Reproducible Evaluation of Semantic Storage Options

Jedrzej Rybicki* and Benedikt von St. Vieth†
Juelich Supercomputing Center (JSC)
Email: *j.rybicki@fz-juelich.de, †b.von.st.vieth@fz-juelich.de

Abstract—**Distributed infrastructures are continuously challenged with the task of storing and managing different types of data. To this end, suitability and performance evaluations of different available technologies have to be conducted. We motivate our work with the concrete challenge of storing semantic annotations in an efficient way. We treat this problem from the resource provider perspective. The paper includes work in progress of evaluating possible storage engines for semantic annotations. The main focus, however, is on creating a framework to conduct such evaluations in a transparent and reproducible way. Our approach is based on Docker tools, and, therefore, the tests can be run on different platforms, and can be repeated if new version of the evaluated technologies become available.**

***Keywords*–Deploying Linked data; Reproducibility; Distributed infrastructures.**

I. INTRODUCTION

Distributed research infrastructures like EUDAT (EUropean DATa [1]) provide generic services to manage research data in an efficient and cost-effective way. Since the research communities which use the services advance over time, they are constantly expressing new requirements with respect to kinds of data and possible usages that the infrastructure should be able to handle. To this end, resource and service providers are constantly evaluating possible approaches and new technologies. Such evaluation must adhere to scientific standards in terms of methodology, transparency, and reproducibility.

In our previous paper [2], we have shown how Docker [3] can be leveraged to provide on-demand instances of popular web services in context of a distributed research infrastructure. Although, such seamless provisioning of services can be used to conduct reproducible research, there are more aspects to it. In this paper, we will exercise a whole workflow from testing, through result processing, up to visualization of the outcomes. We will use Docker and `docker-compose` to conduct the steps in a transparent, sharable, and reproducible way. We will test our approach by evaluating storage options to handle semantic annotations.

Semantic annotations are a very powerful tool to work with data in distributed infrastructures. On the very high level, they allow to add comments to entities managed in the infrastructure. An example would be a keyword attached to a digital object, but more sophisticated examples are envisioned as well. We will explain the model in more detail later in this paper, but astute reader can imagine that efficient annotations handling should enable different types of queries. It should be possible to retrieve all annotations for a given object, but also reverse lookups (i.e., localizing all data objects with given keyword in our example) will be used. The uptake of this new service will only happen if sufficient performance of both kind of queries is offered.

There are many ways in which annotations can be stored. The EUDAT service plans to use the World Wide Web Consortium (W3C) Annotation Data Model [4]. Since it is based on JavaScript Object Notation for Linked Data (JSON-LD), an obvious approach would be to use document stores for the task. Since annotations are attached to data object, the whole data set forms a graph with nodes as managed entities and annotations as relations. Thus, graph databases could also serve as storage backends. Regardless of the technology used it should be possible to evaluate its performance and tune it to account for this particular use case. Such a tuning is usually done in an iterative way, where the results of each change are verified, it is critical to possess tools that can perform the benchmarking tests in a reproducible manner.

The rest of this paper is structured as follows. In Section II, we explain what semantic annotations are and discuss suitable storage options. Subsequently, a short introduction to Docker follows and technical details of our effort to make the evaluation reproducible are presented. Section IV comprises the preliminary results for selected storage options. We conclude the paper with an outlook.

II. SEMANTIC ANNOTATIONS

EUDAT is working on enabling semantic annotations of the objects stored in its distributed research infrastructure. The current approach is to use the W3C format for annotations. The W3C web annotation data format is pretty simple: Each annotation is a relation between a *body*, e.g., EUDAT data object, and *target*, e.g., metadata describing that object. Basic annotation model is shown in Figure 1.

It is important to notice that both target and body have unique identifiers. These are crucial from the user perspective. It can be expect that the users will be interested to view a list of all annotations for given body id, i.e., all metadata descriptions for a given data object. But also a reverse lookup producing all the data objects with specific tag (i.e., a retrieval by target id) embodies important functionality. These expected usage scenarios were used as corner stones for our benchmarks. In particular, we defined three metrics for the storage backend evaluation:

- creation times (creation of new, non-existing annotations),
- annotation retrieval by target id,
- annotation retrieval by body id.

There are many options to store semantic annotations. One obvious approach would be to stick to the JSON-LD rendering as proposed by W3C, use is also as internal storing format and find a storage backend which can support it. There are many NoSQL solutions on the market, which can store JSON documents, with MongoDB [5] being one of the most popular.

```
{
  "@context": "http://www.w3.org/ns/anno.jsonld",
  "id": "http://example.org/anno2",
  "type": "Annotation",
  "body": {
    "id": "http://example.org/analysis1.mp3",
    "format": "audio/mpeg",
    "language": "fr"
  },
  "target": {
    "id": "http://example.gov/patent1.pdf",
    "format": "application/pdf",
    "language": ["en", "ar"],
    "textDirection": "ltr",
    "processingLanguage": "en"
  }
}
```

Figure 1. Basic W3C Annotation Data Model

Another storage option is based on the observation that annotations and annotated objects form a graph (with annotations as edges). To account for this way of thinking, a graph database like neo4j [6] could serve as a storage backend. We decided not to include relation database systems in our evaluation. The reasons are twofold. As can be seen on the Figure 1, the model used for annotations is dynamic with optional fields (like format or language), such flexibility is hard to deal with when using relational database. Second reason was the fact that creation and explorations of data objects and annotations would involve a lot of joins in the case of relational model: One would have to jump from one annotation body (i.e., data object) to a target (e.g., keyword) and then again to body to identify objects similar to the starting node.

III. Experimental setup

To obtain meaningful evaluation results it is important to minimize the number of "moving parts" and reduce the testing environment to components which are absolutely necessary. In particular, we were not interested in the performance of the web interface that will be used to work with annotations or the performance penalty caused by its integration with other EUDAT services. Therefore, we have written a small program in Python, with methods for generating annotations with unique body and target identifiers, and for retrieval of the data. The methods use simple interfaces to access selected database stores: MongoDB and neo4j.

A. Docker

To enable easy reproducibility of the conducted tests, we have prepared a Docker-based environment. Docker is a lightweight virtualization solution which is using Linux Kernel features like *namespaces* and *cgroups* to isolate guest and host systems. Docker uses image templates to start containers (i.e., guest processes). Images are build in a hierarchical fashion by applying a "write-on-modify" principle. Thus, it is possible to trace back all the changes done in a given image during the installation and configuration of the software it comprises. Docker provides tools to easily exchange the images via a public Docker Hub [7], or private on-site repositories. Docker introduces notion of official images which are created and maintained by the providers of a given technology. There are official images for major Linux distributions but also for popular content management systems, or databases. Docker ecosystem embrace many tools, we will use `docker-compose` [8] which is an orchestration solution to start and manage more complex Docker-based deployments.

The main reason why we are using Docker is due to the virtualization it is possible to run our test programs on almost any platform (regardless of the operating system it uses). The images also contains the dependencies and libraries required, so again the configuration of the host system may be neglected. The possibility to review all the changes done in a particular Docker image enables transparency and understandability of the obtained results. Last but not least, by using Docker featured called volumes, it is possible to separate data from the programs, in our case: results and processing tools.

B. Solution details

Both technology providers (MongoDB and neo4j) offer official images for their databases which we used for our evaluation. We created a Docker image with our testing program and prepared a `docker-compose`-based testing environment. The source code and documentation is stored on GitHub [9], allowing for verification and repetition of the tests. In fact, we plan to reuse this framework to do some further testing of different EUDAT-inspired use cases in the future.

Given a system with a running Docker daemon and `docker-compose`, starting tests is a matter of merely issuing one command like:

```
docker-compose run tester --name exp1
```

The last parameter of the above command is not strictly required, it attaches a user-defined name to the particular experimental run which is convenient for the further analysis.

Also, Docker images for processing of the results and visualizing them are provided. The first step transforms the results from the evaluation by using following command:

```
docker run --volumes-from exp1 processor
```

Please note that we are using the name assigned to the experiment in the previous step (`exp1`), the `--volumes-from` parameter is used to attach storage volume with the data produced in the first step to the newly created Docker container.

Finally, the plots that we present in the following section are created with help of `gnuplot` [10] and other tools embodied in a Docker image which again uses volume with data from previous steps and can be run with a simple command:

```
docker run --volumes-from exp1 visualizer
```

To enable sequential processing of data, we internally agreed to store all the data (results, visualizations, etc.) in the same path defined as a Docker volume. Thanks to this contract, we can guarantee that data are not becoming part of the Docker images and thus will not hinder their reuse. Secondly, it is easily possible to extend the workflow by adding new steps or modify existing ones, for instance, if different types of visualization are required.

IV. Results

The tests are defined by three parameters:

1) *engine*: database engine (currently MongoDB and neo4j),

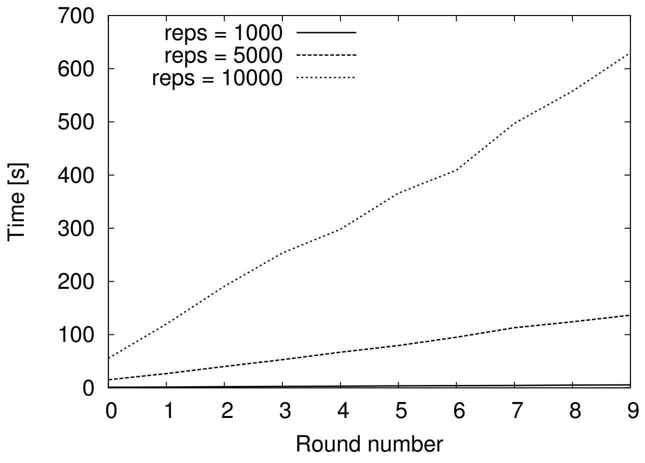

Figure 2. Retrieval scalability for MongoDB ($reps$ records are added, and $reps$ random records are retrieved in each round).

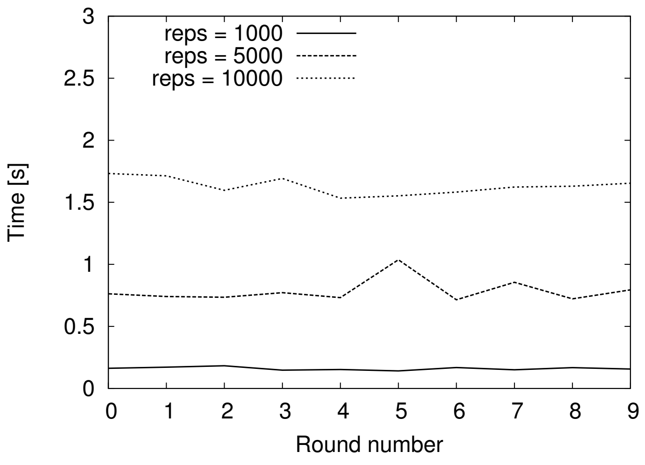

Figure 3. Retrieval scalability for neo4j ($reps$ new records are added, and $reps$ random records are retrieved in each round).

2) $rounds$: number of rounds,
3) $reps$: number of repetitions in each round.

The tests were divided into rounds and in each round all the above database operations were conducted in the given order. Firstly $reps$ number of records were created, subsequently random (with repetition) $reps$ annotations were retrieved by specifying existing $target.id$, finally $reps$ random annotations were fetched by $body.id$. We measured time of each activity, that is complete time to create records, time to retrieve all $reps$ record by target and body id. Three time measurements were made in each round. Please note, that no records were removed, i.e., for given $reps = 1000$, the database grown in each round by new 1000 record.

All the tests were run on the same virtual machine with 4 VCPUs, 4GB RAM, using Ubuntu 16.04.

In Figure 2 and Figure 3, we depicted the retrieval scalability of each database. For that we conducted three experiments with different values of $reps$, each had 10 rounds. Figure 2 shows that the performance of MongoDB is dramatically decreasing with the increasing number of records in store.

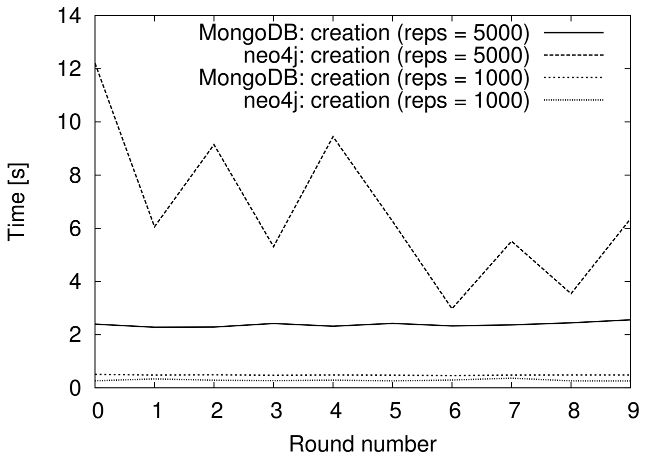

Figure 4. Comparison of creation scalability ($reps$ new records are added in each round).

Also, the absolute values achieved by MongoDB are not very good, to retrieve 10 000 random annotations from a database with 90 000 documents, more than one minute is required.

The retrieval times for the same amount of data from the neo4j database of the same size are much smaller as can be seen in Figure 3 (please note that the y axis was scaled comparing to Figure 2). Also, the scalability of neo4j is much better, neo4j produces constant answer times regardless of the size of the database. For comparison with the MongoDB, to retrieve 10 000 random entities from a neo4j graph with 90 000 annotations, only $1.65s$ is required.

The situation is a little bit different for creation times. We depicted them in Figure 4. For smaller values of $reps$ neo4j outperforms MongoDB but with $reps = 5000$ MongoDB is faster. We also conducted the tests for higher values of $reps$ (not depicted for the sake of clarity) and MongoDB maintained its advantage in this regard. Neo4j also displays high variance in the creation times and the values decreased over time. This kind of behavior could be caused by the fact that neo4j is written in Java and Scala and the Java Virtual Machine can need some time to "warm up". Perhaps further investigations are required there, like warm-up phase before the actual tests to at least get rid of the high delay in the first round.

V. Conclusion and future work

In this paper, we evaluate options for storing semantic annotations in a reproducible manner. We selected two technologies: MongoDB and neo4j. We believe that the presented approach is applicable also for other use cases. Our results support the hypothesis that annotations naturally form a graph and thus, can be efficiently stored in a graph database. Further investigations of the weak creation performance might be necessary.

It is clear that the presented work in progress is just a first step towards answering the question on how to efficiently manage annotation-like data. Both neo4j and MongoDB offer numerous possibilities to fine tune the performance to account for particular data and query types. Although, it was not the primary goal of this work we believe that by having a possibility to conduct performance evaluation in a repeatable way, such a tuning can be done much faster.

The challenge of constantly evaluating emerging technologies and dealing with the management of new kinds of data is common for distributed research infrastructures. Therefore, it is crucial to define evaluations in a reproducible manner. Our Docker-based toolkit has proven its potential as a basis for such reproducible computer-based experiments. In our future work, we will look into ways of extending this toolkit with a means of executing whole workflows rather that manually starting single steps as we currently do.

Acknowledgment

The work has been supported by EUDAT2020, funded by the European Union under the Horizon 2020 programme - DG CONNECT e-Infrastructures (Contract No. 654065).

References

[1] W. Gentzsch, D. Lecarpentier, and P. Wittenburg, "Big data in science and the EUDAT project," in *SRII Global Conference*, Apr. 2014, pp. 191–194.

[2] J. Rybicki and B. v. St. Vieth, "DARIAH Meta Hosting: Sharing software in a distributed infrastructure," in *MIPRO '15: 38th IEEE International Convention on Information and Communication Technology, Electronics and Microelectronics*, May 2015, pp. 217–222.

[3] (2016, Dec.) Docker. [Online]. Available: https://www.docker.com/

[4] (2016, Nov.) Web annotation data model. [Online]. Available: https://www.w3.org/TR/annotation-model/

[5] (2016, Dec.) MongoDB. [Online]. Available: https://www.mongodb.com/

[6] J. Webber, "A programmatic introduction to Neo4j," in *SPLASH '12: 3rd ACM Annual Conference on Systems, Programming, and Applications: Software for Humanity*, Oct. 2012, pp. 217–218.

[7] (2016, Dec.) Docker Hub. [Online]. Available: https://hub.docker.com/

[8] (2016, Dec.) Docker Compose. [Online]. Available: https://docs.docker.com/compose/

[9] (2016, Dec.) Annotations scalablity. [Online]. Available: https://github.com/httpPrincess/annotations-scalability

[10] (2016, Dec.) Gnuplot. [Online]. Available: http://gnuplot.sourceforge.net/