ORCA – a Benchmark for Data Web Crawlers

Michael Röder[1,2], Geraldo de Souza[1], Denis Kuchelev[1], Abdelmoneim Amer Desouki[1], and Axel-Cyrille Ngonga Ngomo[1,2]

[1] DICE group, Department of Computer Science, Paderborn University, Germany
michael.roeder|axel.ngonga@upb.de
[2] Institute for Applied Informatics, Leipzig, Germany

Abstract. The number of RDF knowledge graphs available on the Web grows constantly. Gathering these graphs at large scale for downstream applications hence requires the use of crawlers. Although Data Web crawlers exist, and general Web crawlers could be adapted to focus on the Data Web, there is currently no benchmark to fairly evaluate their performance. Our work closes this gap by presenting the ORCA benchmark. ORCA generates a synthetic Data Web, which is decoupled from the original Web and enables a fair and repeatable comparison of Data Web crawlers. Our evaluations show that ORCA can be used to reveal the different advantages and disadvantages of existing crawlers. The benchmark is open-source and available at https://github.com/dice-group/orca.

1 Introduction

The number of RDF knowledge graphs (KGs) available on the Web has grown continuously over recent years.[3] These KGs are provided through means ranging from SPARQL endpoints over simple dump files to information embedded in HTML pages. Data Web crawlers are employed to discover and make use of such KGs [14]. The efficiency and effectiveness of such crawlers are typically evaluated by crawling the Web for a set amount of time while measuring different performance indicators such as the number of requests the crawler performed [10,14]. While this kind of experiment can be performed for a crawler at a given point in time, the experiments are virtually impossible to repeat and thus, hard to compare with similar experiments. This is due to several factors, including primarily the fact that the Web is an ever-changing, evolving network of single, partly unreliable nodes. Another influence is the geographical location of the machine on which the crawler is executed. For example, geo-blocking can have an influence on the shape of the crawled network. Executing the same crawler on the same hardware might also lead to different evaluation results when different internet service providers offering different connections with different bandwidths are used. In addition, the ground truth is not known in such experiments. Since the content of the complete Web is unknown, it is hard to measure the effectiveness of a crawler, i.e., its ability to retrieve relevant data.

To overcome these limitations, we propose ORCA—a benchmark for Web Data Crawlers. The basic idea of ORCA is to alleviate the limitations of current benchmarking

[3] For an example, see https://lod-cloud.net/.

approaches by (1) generating a synthetic Data Web and (2) comparing the performance of crawlers within this controlled environment. The generation of the synthetic Web is based on statistics gathered from a sample of the real Data Web. The deterministic generation process implemented by our approach ensures that crawlers are benchmarked in a repeatable and comparable way.

This paper has four main contributions. First, we provide an approach to generate a synthetic Data Web. Based on this generator, we present our second contribution, ORCA, the first extensible FAIR benchmark for Data Web crawlers, which can measure the efficiency and effectiveness of crawlers in a comparable and repeatable way. Third, we are the first to directly compare two Data Web crawlers in a repeatable setup. Fourth, we show that ORCA can be used to evaluate the politeness of a crawler, i.e., whether it abides by the Robots Exclusion Protocol [16].

The rest of the paper is organised as follows. Section 2 presents related work while Section 3 defines prerequesites. In Section 4, we describe the approach and its implementation. The experiments and their results are presented in Section 5 and discussed in Section 6. We conclude the paper with Section 7.

2 Related Work

We separate the related work into two parts. First, we present related publications regarding crawlers and their evaluations. Note that due to the limited space, we mainly focus on Data Web crawlers. Second, we present a brief overview of related work, presenting statistics regarding the Semantic Web.

2.1 Crawlers and their Evaluation

The Mercator Web Crawler [10] is an example of a general Web crawler. The authors describe the major components of a scalable Web crawler and discuss design alternatives. The evaluation of the crawler comprises an 8-day run, which has been compared to similar runs of the Google and Internet Archive crawlers. As performance metrics, the number of HTTP requests performed in a certain time period, and the download rate (in both documents per second and bytes per second) are used. Additionally, further analysis is undertaken regarding the received HTTP status codes, different content types of the downloaded data, and which parts of the crawler the most CPU cycles are spent. This publication can be seen as an example of a classical crawler evaluation, which comes with the drawbacks explained in the previous Section.

A crawler focusing on structured data is presented in [9]. It comprises a 5-step pipeline and converts structured data formats like XHTML or RSS into RDF. The evaluation is based on experiments in which the authors crawl 100k randomly selected URIs. To the best of our knowledge, the crawler is not available as open source project. In [14], the authors present LDSpider—a crawler for the Web of Linked Data. It is described in detail in Section 5.2. In [11,12], a distributed crawler is described, which is used to index resources for the Semantic Web Search Engine. In the evaluation, different configurations of the crawler—different numbers of threads as well as machines on which the crawler has been deployed—are compared, based on the time the crawler needs to

crawl a given amount of seed URIs. To the best of our knowledge, the crawler is not available as open-source project. In [4], the authors present the LOD Laundromat—an approach to download, parse, clean, analyse and republish RDF datasets. The tool relies on a given list of seed URLs and comes with a robust parsing algorithm for various RDF serialisations. In [7], the authors use the LOD Laundromat to provide a dump file comprising 650K datasets and more than 28 billion triples.

Apache Nutch is an open-source Web crawler.[4] However, the only available plugin for processing RDF stems from 2007, relies on an out-dated crawler version and was not working during our evaluation.[5]

2.2 The Data Web

There are several publications analysing the Web of data that are relevant for our work, since we use their insights to generate a synthetic data Web. The Linked Open Data (LOD) Cloud diagram project periodically generates diagrams representing the LOD Cloud and has grown from 12 datasets in 2007 to more than 1200 datasets in 2019.[6] These datasets are entered manually, require a minimum size and must be connected to at least one other dataset in the diagram.

Other approaches for analysing the Data Web are based on the automatic gathering of datasets. LODStats [3,6] collect statistical data about more than 9 000 RDF datasets gathered from a dataset catalogue.[7] In a similar way, [19] use the LDSpider crawler [14] to crawl datasets in the Web. In [13], the authors gather and analyse 3.985 million open RDF documents from 778 different domains regarding their conformity to Linked Data best practices. The authors of [17] compare different methods to identify SPARQL endpoints in the Web and suggest that most SPARQL endpoints can be found using the dataset catalogue. [19] confirms this finding by pointing out that only 14.69% of the crawled datasets provide VoID metadata. In [5], the authors analyse the adoption of the different technologies, i.e., RDFa [1], Microdata [15] and Microformats and crawl 3 billion HTML pages to this end. None of these works targets a benchmark for crawlers. We address this research gap with the work presented subsequently.

3 Preliminaries

Data Web Crawler. Throughout the rest of the paper, we model a crawler as a program that is able to (1) download Web resources, (2) extract information from these resources and (3) identify the addresses of other Web resources within the extracted information. It will use these (potentially previously unknown) addresses to start with step 1 again in an autonomous way. A Data Web crawler is a crawler which extracts RDF triples from the given Web resources. Note that this definition excludes programs like the LOD Laundromat [4], which download and parse a given list of Web resources without performing the third step.

[4] http://nutch.apache.org/
[5] https://issues.apache.org/jira/browse/NUTCH-460
[6] https://lod-cloud.net/
[7] The dataset catalogue is http://thedatahub.org.

Crawlable Graph. Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a directed graph. Let $\mathbb{V} = \{v_1, v_2, \ldots\}$ be the set of nodes of the graph. Let $\mathbb{E} = \{e_1, e_2, \ldots\}$ be the set of directed edges where each edge $e_i = (v_j, v_k)$ is represented as a pair of nodes representing the source node v_j and the target node v_k of the edge. We call a graph $\mathbb{G}$ *crawlable w.r.t.* S iff, starting from a given non-empty set of seed nodes $S \subseteq \mathbb{V}$, it is possible to reach all other nodes of the graph in a finite number of steps by traversing the edges of the graph following their direction. A special case of crawlable graphs are graphs which are crawlable w.r.t. a singleton $S = \{v_\epsilon\}$. For such graphs, we will call the node v_ϵ their *entrance node*.

Data Web Analysis. The Data Web comprises servers of varying complexity. The types of nodes in this portion of the Web include simple file servers offering their data as dump files, Web servers able to dereference single RDF URIs, and SPARQL endpoints that are able to handle complex queries. We carried out an analysis of the LODStats [3,6] dump from 2016 to collect some statistics to configure our benchmark. Based on this analysis, we define the different types of nodes in the synthetic Data Web that is to be generated and used for the benchmark: (1) **Dump file node.** This node comprises an HTTP server offering the RDF data as a single dump file. In its current implementation, ORCA randomly chooses one of the following RDF serialisations: RDF/XML, Notation 3, N-Triples and Turtle. Additionally, the file might be compressed with one of three available compression algorithms—ZIP, Gzip or bzip2.[8] (2) **Dereferencing node.** This node comprises an HTTP server and answers requests to single RDF resources by sending all triples of its RDF graph that have the requested resource as subject. The server offers all serialisations supported by Apache Jena.[9] When a request is received, the serialisation is chosen based on the HTTP `Accept` header sent by the crawler. The complete list of serialisations supported by ORCA can be seen in Table 3. (3) **SPARQL endpoint.** This node offers an API, which can be used to query the RDF data using SPARQL via HTTP.[10](4) **CKAN.** CKAN is a dataset catalogue containing meta data about datasets.[11] It offers human-readable HTML pages and an API that can be used to query the catalogue content.

Robots Exclusion Protocol. The Robots Exclusion Protocol allows the definition of rules for bots like crawlers [16]. The draft of the standard defines two rules—`allow` and `disallow`. They allow or disallow the access to a certain path on a domain, respectively. The rules are defined in a `robots.txt` file, which is typically hosted directly under the domain for which the rules have been defined. Although additional rules are not covered by the standard, the standard allows the addition of lines. Some

[8] Details regarding the compressions can be found at https://pkware.cachefly.net/Webdocs/APPNOTE/APPNOTE-6.3.5.TXT, https://www.gnu.org/software/gzip/ and http://sourceware.org/bzip2/, respectively.
[9] https://jena.apache.org/
[10] In its current implementation, ORCA uses Virtuoso instances for this type of node.
[11] https://ckan.org/

domain owners and crawlers make use of a `Crawl-delay` instruction to define how much delay a crawler should have between its requests to this single Web server.[12]

4 Approach

The main idea of ORCA is to ensure the comparable evaluation of crawlers by creating a local, synthetic Data Web. The benchmarked crawler is initialised with a set of seed nodes of this synthetic cloud and asked to crawl the complete cloud. Since the cloud is generated, the benchmark knows exactly which triples are expected to be crawled and can measure the completeness of the crawl and the speed of the crawler. Since the cloud generation is deterministic, a previously used cloud can be recreated for benchmarking another crawler, ensuring that evaluation results are comparable. In the following, we describe the single parts of the benchmark in detail. We begin by explaining the cloud generation in Section 4.1. An overview of the implementation and its details is given in Section 4.2.

4.1 Cloud Generation

Since the synthetically generated Data Web will be used to benchmark a Data Web crawler, we generate it as a crawlable graph w.r.t. a set of seed nodes S as defined in Section 3. The generation of the synthetic Web can be separated into three steps—(1) Generating the single nodes of the cloud, (2) generating the node graph, i.e., the connections between the nodes, and (3) generating the RDF data of the single nodes.

Node Generation. Nodes are generated by virtue of types selected from the list of available types in Section 3. The number of nodes in the synthetic Web and the distribution of node types are user-defined parameters of the benchmark. The node generation process makes sure that at least one node is created for each type with an amount > 0. Formally, let $\Psi = \{\psi_1, \psi_2, \ldots\}$ be the set of node types and $\Psi_u \subseteq \Psi$ be the set of node types to be generated. To ensures that every type occurs at least one, the generation of the first $|\Psi_u|$ nodes of the output list is deterministic and ensures every type in Ψ_u. The remaining types are assigned using a seeded random model based on the user-defined distribution.

Node Graph Generation. In the real-world Data Web, connections between instances of certain node types are unlikely. For example, an open data portal is very likely to point to dump files, SPARQL endpoints or even other open data portals. However, it is very unlikely that it points to a single RDF resource, i.e., to a server which dereferences the URI of the resource. To take into account these situations, we introduce a connectivity matrix. Let $\mathcal{C}$ be a $|\Psi| \times |\Psi|$ matrix with $c_{ij} = 1$ if an edge from ψ_i to ψ_j is allowed and $c_{ij} = 0$ otherwise. The connectivity matrix which will be used throughout

[12] Examples are Bing (https://blogs.bing.com/Webmaster/2012/05/03/to-crawl-or-not-to-crawl-that-is-bingbots-question/) and Yandex (https://yandex.com/support/Webmaster/controlling-robot/robots-txt.html).

Table 1. Connectivity matrix $\mathcal{C}$ used for the experiments.

from \ to	Deref.	Dump file	SPARQL	CKAN
Deref.	1	1	1	1
Dump file	1	1	1	1
SPARQL	1	1	1	1
CKAN	0	1	1	1

the rest of the paper can be seen in Table 1. It can be seen that, for the node types used in the current implementation of ORCA, all connections are allowed, except the example mentioned above.

The algorithm creating the node graph takes the matrix $\mathcal{C}$, the previously created list of typed nodes and the user-configured average node degree as input. It starts with the first $|\Psi_u|$ nodes and creates connections between them. For these initial nodes, all connections allowed in $\mathcal{C}$ are created. This initial graph is extended step-wise by adding the other nodes from the given list of typed nodes. In each step, the next node from the list is added to the graph. The outgoing edges of the new node are added using a weighted sampling over the nodes that are permissible from the new node according to $\mathcal{C}$. Since the Web is known to be a scale-free network, the weights are the in-degrees of the nodes following the Barabási-Albert model for scale-free networks [2]. In the same way, a similar number of connections to the new node are generated.

After generating the node graph, a set of seed nodes S has to be generated to make the graph crawlable as described in Section 3. This search is equivalent to the set cover problem. Hence, searching for a smallest set of seed nodes would be NP-hard. Thus, we use a greedy solution (see Algorithm 1) which takes $\mathbb{V}$ and $\mathbb{E}$ of the generated node graph as input. We starts with defining all nodes as unmarked nodes by adding them to the set V_u (line 2). After that, the first unmarked node is added to S (lines 4 – 6).[13] A simple breadth-first search starts from this node and marks all reachable nodes by adding them to V_m (lines 7 – 14). At the end, V_u is updated. If V_u is not empty, another unmarked node is taken from the set, added to the set of seed nodes and the search is started again. The approach ends when all nodes are reachable.

RDF Data Generation. For the RDF data generation, the benchmark can be configured with three parameters. Firstly, the average number of triples per RDF graph (τ) can be set. Secondly, the way to determine the sizes of the single graphs can be defined. In its current version, ORCA offers a simple approach which statically assigns the given average size to every RDF graph. However, this can be easily changed to use an exponential distribution to generate a realistic diversity of graph sizes. Thirdly, the average node degree of the RDF resources (d) can be set.

Let $\mathcal{G} = (R, P, L, T)$ be an RDF graph. Let $R = \{r_1, r_2, \ldots\}$ be the set of URI resources of the graph and let $P = \{p_1, p_2, \ldots\}$ be the set of properties. Let L be the set of external URI resources, i.e., resources which belong to a different RDF graph, with $R \cap L = \varnothing$. Let $T = \{t_1, t_2, \ldots\}$ be the set of triples of the RDF graph where each triple

[13] The `pop` method returns and removes the first element from the given set.

Algorithm 1: Generation of the set of seeds S

Input : $\mathbb{V}, \mathbb{E}$
Output: S

```
1  S, Q, V_m ← {}
2  V_u ← 𝕍
3  while |V_u| > 0 do
4      v_f ← pop(V_u)
5      Q ← {v_f}
6      S ← S ∪ {v_f}
7      while |Q| > 0 do
8          v_n ← pop(|Q|)
9          V_m ← V_m ∪ {v_n}
10         for e_i ∈ 𝔼 do
11             if (source(e_i) == v_n)&&(target(e_i) ∉ V_m) then
12                 v_t ← target(e_i)
13                 V_m ← V_m ∪ {v_t}
14                 Q ← Q ∪ {v_t}
15     V_u ← 𝕍\V_m
```

has the form $t_j = \{(s_j, p_j, o_j) | s_j \in R, p_j \in P, o_j \in (R \cup L)\}$.[14] T can be separated into two sub sets $T = T_i \cup T_o$. The set of graph internal triples T_i comprises triples with objects $o_j \in R$. In contrast, the set of outgoing triples T_o (a.k.a. link set) contains only triples with external resources as objects ($o_j \in L$). Further, let d be the average node degree of the resources, i.e., the number of triples a resource is part of—either as a subject or as an object.

Like the node graph, each created RDF graph has to be crawlable. For the RDF graphs, we implemented an algorithm based on the Barabási-Albert model for scale-free networks [2]. The algorithm guarantees that all resources within the generated RDF graph can be reached from the first resource it generates. As defined in Section 3, this resource can later on be used as entrance node by all other RDF graphs which have to generate links to this graph. Let τ be the RDF graph size which has been determined based on the chosen parameters. Based on the previously created node graph, the number of outgoing edges $\tau_o = |T_o|$ as well as their objects, i.e., the set of external URI resources L, are known. Algorithm 2 takes $\tau_i = \tau - \tau_o$ together with the average degree d and a generated set of properties P as input to generate an initial version of graph $\mathcal{G}$. The loop (lines 4–13) will add new resources to the graph until the number of necessary triples has been reached. For each new resource r_n, a URI is generated (line 5) before it is connected to the existing resources of the graph. After that, the degree the new resource d_r is drawn from a uniform distribution in the range $[1, 2d]$ (line 3). The d_r resources to which r_n will be connected to are chosen based on their degree, i.e., the higher the degree of a resource, the higher the probability that it will be chosen for a

[14] Note that this simplified definition of an RDF graph omits the existence of literals and blank odes.

Algorithm 2: Initial RDF graph generation

Input : τ_i, P, d
Output: $\mathcal{G}$

1 $E, T_i \leftarrow \{\}$
2 $R \leftarrow \{r_1\}$
3 $d_r \leftarrow$`drawDegree(`d`)`
4 **while** $|T_i| < \tau_i$ **do**
5 $\quad r_n \leftarrow$ `generateResource(`$|R|$`)`
6 $\quad$ **while** $(|T_i| < \tau_i)$&&(`degree(`r_n`)` $< d_r$) **do**
7 $\quad\quad R_c \leftarrow$ `drawFromDegreeDist(`R, T_i`)`
8 $\quad\quad$ **for** $r_c \in R_c$ **do**
9 $\quad\quad\quad$ **if** (`degree(`r_n`)` $== 0$)||(`bernoulli(`$\frac{0.5d_r-1}{d_r-1}$`)`) **then**
10 $\quad\quad\quad\quad T_i \leftarrow T_i \cup \{$`generateTriple(`$r_c$`,draw(`$P$`),`$r_n$`)`$\}$
11 $\quad\quad\quad$ **else**
12 $\quad\quad\quad\quad T_i \leftarrow T_i \cup \{$`generateTriple(`$r_n$`,draw(`$P$`),`$r_c$`)`$\}$
13 $\quad R \leftarrow R \cup \{r_n\}$
14 $\mathcal{G} \leftarrow \{R, P, \varnothing, T_i\}$

Table 2. Templates of resource URIs to refer to an external resource and its dependency on the external node type. {H} = host name; {F} = file format; {N} = resource ID.

Node type	URI template
Dump file	http://{H}/dumpFile{F}#dataset-0-resource-{N}
Dereferencing	http://{H}/dataset-0/resource-{N}
SPARQL	http://{H}:8890/sparql
CKAN	http://{H}:5000/

new connection. The result of this step is the set R_c with $|R_c| = d_r$. For each of these resources, a direction of the newly added triple is chosen. Since the graph needs to be crawlable, the algorithm will choose the first triple to be pointing to the newly resourced node. This ensures that all resources can be reached, starting from the first resource of the graph. For every other triple, the decision is based on a Bernoulli distribution with a probability of $\frac{0.5d_r-1}{d_r-1}$ being a triple that has the new node as an object. This takes into account that the first triple is always added as incoming edge to the newly added node. Hence, the overall probability of an incoming edge as well as for an outgoing edge is 0.5 (line 9). Based on the chosen direction, the new triple is created with a property that is randomly drawn from the property set P (lines 10 and 12).

After the initial version of the RDF graph is generated, the outgoing edges of T_o are created. For each link to another dataset, a triple is generated by drawing a node from the graph as subject, drawing a property from P as predicate and the given external node as object. Both—T_o and L—are added to $\mathcal{G}$ to finish the RDF graph.

URI Generation. Every resource of the generated RDF graphs needs to have a URI. To make sure that a crawler can use the URIs during the crawling process, the URIs of the resources are generated depending on the type of node hosting the RDF dataset. The different URI templates are available in Table 2. All URIs contain the host name (marked with {H} in Table 2). At the moment, the dump file and the dereferencing node have only one single dataset. Therefore, both URI templates contain the string "dataset-0". A numeric ID is attached (marked with {N}) to make each resource URI unique. Additionally, the dump file node URIs contain the file extension representing the format (marked with {F}). This comprises the RDF serialization and the compression (if a compression has been used).

If a resource of the SPARQL node is used in another generated RDF graph (i.e., to create a link to the SPARQL node), the URL of the SPARQL API is used instead of a resorce URI. The resources that are stored within the SPARQL endpoint use the URI template of the dereferencing node. In a similar way, the links to the CKAN nodes are created by pointing to the CKAN's Web interface without any additional information.

4.2 Implementation

Overview. ORCA is a benchmark built upon the HOBBIT benchmarking platform [18].[15] This FAIR benchmarking platform allows Big Linked Data systems to be benchmarked in a distributed environment. It relies on the Docker[16] container technology to encapsulate the single components of the benchmark and the system.

We adapted the suggested design of a benchmark described in [18] to implement ORCA. The benchmark comprises a benchmark controller, data generators, an evaluation module, a triple store and several nodes that form the synthetic Data Web. The benchmark controller is the central control unit of the benchmark. It is created by the HOBBIT platform, receives the configuration defined by the user and manages the other containers that are part of the benchmark. Figure 1 gives an overview of the benchmark components, the data flow and the single steps of the workflow. The workflow itself can be separated into 4 phases—creation, generation, crawling and evaluation. When the benchmark is started, the benchmark controller creates the other containers of the benchmark.[17] During this creation phase, the benchmark controller chooses the types of nodes that will be part of the synthetic Data Web, based on the parameters configured by the user. The Docker images of the chosen node types are started together with an RDF data generator container for each node that will create the data for the node. Additionally, a node data generator, a triple store and the evaluation store are started. The node data generator will generate the node graph. The triple store serves as a sink for the benchmarked Linked Data crawler during the crawling phase while the evaluation module will evaluate the crawled data during the evaluation phase.

After the initial creation, the graph generation phase is started. This phase can be separated into two steps—initial generation and linking. During the first step, each RDF data generator creates an RDF graph for its Web node. In most cases, this is done using

[15] https://github.com/hobbit-project/platform
[16] https://www.docker.com/
[17] The benchmarked crawler is created by the HOBBIT platform as described in [18].

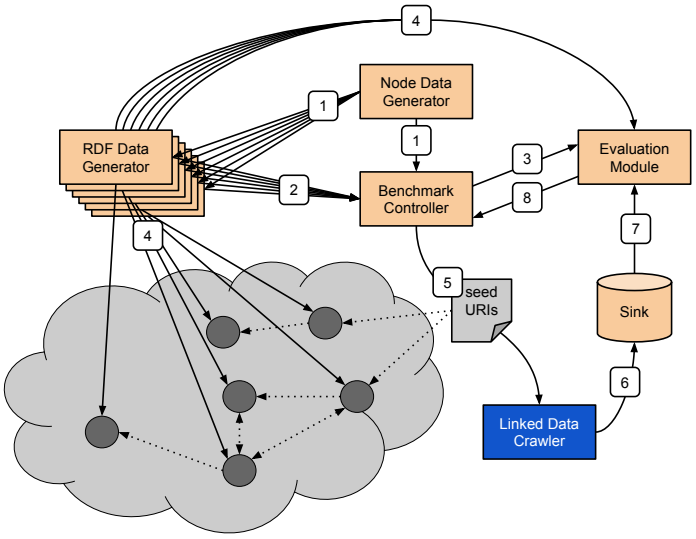

Fig. 1. Overview of the Benchmark components and the flow of data. Orange: Benchmark components; Grey: Synthetic Data Web generated by the benchmark; Dark blue: The benchmarked crawler; Solid arrows: Flow of data; Dotted arrows: Links between RDF datasets; Numbers indicate the order of steps.

the algorithm described in Section 4.1. For data portal nodes, the generation process differs. They solely rely on the information to which other nodes they have to be linked to. For these links, dataset metadata is generated with an outgoing link to these nodes. The node data generator creates the node graph as described in Section 4.1. After this initial generation step, the node graph is sent to the benchmark controller and all RDF data generators (Step 1 in Figure 1). This provides the RDF data generators with the information to which other nodes their RDF graph should be linked. Subsequently, the RDF data generators send their metadata to each other and the benchmark controller (Step 2). This provides the data generators with the necessary data to create links to the entrance nodes of other RDF datasets during the linking step. Additionally, the benchmark controller forwards the collected metadata to the evaluation module as well as the nodes in the cloud (Step 3).[18] At the end of the generation phase, the generated RDF graphs are forwarded to the single nodes and the evaluation module (Step 4). The generation phase ends as soon as all nodes have signalled to the benchmark controller that they have processed the received data.

After the generation phase is finished and the HOBBIT platform signals that the crawler has initialised itself, the benchmark controller submits the seed URIs to the crawler (Step 5). This starts the crawling process in which the crawler must download RDF data from the nodes, process it to extract new, unseen URIs and forward the data to its sink (Step 6) before it crawls the collected, unseen URIs. When the crawler finishes

[18] The submission to the cloud nodes has been omitted in the figure to keep it clean.

Table 3. The RDF serialisations supported by ORCA and the two benchmarked crawlers. (✓) marks serialisations in ORCA that are not used for generating dump nodes for the synthetic Linked Data Web. X marks serialisations listed as processible by a crawler but were not working during our evaluation.

	RDF Serialisations										Comp.		
	RDF/XML	RDF/JSON	Turtle	N-Triples	N-Quads	Notation 3	JSON-LD	TriG	TriX	HDT	ZIP	Gzip	bzip2
ORCA	✓	(✓)	✓	✓	(✓)	✓	(✓)	(✓)	(✓)	–	✓	✓	✓
LDSpider	✓	–	✓	X	✓	✓	✓	–	–	–	–	–	–
Squirrel	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

its crawling—i.e., all given URIs and all URIs found in the crawled RDF data have been crawled—the crawler terminates and the crawling phase ends.

During the evaluation phase, the evaluation module measures the recall of the crawler by checking whether the RDF graphs generated by the data generators can be found in the sink (Step 7). The result of this evaluation is sent to the benchmark controller, which adds further data and results of the benchmarking process (Step 8). This can include data that has been gathered from the single nodes of the cloud, e.g., access times. After this, the final results are forwarded to the HOBBIT platform.

5 Evaluation

For evaluating Data Web crawlers, we use three different experiments. The first experiment uses all available node types to generate the synthetic Data Web and mainly focuses on the recall of the benchmarked crawlers. The second experiment uses a simpler Web to measure efficiency. The last experiment checks whether the crawlers abide by the Robots Exclusion Protocol. Before these three experiments and their results are presented, the hardware setup and the benchmarked crawlers are briefly described.

5.1 Setup

For the experiments, the online instance of HOBBIT is used. It is deployed on a cluster with 3 servers that are solely used by the benchmark and 3 servers that are available for the system. Each of the servers has 16 cores with Hyperthreading and 256 GB RAM.[19]

5.2 Benchmarked crawlers

We identified two working open-source Data Web crawlers—LDSpider and Squirrel. Table 3 shows the RDF serialisations supported by them in comparison to ORCA.

[19] The details of the hardware setup that underlies the HOBBIT platform can be found at https://hobbit-project.github.io/master#hardware-of-the-cluster.

LDSpider [14] is an open-source Linked Data crawler that has been used in several publications to crawl data from the Web [19].[20] Following the documentation, it is able to process triples serialised as RDF/XML, N3 and Turtle. Additionally, it supports Apache Any23, which can be used to parse RDFa, Microdata, Microformats and JSON-LD.[21] The crawler uses multiple threads to process several URIs in parallel. It offers different crawling strategies—a classic breadth-first strategy (BFS), and a load-balancing strategy (LBS). The latter tries to crawl a given number of URIs as fast as possible by making sure that parallel calls to different domains to send many request without overloading the server of a single domain. The crawler can write its results either to files or send them to a triple store.

For our experiments, we dockerized LDSpider and implemented a system adapter to make it compatible with the HOBBIT platform. We created several LDSpider instances with different configurations. LDSpider (T1), (T8), (T16) and (T32) use BFS and 1, 8, 16 or 32 threads, respectively. During our first experiments, we encounter issues with LDSpiders' SPARQL client, which is not storing the crawled data in the provided triple store. To achieve a fair comparison of the crawlers, we extend our system adapter to implement our own SPARQL client, use LDSpiders file sink to get the output of the crawling process, and send file triples to the benchmark sink. These instances of LDSpider are marked with the addition "FS". Additionally, we configured the LDSpider instance (T32,FS,LSB), which makes use of the load-balancing strategy to compare the two strategies offered by the crawler.

Squirrel[22] is an open-source, distributed Linked Data crawler, which uses Docker containers to distribute its components. It crawls resources in a similar way to the LSB strategy of LDSpider, by grouping URIs based on their domain and assigning the created URI sets to its workers. Following the documentation, it supports all RDF serialisations implemented by Apache Jena. Like LDSpider, Squirrel uses Apache Any23 to parse Microdata, Microformats and the Semargl parser for RDFa.[23] Furthermore, it supports the crawling of HDT dump files [8], SPARQL endpoints and open data portals. The latter includes the crawling of CKAN portals and configuration of a scraper that can extract information from HTML pages following predefined rules. For compressed dump files, Squirrel implements a recursive decompression strategy for ZIP, Gzip, bzip2 and tar files. For our experiments, we implement an adapter for the Squirrel crawler. Squirrel (W1), (W3), (W9) and (W18) are instances of the crawler using 1, 3, 9 or 18 worker instances, respectively.[24]

5.3 Data Web Crawling

The first experiment simulates a real-world Data Web and focuses on the effectiveness of the crawlers, i.e., whether they are able to find and store all expected triples. To

[20] https://github.com/ldspider/ldspider
[21] https://any23.apache.org/
[22] https://github.com/dice-group/Squirrel
[23] https://github.com/semarglproject/semargl
[24] Since the HOBBIT cluster assigns 3 servers for the benchmarked crawler, we use multiples of 3 for the number of workers.

Table 4. Results of the Data Web crawling and efficiency experiments.

Crawler	Data Web		Efficiency			
	Micro Recall	**Runtime** (in s)	**Micro Recall**	**Runtime** (in s)	**CPU** (in s)	**RAM** (in GB)
LDSpider (T8)	0.00	63	–	–	–	–
LDSpider (T16)	0.00	109	–	–	–	–
LDSpider (T32)	0.02	394	–	–	–	–
LDSpider (T1,FS)	0.17	769	1.00	1 888	137.2	0.7
LDSpider (T8,FS)	0.17	741	1.00	1 656	68.1	1.2
LDSpider (T16,FS)	0.17	774	1.00	1 686	61.0	1.2
LDSpider (T32,FS)	0.17	804	1.00	1 662	60.3	1.6
LDSpider (T32,FS,LBS)	0.10	104	0.22	215	4.7	1.4
Squirrel (W1)	0.95	25 634	1.00	16 647	1 294.6	3.6
Squirrel (W3)	0.95	9 170	1.00	5 772	1 208.7	8.4
Squirrel (W9)	1.00	3 894	1.00	2 361	662.5	16.6
Squirrel (W18)	1.00	3 158	1.00	1 756	577.3	19.0

this end, we use ORCA to simulate a cloud comprising 100 nodes with 40% dump file nodes, 25% dereferencing nodes, 30% SPARQL nodes and 5% CKAN nodes. The average degree of each node is set to 5 while the usage of `robots.txt` files is disabled. The RDF graph generation is configured to create 1000 triples with an average degree of 6 triples per resource. 30% of the dump file nodes use one of the available compression algorithms for the dump file.

Since LDSpider does not support the crawling of SPARQL endpoints, data catalogues like CKAN, or compressed dump files, we expect LDSpider to achieve a lower Recall than Squirrel. The results of the experiment are listed in Table 4.[25]

5.4 Efficiency evaluation

The second experiment focuses on the efficiency of the crawler implementations. For this purpose, a synthetic Web comprising 100 dereferencing nodes is used since they offer to negotiate the RDF serialisation for transferring the data. This ensures that all crawlers can crawl the complete Web. The average degree of the nodes is set to 20, the degree of a resource in the RDF graph is set to 6 and the usage of `robots.txt` files is disabled. For LDSpider, we use only the FS instances. We expect both crawlers to be able to crawl the complete cloud and that crawler instances with more threads or workers will crawl faster. The results of the experiment are listed in Table 4.[26]

[25] The detailed results can be seen at https://w3id.org/hobbit/experiments#1575687734335, 1575760952793,1575653901154,1575578913785,1575760590382,1575887162440, 1575760746622,1575687773836,1575687826299,1575809346261,1575718283953, 1575718320614.

[26] The detailed results can be seen at https://w3id.org/hobbit/experiments#1574885056935, 1574941226838,1574885190735,1574885221074,1575592539461,1574885407526, 1574885433587,1575544399271,1574885459448.

Table 5. Results for a Data Web with `robots.txt` files including disallow and crawl-delay rules. CDF = Crawl delay fulfilment; RDR = Requested disallowed resources.

Crawler	CDF			RDR	Runtime
	Min	Max	Avg		(in s)
LDSpider (T32,FS,BFS)	0.052	0.122	0.089	0.0	224
LDSpider (T32,FS,LBS)	0.002	0.007	0.004	0.0	43
Squirrel (W18)	0.697	0.704	0.699	0.0	2384

5.5 Robots Exclusion Protocol check

In the third experiment, we evaluate whether the crawlers follow the rules defined in the node's `robots.txt` file. To this end, we configure ORCA to generate a smaller Web comprising 25 dereferencing nodes. Each of the nodes copies 10% of its RDF resources and marks the copies disallowed for crawling using the `disallow` instruction in its `robots.txt` file. Additionally, we define a delay of 10 seconds between two consecutive requests using the `Crawl-delay` instruction in the same file. The average node degree of the nodes is configured as 5 while the average resource degree is set to 6. Table 5 shows the results of this experiment.[27]

6 Discussion

The experiment results show several insights. As expected, none of the instances of LDSpider was able to crawl the complete synthetic Linked Data Web during the first experiment. Apart from the expected reasons previously mentioned (i.e., the missing support for SPARQL, CKAN nodes and compressed dump files), we encountered two additional issues during the experiments. First, as mentioned in Section 5.2, the SPARQL client of LDSpider was not working as expected, i.e., the crawler did not store all the crawled triples in the provided triple store. This leads to the different recall values of the LDSpider instances with and without the "FS" extension. Second, although it tests several content handler modules and configurations, LDSpider does not crawl dump files provided as N-Triples. In comparison, the Squirrel instances crawl the complete cloud, except for some cases where it is not able to crawl all CKAN nodes completely, leading to a micro recall of 0.95.

The second experiment reveals that overall, LDSpider is more time-efficient than Squirrel. In nearly all cases, LDSpider crawls the Web faster and uses less resources than the Squirrel instances. Only with the large amount of 18 workers is Squirrel able to crawl slightly faster. For the size of the graph, the number of threads LDSpider uses does not seem to play a major role when employing the BFS strategy. It could be assumed that the synthetic Web, with 200 nodes, provides only rare situations in which several nodes are crawled by LDSpider in parallel. However, this assumption can be refuted since Squirrel (W18) has a lower runtime. Therefore, the load-balancing strategy of

[27] The detailed results can be seen at https://w3id.org/hobbit/experiments#1575626666061, 1575592492658,1575592510594.

Squirrel seems to allow faster crawling of the Web than the BFS of LDSpider. However, the LDSpider (T32,FS,LBS) instance implementing a similar load-balancing strategy aborts the crawling process very early in all three experiments. Therefore, a clearer comparison of both strategies is not possible.

The third experiment shows that both crawlers follow the Robots Exclusion Protocol. However, Squirrel seems to slow down its requests following the `Crawl-delay` instruction—although it still crawls a little bit too fast—while LDSpider does not take the delay into account.

7 Conclusion

In this paper, we present ORCA—the first extensible FAIR benchmark for Data Web crawlers, which measures the efficiency and effectiveness of crawlers in a comparable and repeatable way. Using ORCA, we compared two Data Web crawlers in a repeatable setup. We showed that ORCA revealed strengths and limitations of both crawlers. Additionally, we showed that ORCA can be used to evaluate the politeness of a crawler, i.e., whether it abides by the Robots Exclusion Protocol. Our approach will be extended in various ways in future work. First, we will include HTML pages with RDFa, Microdata, Microformat or JSON-LD into the benchmark. A similar extension will be the addition of further compression algorithms to the dump nodes (e.g., `tar`), as well as the HDT serialization [8]. The generation step will be further improved by adding literals and blank nodes to the generated RDF KGs and altering the dataset sizes. A simulation of network errors will round up the next version of the benchmark.

References

1. Adida, B., Herman, I., Sporny, M., Birbeck, M.: RDFa 1.1 Primer – third edition. W3C Note, W3C (March 2015), http://www.w3.org/TR/rdfa-primer/
2. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Reviews of modern physics 74(1), 47 (2002)
3. Auer, S., Demter, J., Martin, M., Lehmann, J.: Lodstats – an extensible framework for high-performance dataset analytics. In: ten Teije, A., Völker, J., Handschuh, S., Stuckenschmidt, H., d'Acquin, M., Nikolov, A., Aussenac-Gilles, N., Hernandez, N. (eds.) Knowledge Engineering and Knowledge Management. pp. 353–362. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
4. Beek, W., Rietveld, L., Bazoobandi, H.R., Wielemaker, J., Schlobach, S.: Lod laundromat: A uniform way of publishing other people's dirty data. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., Goble, C. (eds.) The Semantic Web – ISWC 2014. pp. 213–228. Springer International Publishing, Cham (2014)
5. Bizer, C., Eckert, K., Meusel, R., Mühleisen, H., Schuhmacher, M., Völker, J.: Deployment of rdfa, microdata, and microformats on the web–a quantitative analysis. In: International Semantic Web Conference. pp. 17–32. Springer (2013)
6. Ermilov, I., Lehmann, J., Martin, M., Auer, S.: Lodstats: The data web census dataset. In: Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., Gil, Y. (eds.) The Semantic Web – ISWC 2016. pp. 38–46. Springer International Publishing, Cham (2016)

7. Fernández, J.D., Beek, W., Martínez-Prieto, M.A., Arias, M.: Lod-a-lot. In: d'Amato, C., Fernandez, M., Tamma, V., Lecue, F., Cudré-Mauroux, P., Sequeda, J., Lange, C., Heflin, J. (eds.) The Semantic Web – ISWC 2017. pp. 75–83. Springer International Publishing, Cham (2017)
8. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary rdf representation for publication and exchange (hdt). Web Semantics: Science, Services and Agents on the World Wide Web 19, 22–41 (2013), http://www.websemanticsjournal.org/index.php/ps/article/view/328
9. Harth, A., Umbrich, J., Decker, S.: Multicrawler: A pipelined architecture for crawling and indexing semantic web data. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) The Semantic Web - ISWC 2006. pp. 258–271. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
10. Heydon, A., Najork, M.: Mercator: A scalable, extensible web crawler. Word Wide Web (1999)
11. Hogan, A.: Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora (2011), http://aidanhogan.com/docs/thesis/
12. Hogan, A., Harth, A., Umbrich, J., Kinsella, S., Polleres, A., Decker, S.: Searching and browsing linked data with SWSE: The semantic web search engine. Web Semantics: Science, Services and Agents on the World Wide Web 9(4), 365 – 401 (2011), http://www.sciencedirect.com/science/article/pii/S1570826811000473, JWS special issue on Semantic Search
13. Hogan, A., Umbrich, J., Harth, A., Cyganiak, R., Polleres, A., Decker, S.: An empirical survey of linked data conformance. Journal of Web Semantics 14, 14 – 44 (2012), http://www.sciencedirect.com/science/article/pii/S1570826812000352, special Issue on Dealing with the Messiness of the Web of Data
14. Isele, R., Umbrich, J., Bizer, C., Harth, A.: LDspider: An open-source crawling framework for the Web of Linked Data. In: Proceedings of the ISWC 2010 Posters & Demonstrations Track: Collected Abstracts. vol. 658, pp. 29–32. CEUR-WS (2010)
15. Kellogg, G.: Microdata – second edition. W3C Note, W3C (December 2014), https://www.w3.org/TR/microdata-rdf/
16. Koster, M., Illyes, G., Zeller, H., Harvey, L.: Robots Exclusion Protocol. Internet-draft, Internet Engineering Task Force (IETF) (July 2019), https://tools.ietf.org/html/draft-rep-wg-topic-00
17. Paulheim, H., Hertling, S.: Discoverability of SPARQL Endpoints in Linked Open Data. In: Proceedings of the ISWC 2013 Posters & Demonstrations Track. vol. 1035, pp. 245–248. CEUR-WS.org, Aachen, Germany, Germany (2013), http://ceur-ws.org/Vol-1035/iswc2013_poster_17.pdf
18. Röder, M., Kuchelev, D., Ngonga Ngomo, A.C.: HOBBIT: A platform for benchmarking Big Linked Data. Data Science (2019)
19. Schmachtenberg, M., Bizer, C., Paulheim, H.: Adoption of the linked data best practices in different topical domains. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., Goble, C. (eds.) The Semantic Web – ISWC 2014. pp. 245–260. Springer International Publishing (2014)