). While we think that it is generally applicable and yields accurate performance evaluations, other constraints or positive developments in the implementations of SHACL evaluation in graph databases may render other, more advanced approaches in the future.

One possible approach could be to provide shapes for a complete validation of a dataset, instead of just a representative subset. A benchmark evaluating the performance of complete validations would yield a more representative evaluation, but because of the substantial differences in resource demand the result would depend more on the evaluation hardware. Other approaches that could be worth trying out include handcrafted benchmark databases to be able to test specific constraints in a more controlled fashion and the expansion of the evaluation on truthfulness of the validation reports.

References

1. Apache: Jena SHACL. `https://jena.apache.org/documentation/shacl/index.html` (2019)
2. Apache: Jena TDB. `https://jena.apache.org/documentation/tdb` (2019)
3. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American **284**(5), 34–43 (2001)
4. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. International Journal on Semantic Web and Information Systems (IJSWIS) **5**(2), 1–24 (2009)

5. Capotă, M., Hegeman, T., Iosup, A., Prat-Pérez, A., Erling, O., Boncz, P.: Graphalytics: A big data benchmark for graph-processing platforms. In: Proceedings of the GRADES'15. p. 7. ACM (2015)
6. Dominguez-Sal, D., Urbón-Bayes, P., Giménez-Vanó, A., Gómez-Villamor, S., Martínez-Bazan, N., Larriba-Pey, J.L.: Survey of graph database performance on the HPC scalable graph analysis benchmark. In: International Conference on Web-Age Information Management. pp. 37–48. Springer (2010)
7. Eclipse PMC: RDF4J. `https://rdf4j.org/documentation` (2019)
8. Eclipse PMC: RDF4J Javadoc. `https://rdf4j.org/javadoc/latest/` (2019)
9. Franz Inc: AllegroGraph. `https://franz.com/agraph/support/documentation` (2019)
10. Guha, R.V., Brickley, D., Macbeth, S.: Schema. org: evolution of structured data on the web. Communications of the ACM **59**(2), 44–51 (2016)
11. International Organization for Standardization: ISO 3166 Standard for Austria. `https://www.iso.org/obp/ui/#iso:code:3166:AT` (2019)
12. Jouili, S., Vansteenberghe, V.: An empirical comparison of graph databases. In: 2013 International Conference on Social Computing. pp. 708–715. IEEE (2013)
13. Kärle, E., Fensel, A., Toma, I., Fensel, D.: Why Are There More Hotels in Tyrol than in Austria? Analyzing Schema.org Usage in the Hotel Domain. In: Information and Communication Technologies in Tourism 2016: Proceedings of the International Conference in Bilbao, Spain. pp. 99–112. Springer (2016)
14. Kärle, E., Şimşek, U., Panasiuk, O., Fensel, D.: Building an ecosystem for the tyrolean tourism knowledge graph. In: International Conference on Web Engineering. pp. 260–267. Springer (2018)
15. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL), W3C Recommendation 20 July 2017. URL: https://www.w3.org/TR/shacl (2017)
16. McColl, R.C., Ediger, D., Poovey, J., Campbell, D., Bader, D.A.: A performance evaluation of open source graph databases. In: Proceedings of the first workshop on parallel programming for analytics applications. pp. 11–18. ACM (2014)
17. Ontotext: GraphDB. `http://graphdb.ontotext.com/documentation` (2019)
18. Panasiuk, O., Kärle, E., Şimşek, U., Fensel, D.: Defining tourism domains for semantic annotation of web content. e-Review of Tourism Research **9** (Jan 2018), `https://arxiv.org/abs/1711.03425`, research notes from the ENTER 2018 Conference on ICT in Tourism.
19. Panasiuk, O., Akbar, Z., Gerrier, T., Fensel, D.: Representing geodata for tourism with schema. org. In: GISTAM. pp. 239–246 (2018)
20. Schema Tourism Working Group: Schema tourism. `https://ds.sti2.org/` (2020)
21. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP2Bench: a SPARQL performance benchmark. In: 2009 IEEE 25th International Conference on Data Engineering. pp. 222–233. IEEE (2009)
22. Şimşek, U., Kärle, E., Holzknecht, O., Fensel, D.: Domain specific semantic validation of schema.org annotations. In: Petrenko, A.K., Voronkov, A. (eds.) Perspectives of System Informatics. pp. 417–429. Springer, Cham (2018)
23. Singhal, A.: Introducing the knowledge graph: things, not strings. `https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html` (2012)
24. Solid IT gmbh: Db-engines ranking of graph dbms. `https://db-engines.com/en/ranking/graph+dbms` (2020)
25. Stardog Union Inc.: Stardog. `https://www.stardog.com/docs` (2019)
26. Ugander, J., Karrer, B., Backstrom, L., Marlow, C.: The anatomy of the facebook social graph. CoRR **abs/1111.4503** (2011), `http://arxiv.org/abs/1111.4503`