Benchmarking Big Linked Data

Michael Röder
AKSW
Leipzig University
Leipzig, Germany
roeder@informatik.uni-leipzig.de

Axel-Cyrille Ngonga Ngomo
AKSW
Leipzig University
Leipzig, Germany
ngonga@informatik.uni-leipzig.de

***Abstract*—Big Data is gradually getting adopted in the new data economy. Systems are constantly being developed in order to support the booming exchange of data in the Web and the Enterprise. Hence, ever more complex requirements emerge, pertaining to the efficiency and effectiveness of solutions driven this data. The provision of high-quality benchmarks and benchmarking results yields the potential of pushing the development of better approaches and solutions. This observation holds in particular for Big Linked Data, which is a particular manifestation of Big Data available as Linked Data. This data representation is increasingly being used across the Web. We present a benchmarking platform designed for the unified execution of benchmarks that address the life cycle of Big Linked Data. We show how the platform address the features of Big Data and preliminary results in the area of knowledge extraction from text streams.**

***Keywords*-Big Data, Linked Data, Benchmarking Platform, KPIs, Named Entity Recognition and Linking**

I. Introduction

While the adoption of Big Linked Data is increasing both in academia and in industry, the selection of the right frameworks for a given application driven by Big Linked Data remains elusive. This is partly due to the lack of comparable evaluation results for the technologies which deal with this type of data. The provision of benchmarks has been regarded as having provided key contributions in the development of efficient and effective solutions in several domains as well as in facilitating the selection of solutions from these domains. For example, the TPC family of benchmarks is widely regarded as having provided the foundation for the development of efficient relational databases [1]. Modern examples of benchmarks that have achieved similar effects include the QALD [2] and BioASQ [3] benchmarks, which have successfully contributed to an increase of the performance of question answering systems over Linked Data as well as in the bio-medical domain. Modern benchmarking platforms have also contributed to ensuring the comparability of the measurements used to evaluate the performance of systems. For example, benchmarking platforms such as BAT and GERBIL [4] provide consistent implementations and the corresponding theoretical frameworks to evaluate named entity linking frameworks in a consistent manner. Based on these insights, we are developing the HOBBIT (Holistic Benchmarking of Big Linked Data) platform to support the informed selection of components for Big Linked Data processing platforms as well as to push the development of Big Linked Data solutions. The HOBBIT platform is generic as it allows benchmarking any step of the Linked Data life cycle. In addition to being open-source and thus available for local experimentation, the platform will also be provided as an online instance running on a cluster that will allow for comparable results within benchmarking challenges and campaigns. In this paper, we detail the infrastructure of the platform and present preliminary results in the area of knowledge extraction.

The rest of this paper is structured as follows: We begin by giving an overview of the state of the art in benchmarking Linked Data. Thereafter, we present requirements to the benchmarking platform that were gathered from 68 experts. We then derive an architecture for the platform. A preliminary implementation of the said architecture is finally used to benchmark knowledge extraction framework along the axes of accuracy and scalability.

II. Related Work

A large number of benchmarks have been developed to assess the performance of frameworks from single areas of Linked Data life cycle. We hence give a brief incomplete overview of existing benchmarks and refer to survey papers when possible.

The first family of benchmarks is concerned with storage and query processing. The Lehigh University Benchmark [5] is a synthetic benchmark aiming to test triple stores and reasoners for their reasoning capabilities. The SP^{2}Bench [6] is a synthetic benchmark for testing the query processing capabilities of triple stores. The Berlin SPARQL Benchmark (BSBM) [7] is a synthetic triple stores benchmark based on an e-commerce use case in which a set of products is provided by a set of vendors and consumers post reviews regarding those products. The SRBench [8] is an RDF benchmark designed for benchmarking streaming RDF/SPARQL engines. In [9], the authors propose a synthetic queries benchmark centered around social network data; more specifically the social music network *Last.fm* [10] proposes a SPARQL benchmark based on an electronic publishing scenario. The Waterloo SPARQL Diversity Test Suite [11] provides synthetic data and a query generator to generate a large number of queries from a total of 125 query templates. The DBpedia SPARQL Benchmark (DBPSB) [12], [13] uses real data, i.e., DBpedia, and real queries, i.e., the query log of the DBpedia endpoint, for benchmarking. However, this benchmark does not consider all important query features. This drawback is addressed by the FEASIBLE benchmark [14], a real benchmark generation framework which can generate customized benchmarks out of query logs. Finally FedBench [15] is a benchmark for federated SAPRQL query processing.

Several benchmarks have also been developed in the area of interlinking datasets represented using the Resource Description Framework (RDF), which is the pillar of Linked Data and the Semantic Web [16]. A recent in-detail comparison of instance matching benchmarks can be found in [17]. The authors show that there are several benchmarks using either real or synthetically generated datasets.

In the area of Information Extraction–especially Named Entity Recognition and Linking–there are several datasets that are used for evaluating the effectiveness of recognition and linking systems. Additionally, there were several conferences and workshops aiming at the comparison of information extraction systems ranging from the Message Understanding Conference [18], the Conference on Computational Natural Language Learning [19], the Automatic Content Extraction challenge [20], the text analytics conference hosting the workshop on knowledge base population [21], the Senseval challenge [22], the Making Sense of Microposts workshop series [23] to the Open Knowledge Extraction Challenge [24]. In 2014, Carmel et al. [25] introduced one of the first Web-based evaluation systems for Named Entity Recognition and Linking that served the centerpiece of the entity recognition and disambiguation challenge. Here, all frameworks are evaluated against the same unseen dataset and provided with corresponding results. Similarly, the BAT-framework [26] is designed to facilitate the benchmarking based on these datasets by combining seven Wikipedia-based systems and five datasets. It offers 6 different experiment types and six evaluation measures. The GERBIL framework [4] extended this idea by being knowledge base agnostic and offering additional analysis of the results [27]. Additionally, it is hosted as a web platform offering an interface with which system developers can compare their current approach with other publicly available approaches.

In the area of Question Answering, the QA track [28] of the TREC conference aims at providing domain-independent evaluations over large, unstructured corpora since 1998. The BioASQ series [3] challenges semantic indexing as well as Question Answering systems on biomedical data and is currently at its fifth implementation. The developers of the open Question Answering platform OKBQA[1] released the NLQ datasets[2]. The Question Answering over Linked Data [29] campaign, currently running in its 6th instantiation, is a diverse evaluation series including 1) RDF-based, 2) hybrid, i.e., RDF and textual data, 3) statistical as well as 4) multi knowledge base and 5) music-domain-based benchmarks.

While benchmarking efforts exist for single steps of the Linked Data life cycle, there is no established evaluation framework that is able to cover all areas and measures both the efficiency and effectiveness of solutions. With HOBBIT, we present exactly such as platform.

III. Requirements

The requirements underlying the development of the HOBBIT benchmarking platform were mainly derived from the results of a community survey [30] with 68 participants.

A. Functional requirements

- The main functionality of the platform is the execution of benchmarks.
- The benchmark results should be presented in human- and machine-readable form.
- It should be possible to add new benchmarks.
- It should be possible to add new systems.
- The platform should offer analysis of results as shown in [27].
- The key performance indicators (KPIs) should include the effectiveness, e.g., the accuracy, and the efficiency, e.g., run time of systems. The platform should also support measuring the scalability of solutions.
- The platform should support the benchmarking of distributed systems.
- The platform should support distributed data generation.
- The platform should support the repeatability of experiments.

B. Qualitative requirements

- The benchmarks should be easy to use and interfaces provided should be as simple as possible.
- The platform should support different programming languages.
- The results should be archived safely for later reference.
- The platform needs to be robust regarding faulty benchmarks or systems.

IV. Architecture Overview

Figure 1 gives an overview of the architecture of the HOBBIT platform. The development of the platform was driven by the requirements explicated in the previous section. The platform comprises several components and is based on a container architecture, i.e., the components are

[1]http://www.okbqa.org
[2]http://2015.okbqa.org/nlq

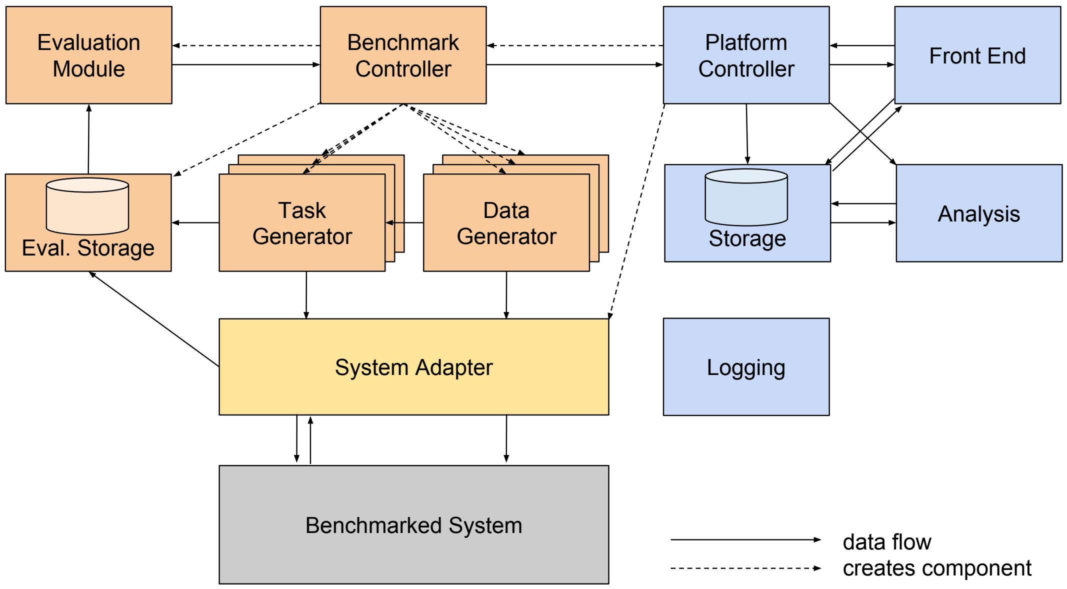

Figure 1. Overview of the platform components.

implemented as independent containers. The communication between these components is ensured by means of a message bus.[3] As shown by the colours in Figure 1, the platform can be separated into two parts. The first part comprises the platform components that always run when the platform is executed (displayed in blue in the figure). The second part contains all components that belong to a particular experiment, i.e., the benchmark components as well as the benchmarked system and its adapter (shown in orange, yellow and grey). The system that is benchmarked as well as the system adapter are not regarded as belonging to the platform itself. Consequently, they do not belong to the group of benchmark components nor to the platform components.

The data flow shown in the overview picture is implemented using the message bus. It has to be noted that the platform is modularized so as to allow the benchmark components to use a different communication technology. However, throughout the document it is assumed that the benchmark components rely on RabbitMQ as well.

V. Benchmark Workflow

In this section, the general workflow carried out to benchmark a system is described. Figure 2 shows a sequence diagram containing the steps as well as the type of communication that is used. Note that the orchestration of the single benchmark components is part of the benchmark and can be different across different benchmark implementations.

1) The platform controller makes sure that a benchmark can be started. This includes a check to make sure that all hardware nodes of the cluster are available.
2) The platform controller generates the system adapter for the system.
   - The system adapter initializes itself,
   - starts the system to be benchmarked and makes sure that it is working properly.
   - It sends a message to the platform controller to indicate that it is ready.
3) The platform controller generates the benchmark controller.
   - The benchmark controller generates the data and task generators as well as the evaluation storage.
   - It sends a message to the platform controller to indicate that it is ready.
4) The platform controller waits until the system adapter as well as the benchmark controller are ready before it sends a start signal to the benchmark controller which starts the data generators.
   - The data generators start the data generation algorithms to create the data that will underly the benchmark. The data is sent to the system adapter and to the task generators.
   - The task generators generate the tasks and send them to the system adapter, which triggers the required processing of the data in the system to benchmark.
   - The system response is forwarded to the evalua-

[3]After a comparison of different frameworks we decided to use Docker as framework for the containerization and RabbitMQ as message bus. However, the focus of this paper is the architecture and not a comparison of technologies. Since it would be possible to use other technologies as well for the same architecture, the choice of tools is not discussed further.

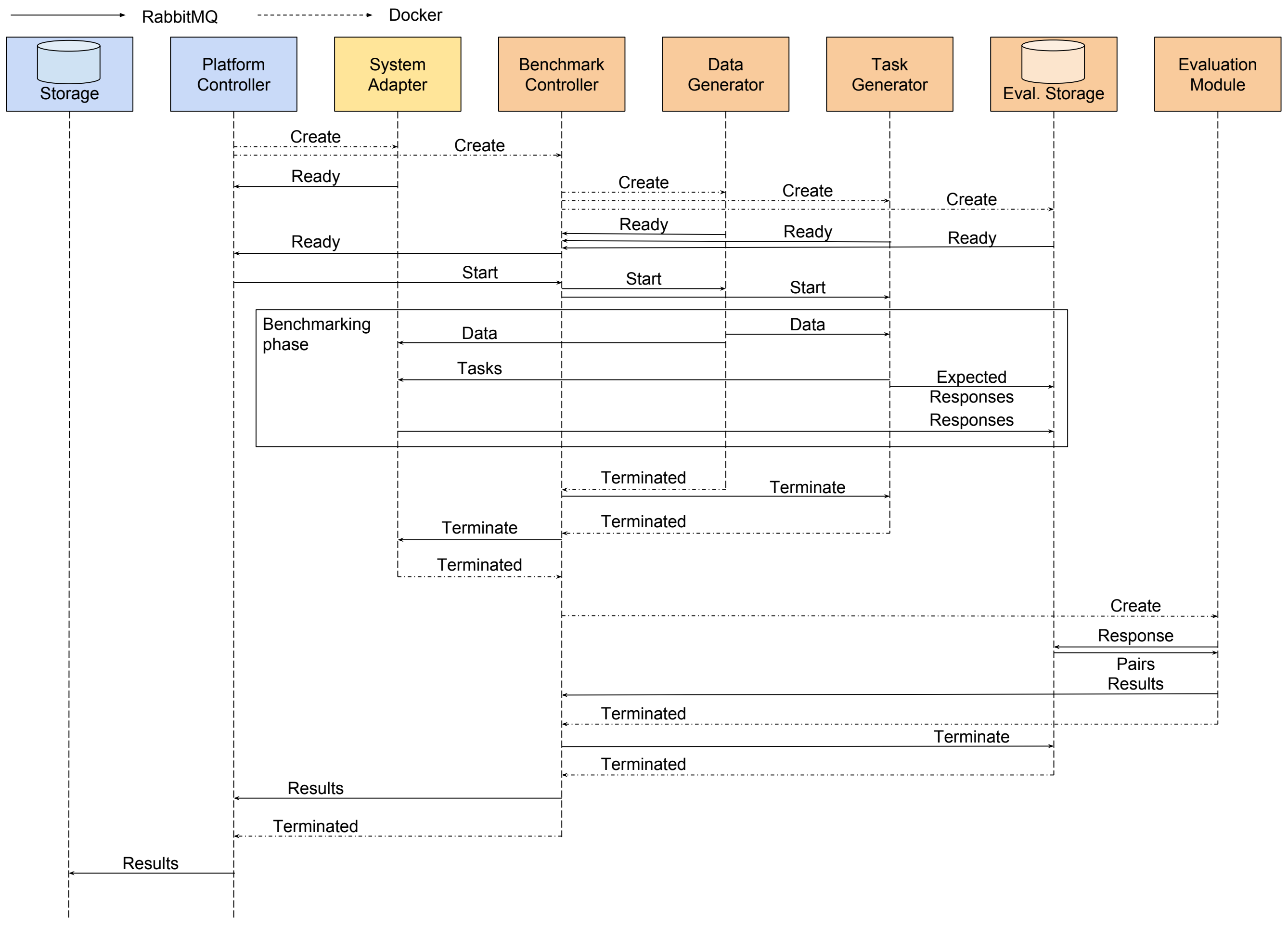

Figure 2. Overview of the general benchmarking workflow. For keeping simplicity, the system as well as the front end are left out and the benchmark controller creates other containers directly, without sending requests to the platform controller.

tion storage by the system adapter.
- The task generators store the corresponding expected result in the evaluation storage.

5) After the data and task generators finished their work the benchmarking phase ends and the generators as well as the system adapter terminate.
6) The benchmark controller creates the evaluation module.
7) The evaluation module loads the results from the evaluation storage. This is done by requesting the results pairs, i.e., the expected result and the result received from the system for a single task, from the storage. The evaluation module uses these pairs to evaluate the systems performance and calculate the KPIs. The results of this evaluation are returned to the benchmark controller before the evaluation module and storage terminate.
8) The benchmark controller adds information for repeating the experiment, e.g., its parameters, to the evaluation results, sends them to the platform controller and terminates.
9) After the benchmark controller has finished its work, the platform controller can add additional information to the result, e.g., the configuration of the hardware, and store the result. After that, a new evaluation could be started.
10) The platform controller sends the URI of the new experiment result to the analysis component. The analysis component reads the evaluation results from the storage, processes them and stores additional information in the storage.

Beside the described orchestration scheme the platform will support other schemes as well. For example, it will be possible to generate all the data in a first step before the task generators start to generate their tasks based on the complete data. In another variant, the task generators will also be enabled to generate a task, wait for the response of the system and then send the subsequent task.

VI. Components

A. Platform Components

The platform has several components that are started and stopped together with the complete platform.

1) Platform Controller: The platform controller is the central component of the HOBBIT platform. Its main role is to coordinate the interaction of other components as needed. This mainly includes handling of requests that come from the front end component, starting and stopping of benchmarks, observing of the health of the cluster and triggering the analysis component.

The platform controller manages a queue that contains user configured experiments that are to be executed in the future. The execution order of experiment configurations is determined using the following features:

- The resources needed for an experiment. For example, if an experiment only needs half of the available cluster nodes, the remaining nodes might be used for a different experiment that fits into the remaining resources.
- The time at which they have been configured by the user (following the first-in-first-out principle).
- Whether the experiment is part of a scheduled challenge.

The platform controller guarantees that experiments can be schedule to be executed at a certain data, for example for benchmarking challenges.

The internal status of the platform controller is stored in a database. This enables shutting down or restarting the controller without losing its current status, e.g., benchmarks that have been configured and scheduled inside the queue.

The platform controller interacts with the front end in the following situations.

1) The front end shows a configuration screen to the user with which the user can start a benchmark with a particular system. The platform controller has to give the following information to the front end:
   - available benchmarks,
   - available systems that fit a chosen benchmark and
   - benchmark parameters of the chosen benchmark.
2) The front end sends a configuration for an experiment comprising
   - the name of a benchmark,
   - its parameter values,
   - the name of the system adapter and
   - a notification method, e.g., an e-mail address of the user, with which the user wants to be informed about his experiment.
3) The platform controller should be able to send the status of its experiment queue to the front end, if the user wants to see the position of its experiment in the queue.

The platform controller will use features of Docker Swarm to observe the status of the cluster that is used to execute the experiments.[4] If one of the nodes drops out of the cluster, the comparability between single experiments is not given. Thus, the platform controller needs to be aware of the number of working nodes that are available for the experiment. If there is no running experiment and the queue is not empty, the platform controller should initiate the execution of an experiment and the it observes the state of a started experiment. If the experiment takes more time than a configured maximum, the platform controller terminates the benchmark components as well as the system that belong to it.

The platform controller is the only component that has direct access to the Docker daemon. If another component would like to start a docker container, it has to send a request to the platform controller containing the image name and parameters. Thus, the platform controller offers a central control of commands that are sent to the docker daemon. This architectural choice increases the security of the system.

2) Storage: The storage component contains the experiment results. It comprises two containers—a triple store that uses the HOBBIT ontology to describe the results and a Java program that handles the communication between the message bus and the triple store. The storage component offers a public SPARQL Endpoint with read-only access. Additionally, the front end needs to be able to load results to present them to the user while the analysis component needs the results for deeper analysis. In addition, this component offers write access to the benchmark component for storing experiment results. Additionally, the analysis component needs to be able to store additional results of the result analysis. To make sure that the write access is limited to these two components the storage component offers a secured communication using a generated key as described in section VIII.

3) Ontology: The experiment results as well as the results of the analysis component are stored as RDF triples in the storage component. Hence, an ontology is needed to define how the data is to be formulated in RDF. The following requirements are to be fulfilled by the ontology:

- It has to offer classes and properties to store the configuration and the results of an experiment
  - URIs will be assigned to the single benchmarks and systems.
  - It will be able to describe the KPIs.
  - The cluster and its configuration with which the experiment has been carried out has to be described.
  - The benchmark configuration needs to be stored.
- The ontology will support the work of the analysis component.

[4]Since version 1.12 Docker Swarm has embedded health checks and failover policies. https://docs.docker.com/engine/swarm/

  - It will be possible to define the features of a system and a benchmark.
  - The KPIs will be described with their type.

4) Front end: The front end component handles the interaction with the user. It contains a user management that allows different roles for authenticated users as well as a guest role for unauthenticated users and offers different functionalities to the different user groups. A guest is only allowed to read the results of experiments and analysis. Authenticated users have additional rights ranging from the start of an experiment to organizing challenges, i.e., define experiments with a certain date at which they will be executed. Additionally, they can upload new system and system adapter images as well as images for benchmarks. These images are stored in a local Docker repository.

5) Analysis: This component is triggered after an experiment has been carried out successfully. Its task is to enhance the benchmark results by combining them with the features of the benchmarked system and the data or task generators. These combination can lead to additional insights, e.g., strengths and weaknesses of a certain system.

While the component uses the results of a benchmark it is modelled independently from any benchmark implementation. It uses the knowledge gained from the system and benchmark described using the HOBBIT ontology and decides which analysis it can execute.

6) Message Bus: This component contains the message bus system. We use three different communication patterns.

First, we use labelled queues to forward data, e.g., the data generated by the mimicking algorithm from several data generators to several task generators.

The second pattern works like remote procedure calls. The queue has one single receiving consumer that executes a command, e.g., a SPARQL query, and sends a response containing the result.

Third, we use a central broadcasting queue (`hobbit.command`), i.e., every component connected to this queue receives all messages send by one of the other connected components. This queue is used to connect the loosely coupled components and orchestrate their activities. Since the platform should be able to support the execution of more than one experiment in parallel, we will use a simple addressing scheme to be able to distinguish between platform components and benchmark components of different experiments. Every experiment gets a unique HOBBIT ID that is added at the beginning of a message. Based on this ID a benchmark component can decide whether it has to process the message.

7) Logging: The logging comprises of three single components—Logstash[5], Elasticsearch[6] and Kibana[7]. While Logstash collects the log messages from the single components, Elasticsearch is used to store them inside a fulltext index. Kibana offers the front end for accessing this index.

[5]https://www.elastic.co/de/products/logstash

[6]https://www.elastic.co/de/products/elasticsearch

[7]https://www.elastic.co/de/products/kibana

B. Benchmark Components

These components are part of the benchmark. They are instantiated for a particular experiment and should have been destroyed when the experiment ends.

1) Benchmark Controller: The benchmark controller is the central component of an evaluation. It creates and controls the data generators, task generators, evaluation-storage and evaluation-module.

Workflow:

1) It is created by the platform controller with the following parameters
   - the HOBBIT ID of the experiment,
   - benchmark specific parameters and
   - the Docker container ID of the system adapter.
2) The benchmark controller initializes itself and connects to the `hobbit.command` queue.
3) It waits for the system to send the start signal.
4) It sends a request to the platform controller to create the data and task generators as well as the evaluation storage.It waits for the single components to report that they are ready.
5) It sends a start signal to the created benchmark components.
6) It waits for all data generators to end before sending a signal to the task generators that all data generators have terminated.
7) It waits for all task generators to end before sending a signal to the system adapter that all task generators have terminated.
8) It waits for the system adapter to end before it sends a request to the platform controller to create the evaluation module.
9) It waits for the evaluation module and evaluation storage to finish and sends the results received from the evaluation module to the platform controller.

2) Data generator: The data generator contains an algorithm implementation that is able to generate the data needed for the evaluation. For benchmarking a system using large volumes of data, the data generator can be instantiated several times. Each instance receives an ID that is used to seed the generator so as to generate different data. Typically, a data generator is created by the benchmark controller with the HOBBIT ID of the experiment, the ID of this particular generator instance as well as the number of generators and benchmark specific parameters (incl., e.g., seeds). It initializes itself, sends the ready signal to the benchmark controller and waits for the start signal. It generates data based on the given parameters and sends it to the task generators as well as the system adapter and terminates when the required data has been created.

3) Task generator: The task generator gets the data from the data generator, generates tasks that can be identified with an ID and are sent to the system adapter. The expected response is sent to the evaluation storage.

Typically, a task generator is created by the benchmark controller with the HOBBIT ID of the experiment, the ID of this particular generator instance as well as the number of generators and benchmark specific parameters. It initializes itself, sends the ready signal to the benchmark controller and waits for the start signal. It generates tasks by reading incoming data from the data generators and generating a task and the expected solution based on this data and sending the task to the system adapter and the expected solution to the evaluation storage. If it receives the signal that all data generators terminated it consumes all data that is still available and terminates.

4) Evaluation storage: The evaluation storage is a component that stores the gold standard results as well as the responses of the benchmarked system during the computation phase. During the evaluation phase it sends this data to the evaluation module. Internally, the component is based on a key-value store and a small java program that handles the communication with other components. The ID of the task is used as key while the value comprises the expected result as well as the result calculated by the benchmarked system.

The evaluation storage is created by the benchmark controller with the HOBBIT ID of the experiment. It initializes itself and sends the ready signal to the benchmark. It stores incoming system responses and expected results and adds a timestamp to the results received from the system adapter to enable time measurements. After the evaluation module has been started, the evaluation storage will receive a request to iterate the result pairs. Every request will be answered with the next result pair. If a request from the evaluation module is received but can not be answered because all result pairs have been sent, an empty response is send and the evaluation storage terminates.

5) Evaluation Module: The evaluation module evaluates the result generated by the benchmarked system. It is created by the benchmark controller with the HOBBIT ID of the experiment. It initializes itself before requesting result pairs from the evaluation storage. It evaluates them until an empty response is received. It summarizes the evaluation results, sends them to the benchmark controller and terminates.

C. Benchmarked System Components

The system adapter serves as a proxy translating messages from the HOBBIT platform to the system to be benchmarked and vice versa. It is created by the platform controller with the HOBBIT ID of the experiment. It initializes itself, starts the system container(s) and waits until they are ready. After that, it sends the ready signal to the platform controller. It receives incoming data and tasks, forwards them to the system and sends its responses to the evaluation storage. It terminates after it received a command indicating that all tasks have been finished.

VII. Safety

The safety of the platform, i.e., its running processes and its data, can be endangered on two levels—the failure of a single software component or the failure of a hardware node. We address this potential problem by 1) defining which data has to be secured and 2) defining actions that have to be performed in case of failures.

The hobbit platform comprises three data sources that need to be stored safely:

- the triple store,
- the store of the user management and
- the store containing the queue of the platform controller.

For these stores, we rely on established strategies, i.e., storing the data on a RAID and performing regular backups.

The safety of processes of a platform component is achieved by using an automatic restart mechanism offered by Docker. If a benchmark component or the system adapter crashes the experiment is aborted.

VIII. Security

One central feature of the platform is the ability to upload systems and benchmarks. However, the execution of third party containers can be risky since an attacker could upload harmful software code. The platform contains several critical components that have to be protected against attacks from outside as well as attacks from inside, i.e., uploaded components.

A goal of a typical attacker is to gain control of a hardware node. However, all software components are executed in Docker containers. We will restrict the rights of these containers to exclude the execution of malicious commands using AppArmor security profiles.[8] Thus, an attacker might have the ability to execute code inside an uploaded container but the process is not able to influence the underlying operating system.

Another attack might be to get direct access to the Docker daemon to download and execute additional containers with extended rights. This is prohibited by defining user credentials for the Docker daemon. A container that has to start another container, e.g., the benchmark controller that has to create the data and task generators, is not able to communicate directly with the Docker daemon but has to send a request to the platform controller. Thus, the platform controller can decide whether a Docker container should be started or not and which volumes can be mounted.

An additional goal of an attacker might be the manipulation of data that is stored either in the central triple store

[8]https://docs.docker.com/engine/security/apparmor/

of the platform or in the evaluation storage. Such an attack will be prevented by the following steps.

- Allow the definition of user credentials for the triple store in a non-public properties file making sure that only the java program of the storage component can write to the triple store.
- In a similar way the evaluation storage component should be able to generate new credentials and apply them to its key-value store every time it is executed.
- Secure the communication between the analysis, platform controller and storage components.
- Secure the communication between the task generator and the evaluation storage to make sure that an uploaded system adapter is not able to insert his own gold standard answers.

There are several ways to secure the message-bus-based communication between two components. First, the endpoints can secure every single message by using either a symmetric encryption algorithm to encrypt the complete message or an asymmetric encryption to sign the single messages. However, in both cases every message would cause an additional overhead because of encryption and decryption steps. A second approach is to hide the queue name by adding a random number to the name during its creation. Since a RabbitMQ client is not able to get the list of existing queues, an attacker would have to guess the name of the queue. In this case the random number serves as a key that must be known to connect to the queue.

An attacker could try to get access to the user management to gain additional rights. Since we aim to reuse an already established solution for our user management, choosing a secure solution will prevent these attacks.

IX. Evaluation

We evaluated our architecture by implementing a prototype and showing that we can benchmark the efficiency and effectiveness of a system for a given task. For our implementation we used the problem of spotting named entities from a given text and linking them to a given knowledge base. We benchmarked FOX [31]—a Named Entity Recognition and Linking tool as well as the following named entity recognition tools:

- the Ottawa Baseline Information Extraction (Balie) [32],
- the Illinois Named Entity Tagger (Illinois) [33],
- the Apache OpenNLP Name Finder (OpenNLP) [34],
- and the Stanford Named Entity Recognizer (Stanford) [35].

The entities that were found in the text by any of the tools were linked to a given knowledge base using AGDISTIS [36]. In our experiment, we used DBpedia 2015[9] as the reference knowledge base.

[9]http://dbpedia.org

Table I
The effectiveness of the system configurations (Macro measures).

System	Precision	Recall	F1-measure
FOX	**0.515**	**0.310**	**0.351**
Balie	0.369	0.230	0.249
Illinois	0.500	0.288	0.327
OpenNLP	0.442	0.241	0.285
Stanford	0.486	0.303	0.335

We used the 128 documents of the existing Reuters-128 dataset [37]. These documents already contained annotations for all named entities that could be found in the text and were sent to the task generator.[10] The task generator sent the documents with all its annotations to the evaluation storage and only the text of the document without the annotations to the system adapter. The evaluation module was based on the evaluation that is used in [4] and measured the run time for single documents as well as the quality of the result in terms of macro-precision[11], recall[12] and F1-measure[13].

We used 4 data and 4 task generators for our benchmark. The data generators started by sending a single document every second leading to a workload of 4 documents per second. Every time all 128 documents had been sent, the duration of the break between two documents was reduced to $\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, 0\}$ seconds leading to an increasing workload of $\{8, 16, 32, 64, 512\}$ documents per second. Since none of the tools uses caching, we did not restart it before repeating the documents. We configured all tools to run 10 parallel instances that get their tasks from a central built-in scheduler.

The experiment was repeated 5 times with each tool. While Table I shows that FOX achieves the highest precision, recall and F-measure, Figure 3 shows that it also has the highest run times. This was to be expected as FOX combines the other four tools using ensemble learning. In addition, the comparison clearly shows that Balie has much higher run times than the other three tools. Given that Balie also has the lowest F1-score (see Table I), it can be argued that Balie might be removed from FOX to increase its efficiency.[14] Most, importantly, these results show that our benchmarking platform is indeed able to run both efficiency and effectiveness experiments (which are the most sought after) for data that abide by the attributes of Big Data.

[10]Note that the documents are not sent to the system adapter since it is not needed for this benchmark.

[11]The number of correct named entities divided by the number of all named entities found by the system.

[12]The number of correct named entities found by the system divided by the number of all named entities that could have been found in the text.

[13]The harmonic mean of precision and recall.

[14]During our experiments, it was not possible to configure FOX to use all tools except Balie to evaluate this configuration as well.

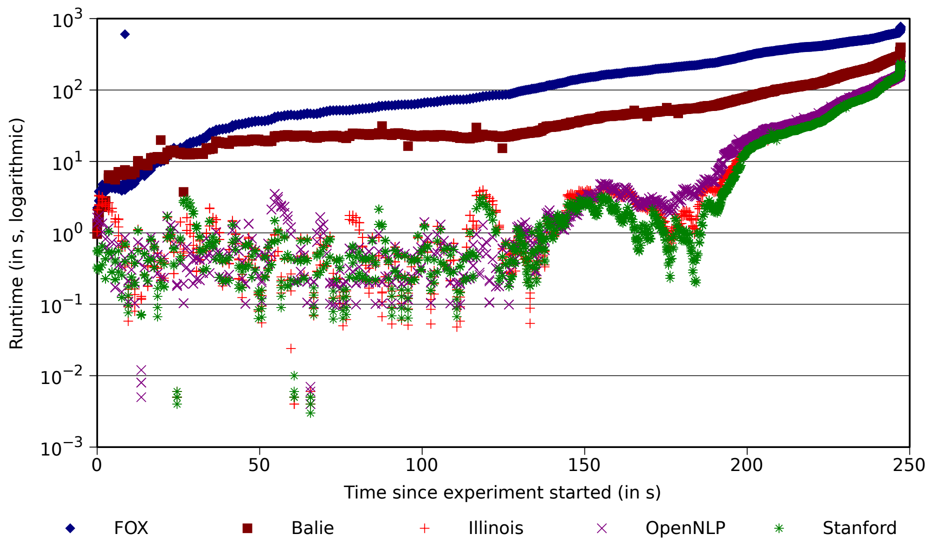

Figure 3. Runtimes of the single tasks sorted by the time at which they have been sent to the system.

X. Conclusion

This paper presents the architecture of the HOBBIT benchmarking platform, which is based on real requirements from industry as well as academia. The platform is modular and considers aspects of scalability, security and data safety. By ensuring that it can be easily distributed at large scale, the HOBBIT platform can be used for stress testing Big Data systems and hence benchmark several facets of Big Data applications. These results are corroborated by our experiments, in which we measured both the scalability and accuracy of state-of-the-art approaches for Named Entity Recognition and Entity Linking. In future work, we aim to make a full-fledged implementation of the platform available as an open-source solution for benchmarking. As the platform is not limited to a particular step of the Linked Data life cycle and can be configured to use virtually any data generator and task generator, it is well suited for benchmarking any step of the Big Linked Data life cycle. We hence aim to unfold it to the reference point for benchmarking Big Linked Data applications.

Acknowledgment

This work has been supported by Eurostars projects DIESEL (E!9367) and QAMEL (E!9725) as well as the European Union's H2020 research and innovation action HOBBIT (GA 688227).

References

[1] J. Gray and C. Levine, "Thousands of debitcredit transactions-per-second: Easy and inexpensive," *arXiv preprint cs/0701161*, 2007.

[2] C. Unger, C. Forascu, V. Lopez, A.-C. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter, "Question answering over linked data (qald-4)," in *Working Notes for CLEF 2014 Conference*, 2014.

[3] G. Tsatsaronis, M. Schroeder, G. Paliouras, Y. Almirantis, I. Androutsopoulos, E. Gaussier, P. Gallinari, T. Artieres, M. R. Alvers, M. Zschunke *et al.*, "Bioasq: A challenge on large-scale biomedical semantic indexing and question answering." in *AAAI fall symposium: Information retrieval and knowledge discovery in biomedical text*. Citeseer, 2012.

[4] R. Usbeck, M. Röder, A.-C. Ngonga Ngomo, C. Baron, A. Both, M. Brümmer, D. Ceccarelli, M. Cornolti, D. Cherix, B. Eickmann, P. Ferragina, C. Lemke, A. Moro, R. Navigli, F. Piccinno, G. Rizzo, H. Sack, R. Speck, R. Troncy, J. Waitelonis, and L. Wesemann, "GERBIL – general entity annotation benchmark framework," in *24th WWW conference*, 2015.

[5] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A Benchmark for OWL Knowledge Base Systems," *J. Web Sem.*, vol. 3, no. 2-3, 2005.

[6] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel, "SP2Bench: A SPARQL performance benchmark." in *International Conference on Data Engineering (ICDE)*. IEEE, 2009.

[7] C. Bizer and A. Schultz, "The Berlin SPARQL Benchmark." *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 2, 2009.

[8] Y. Zhang, M.-D. Pham, O. Corcho, and J.-P. Calbimonte, "SRBench: A Streaming RDF/SPARQL Benchmark." in *International Semantic Web Conference (ISWC)*, ser. Lecture Notes in Computer Science, vol. 7649. Springer, 2012.

[9] M. Przyjaciel-Zablocki, A. Schätzle, T. Hornung, and I. Taxidou, "Towards a sparql 1.1 feature benchmark on real-world social network data," in *Proceedings of the First International Workshop on Benchmarking RDF Systems*, 2013.

[10] T. Tarasova and M. Marx, "Parlbench: a sparql benchmark for electronic publishing applications," in *The Semantic Web: ESWC 2013 Satellite Events*. Springer, 2013.

[11] G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee, "Diversified stress testing of rdf data management systems," in *International Semantic Web Conference (ISWC)*, 2014.

[12] M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo, "DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data," in *International Semantic Web Conference*, 2011.

[13] ——, "Usage-Centric Benchmarking of RDF Triple Stores," in *AAAI*, 2012.

[14] M. Saleem, Q. Mehmood, and A.-C. Ngonga Ngomo, "FEASIBLE: A featured-based sparql benchmark generation framework," in *ISWC*, 2015.

[15] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran, "FedBench: A Benchmark Suite for Federated Semantic Data Query Processing," in *ISWC*, 2011.

[16] M. Nentwig, M. Hartung, A.-C. Ngonga Ngomo, and E. Rahm, "A survey of current link discovery frameworks," *Semantic Web*, no. Preprint, 2015.

[17] E. Daskalaki, G. Flouris, I. Fundulaki, and T. Saveta, "Instance matching benchmarks in the era of linked data," *Web Semantics: Science, Services and Agents on the World Wide Web*, 2016.

[18] B. M. Sundheim, "Tipster/MUC-5: Information extraction system evaluation," in *Proceedings of the 5th Conference on Message Understanding*, 1993.

[19] E. F. Tjong Kim Sang and F. De Meulder, "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition," in *Proceedings of CoNLL-2003*, 2003.

[20] G. R. Doddington, A. Mitchell, M. A. Przybocki, L. A. Ramshaw, S. Strassel, and R. M. Weischedel, "The automatic content extraction (ACE) program-tasks, data, and evaluation." in *LREC*, 2004.

[21] P. McNamee and H. T. Dang, "Overview of the tac 2009 knowledge base population track," in *Text Analysis Conference (TAC)*, vol. 17, 2009.

[22] A. Kilgarriff, "Senseval: An exercise in evaluating word sense disambiguation programs," *1st LREC*, 1998.

[23] M. Rowe, M. Stankovic, and A.-S. Dadzie, Eds., *Proceedings, 4th Workshop on Making Sense of Microposts (#Microposts2014): Big things come in small packages, Seoul, Korea, 7th April 2014*, 2014.

[24] A. G. Nuzzolese, A. L. Gentile, V. Presutti, A. Gangemi, D. Garigliotti, and R. Navigli, "Open knowledge extraction challenge," in *Semantic Web Evaluation Challenge*. Springer, 2015.

[25] D. Carmel, M.-W. Chang, E. Gabrilovich, B.-J. P. Hsu, and K. Wang, "ERD 2014: Entity recognition and disambiguation challenge," *SIGIR Forum*, 2014.

[26] M. Cornolti, P. Ferragina, and M. Ciaramita, "A framework for benchmarking entity-annotation systems," in *22nd World Wide Web Conference*, 2013.

[27] R. Usbeck, M. Röder, and A.-C. Ngonga Ngomo, "Evaluating entity annotators using gerbil," in *ESWC*, 2015.

[28] E. M. Voorhees *et al.*, "The trec-8 question answering track report." in *Trec*, vol. 99, 1999.

[29] C. Unger, C. Forascu, V. Lopez, A. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter, "Question answering over linked data (QALD-5)," in *CLEF*, 2015.

[30] I. Fundulaki, "Deliverable 1.2.1: Requirements specification from the community," 2016. [Online]. Available: http://project-hobbit.eu/about/deliverables/

[31] R. Speck and A.-C. Ngonga Ngomo, "Ensemble learning for named entity recognition," in *ISWC*, 2014.

[32] D. Nadeau, "Balie–baseline information extraction: Multilingual information extraction from text with machine learning and natural language techniques," Technical report, University of Ottawa, Tech. Rep., 2005.

[33] L. Ratinov and D. Roth, "Design challenges and misconceptions in named entity recognition," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, ser. CoNLL '09. ACL, 2009.

[34] J. Baldridge, "The opennlp project," 2005.

[35] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *ACL*, 2005.

[36] R. Usbeck, A.-C. N. Ngomo, M. Röder, D. Gerber, S. Coelho, S. Auer, and A. Both, "AGDISTIS - Graph-Based Disambiguation of Named Entities Using Linked Data," in *ISWC*, 2014.

[37] M. Röder, R. Usbeck, S. Hellmann, D. Gerber, and A. Both, "N3 - a collection of datasets for named entity recognition and disambiguation in the nlp interchange format," in *LREC*, 2014.