Easily setting up a local Wikidata SPARQL endpoint using the qEndpoint

Antoine Willerval[1], Dennis Diefenbach[1,2] and Pierre Maret[1,2]

[1]*The QA Company SAS, France*
[2]*CNRS, Laboratoire Hubert Curien UMR 5516, University of Lyon, France*

Abstract
Setting up a local Wikidata SPARQL endpoint is currently technically complex and requires costly hardware resources. In this paper we propose a novel workflow to index Wikidata and setup a local SPARQL endpoint which is easy (simple a docker pull), fast (takes less then 3 hours), cheap (can be executed on a laptop with 16GB of RAM) and efficient (achieves comparable query response times to the official endpoint). We achieve this by exploiting a newly open source SPARQL endpoint called the qEndpoint.

Keywords
Wikidata, SPARQL endpoint, qEndpoint, scalability, HDT, RDF4J

1. Introduction

Wikidata[1] has aggregated, thanks to the contribution of currently more than 23,000 active users[1], knowledge from a large collection of sources over very diverse topics (books, stars, authors, proteins and more). As a consequence Wikidata is one of the largest existing KGs that is publicly available. Its RDF export contains more than 16 billion triples in the integral version and more than 7 billion in the truthy version (where only non-reified statements are included, i.e. statements without qualifiers and references).
As an effect of the size of Wikidata, maintaining the SPARQL query infrastructure is getting increasingly challenging. This is particularly evident by the fact that Wikimedia, who is maintaining Wikidata, announced during Wikidata Conference 2021 a disaster mitigation plan[2] for the query service. This plan was created since Wikimedia cannot currently guarantee that it will be able to provide a stable publicly available query service. This is due to the high traffic and the underlying triple store technology. Currently the query service at https://query.wikidata.org is using as an infrastructure 11 servers, with 128 GB of RAM and 1.6 TB of space each[3]. The only option given by Wikimedia to scale out is to scale vertically which, given the specs of the machine, is challenging and costly. The difficulty to find viable alternatives for a triplestore is

Wikidata'22: Wikidata workshop at ISWC 2022
antoine.willerval@the-qa-company.com (A. Willerval); dennis.diefenbach@the-qa-company.com (D. Diefenbach); pierre.maret@the-qa-company.com (P. Maret)
0000-0003-2186-074X (A. Willerval); 0000-0002-0046-2219 (D. Diefenbach); 0000-0001-5437-7725 (P. Maret)

[1]https://www.wikidata.org/wiki/Wikidata:Statistics
[2]https://www.youtube.com/watch?v=wn2BrQomvFU&t=2h38m48s
[3]https://www.youtube.com/watch?v=wn2BrQomvFU&t=2h34m03s

also shown in a long standing issue[4] to tackle this problem.
One option, suggested also by the community[5], is to offer an easier setup to locally run a query service over Wikidata. This would have mainly two consequences:

- the query service at https://query.wikidata.org would have to handle lower pressure allowing a better service overall for all users;
- users querying the local setup would not be blocked due to the query limits of the public endpoint[6], and should experience less *query timeouts* since all the local resources would be reserved for them.

While this option seems a viable approach it is technically challenging. The instructions provided by Wikimedia[7] to setup a local SPARQL endpoint state that: "it will take about 12 days to get all data imported (into the triple-store), and another 12 days to make the query service catching up the lag". Additionally one needs to add the time to download the dump. Moreover considerable computing resources are necessary[8] for the setup, i.e. a server with 100GB of RAM and 16 CPUs. This explains why this solution is not widespread.
In this paper we present an alternative workflow to setup a local SPARQL endpoint over Wikidata. In this version we restrict to a SPARQL endpoint over the truthy statements, but we plan to extend it to the full dump. The workflow allows to setup a local SPARQL endpoint over Wikidata in 3 hours[9] that contains data that is not older then 24h and that can run on a laptop with 16GB of RAM. This represents a huge improvement over the existing setup. As a side effect we publish truthy dumps for Wikidata that are more up to date than the once published by Wikidata itself. We achieve this by implementing a new indexing workflow and use a recently created open source triple-store, the qEndpoint.

2. Definition

In this article we are using tools and algorithm, in this section we are describing them.

2.1. qEndpoint

The qEndpoint is a new triple-store that aims at making RDF datasets queriable at scale with low hardware requirements. The qEndpoint combines two established Semantic-Web technologies namely RDF4J[2][10] and HDT[3][11]. RDF4J is a popular Java library that includes SPARQL querying capabilities. It provides stores that are recommended only for datasets up to 200 million triples. On the other side, HDT is a compressed RDF dataset format scaling to billions of triples, but it is read-only and it offers only the ability to search triple patterns (without offering

[4]https://phabricator.wikimedia.org/T206560
[5]Discussions were raised for example during the Wikidata Workshop at ISWC in 2020
[6]https://www.mediawiki.org/wiki/Wikidata_Query_Service/User_Manual#Query_limits
[7]https://www.mediawiki.org/wiki/Wikidata_Query_Service/User_Manual#Standalone_service
[8]https://addshore.com/2019/10/your-own-wikidata-query-service-with-no-limits/
[9]assuming a fast internet connection with 10Mb/s
[10]https://rdf4j.org
[11]https://www.rdfhdt.org

full SPARQL support).
By taking inspiration on architectures developed in the relational databases world, the qEndpoint combines the advantages of both HDT and RDF4J. It is highly scalable thanks to HDT and supports updates and SPARQL syntax thanks to RDF4J.
The qEndpoint is available as an open source project at https://github.com/the-qa-company/qEndpoint. It is already used for instance as a part of the infrastructure of the EU Knowledge Graph[4][12] and is the index used by the question answering system QAnswer[5][13][14].

2.2. HDTDiff and HDTCat

The compressed RDF dataset format HDT doesn't supports updates by default, to fix this issue, 2 methods are available, HDTCat[6] and HDTDiff[15].

2.2.1. HDTCat

This method allows to create the UNION of 2 HDT datasets into a new HDT dataset.

$$\text{HDTCat}(hdt_1, hdt_2) = hdt_1 \cup hdt_2$$

2.2.2. HDTDiff

This method allows to remove triples from an HDT dataset to create a new HDT dataset. We need to give it an HDT hdt and a bitmap bm of the size the number of triples of hdt if a bit $i = 1$, the triple at the index i in the HDT should be removed.

$$\text{HDTDiff}(hdt, bm) = \{hdt_i \mid hdt_i \in hdt \wedge bm_i = 0\}$$

3. Wikidata Indexing Workflow

In the following we describe the workflow that we use to index Wikidata. It makes an intensive use of HDT and some of its related functions. The steps are depicted in Figure 1.

1. We take the truthy dump of Wikidata [16] at a given time t_0 and we compress it obtaining HDT_{t_0}.
2. We fetch all the changes since t_0 using the Wikimedia Recent Changes API[17] and for each of the changed entities we download its current turtle representation[18]. By concatenating and compressing all these files we create an HDT_δ.
3. We use HDTDiff (2.2.2) to eliminate from HDT_{t_0} all triples having as a subject one of the entities we fetched from the recent changes API to obtain $HDT_{t_0-minus}$.

[12]https://linkedopendata.eu
[13]https://app.qanswer.ai
[14]https://wikidata.qanswer.ai
[15]https://github.com/rdfhdt/hdt-java/pull/153
[16]https://dumps.wikimedia.org/wikidatawiki/entities/
[17]https://www.mediawiki.org/wiki/API:RecentChanges
[18]for example for the entity Q1 https://www.wikidata.org/wiki/Special:EntityData/Q1.ttl?flavor=simple

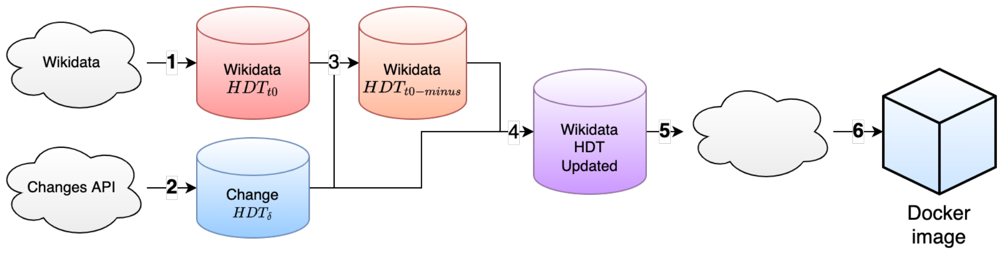

Figure 1: Wikidata changes merge workflow.

Step	Time 1d (h)	Time 3d (h)
Compress t_0	49h	
Fetch HDT_δ	3h	8h
HDTDiff	2h	2h
HDTCat+Index	8h	8h
Total (no compress)	13h	18h

Table 1
Time to run each steps

4. Using HDTCat (2.2.1) we join $HDT_{t_0-minus}$ with HDT_δ to obtain an up-to-date HDT. Finally we compute the corresponding HDT index file.
5. We publish the newly updated Wikidata HDT.
6. We publish a docker image of the qEndpoint that downloads on demand the created Wikidata HDT.

Note that the steps 1 to 5 are carried out by the publisher. The consumer, i.e. the person that wants to set up the SPARQL endpoint, just needs to download the HDT file and its associated index with the qEndpoint. No other loading time is required. In this case we take advantage of HDT that highly compresses RDF datasets while offering triple pattern query capabilities.

The steps 1 to 5 are published in the Wiki-changes repository at https://github.com/the-qa-company/wiki-changes/releases. Step 6 is a docker image that is available at https://hub.docker.com/r/qacompany/qendpoint-wikidata. It bundles the qEndpoint and downloads at start time the latest HDT index over Wikidata available at https://qanswer-svc4.univ-st-etienne.fr.

4. Experiments

First, we show the time that is necessary to generate a fresh index (i.e. steps 1 to 5). To run Wiki-changes, we used a server with 32GB of RAM. The results are presented in the Table 1. We made tests for an 1-day old HDT and a 3-day old HDT. The figures show that once we have the t_n HDT we can get a new up-to-date t_{n+1} HDT in less than 18 hours (13 hours for a 1-day old HDT), allowing a maximum difference of 1 day between our HDT and Wikidata.

Next, let's consider the setup time. On the docker image, the download time of the image of the qEndpoint is only a few minutes and is negligible. Once the container starts, the wikidata

Endpoint	# of Queries	Total time (s)	# of errors	# cases faster
Official	3600	2256	3	110
Local	3600	1164	42	3490

Table 2
Difference during the querying over the local and the official endpoints.

index is downloaded from https://qanswer-svc4.univ-st-etienne.fr. The current size of the HDT Wikidata index is 100GB. With a download speed of 10MB/s the endpoint can be setup in less than 3 hours. We reproduced this number. Note that this is just 3 times slower then downloading the full truthy dump[19] of wikidata in bz2 format which is 32GB. A consequence of this is also that the space requirement for the local endpoint is as little as 100GB on disk.

In the next step we analyze the performance of the SPARQL endpoint using the official query log of Wikidata[7]. For our experiments we take the interval 7 dump[20] which contains more than 82 million queries fired against the endpoint in February and March 2018. We analyzed the first 100.000 queries in the log. 11.378 contain the http://www.bigdata.com/ prefix. Most notably this includes the functionality to get the labels of entities[21]. We exclude these queries from our experiments since they are not covered by the SPARQL 1.1 standard. By excluding this queries we obtain 88.622 queries. From these we exclude the once containing the prefix http://www.wikidata.org/prop/statement/ which indicates a reefied statement which is not included in the truthy dump. We get 84.250 remaining statements. This in particular means that a large part of the query log is answerable by querying the truthy dump only.
We choose the first 3.600 queries of the interval 7 dump which satisfy the above criteria and run them against the official Wikidata SPARQL endpoint and the local docker setup of the qEndpoint (both configured with a timeout of 60 seconds). For the qEndpoint we configured the JVM to use at maximum 6GB of memory[22]. We excluded 3 queries since they were running indefinitely on the qEndpoint even with the timeout set due to a bug[23]. The results are depicted in Figure 2. Each square represents a query and the time difference between the answer time over the qEndpoint and the official endpoint. Squares in green indicate queries that were answered faster on the qEndpoint, and squares in red the queries that were answered faster on the public endpoint. The colors are proportional to the time difference. Overall we can see in the Table 2 that for 3490 cases the qEndpoint gives better results in term of speed compared to the official Wikidata SPARQL endpoint.

5. Conclusion

We have presented an alternative option to set up a local SPARQL endpoint over Wikidata. This solution is quick (3 hours compared to several weeks), low in hardware requirements (only 6GB

[19]https://dumps.wikimedia.org/wikidatawiki/entities/
[20]https://iccl.inf.tu-dresden.de/web/Wikidata_SPARQL_Logs/en
[21]https://en.wikibooks.org/wiki/SPARQL/SERVICE_-_Label
[22]-Xmx setting of the JVM
[23]https://github.com/the-qa-company/qEndpoint/issues/80, https://github.com/eclipse/rdf4j/issues/636

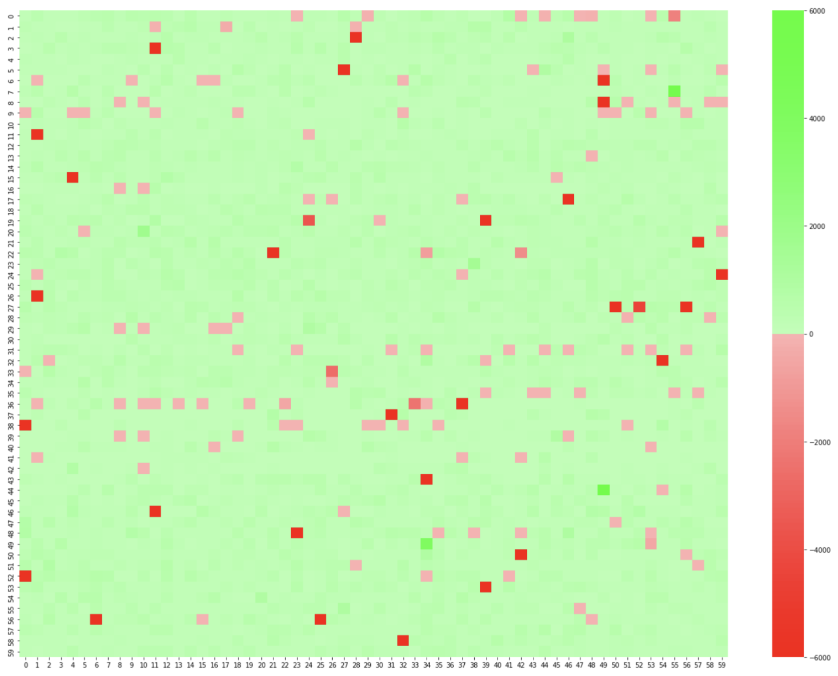
Figure 2: Comparative duration for 3600 queries over the Wikidata SPARQL endpoint and the local endpoint. Green (resp. red) indicates that the local endpoint is faster (resp. slower) than the official endpoint.

of RAM and 100GB of disk compared to 100GB of RAM and TBs of disk) and fast (it can handle a large amount of real world queries faster than the existing endpoint).

We hope that the easy setup of a local Wikidata endpoint will allow to offer new types of services around Wikidata that are not bounded by the limits imposed by the official Wikidata SPARQL service. We hope also that this setup will move part of the workload away from the public Wikidata endpoint so that it can be hosted in a more decentralized manner.

In future we aim at:

- integrating the above workflow and publishing new dumps regularly;
- releasing the same setup for the full Wikidata dump;
- allow a setup that follows live the changes of Wikidata or Wikibase implementations;
- implementing the SPARQL syntax introduced by Blazegraph in order to allow a full compatibility in the current Wikibase ecosystem;
- evaluate the endpoint on a larger part of the query log;
- take more advantage of the internals of the qEndpoint for analytic queries which is an interesting and exciting direction to further exploit the data published in Wikidata.

Acknowledgment: We would like to thank José M. Giménez-García for the discussions we had while building this work. We would like to thank the HDT community for the amazing work they have done, in particular the creators of HDT and more in particular Javier D. Fernández. We would like the RDF4J community that provides a high quality library with amazing code contributions.

References

[1] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Communications of the ACM 57 (2014) 78–85.

[2] J. Broekstra, A. Kampman, F. v. Harmelen, Sesame: A generic architecture for storing and querying rdf and rdf schema, in: International semantic web conference, Springer, 2002, pp. 54–68.

[3] J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, M. Arias, Binary rdf representation for publication and exchange (hdt), Journal of Web Semantics 19 (2013) 22–41.

[4] D. Diefenbach, M. D. Wilde, S. Alipio, Wikibase as an infrastructure for knowledge graphs: The eu knowledge graph, in: International Semantic Web Conference, Springer, 2021, pp. 631–647.

[5] D. Diefenbach, A. Both, K. Singh, P. Maret, Towards a question answering system over the semantic web, Semantic Web 11 (2020) 421–439.

[6] D. Diefenbach, J. M. Giménez-García, Hdtcat: let's make hdt generation scale, in: International Semantic Web Conference, Springer, 2020, pp. 18–33.

[7] S. Malyshev, M. Krötzsch, L. González, J. Gonsior, A. Bielefeldt, Getting the most out of wikidata: semantic technology usage in wikipedia's knowledge graph, in: International Semantic Web Conference, Springer, 2018, pp. 376–394.