# qEndpoint: A Wikidata SPARQL endpoint on commodity hardware

Antoine Willerval
antoine.willerval@etu.univ-lyon1.fr
Lyon 1 University, CNRS Liris
Lyon, France
The QA Company SAS
Saint Etienne, France

Dennis Diefenbach
dennis.diefenbach@the-qa-company.com
The QA Company SAS
Saint Etienne, France

Angela Bonifati
angela.bonifati@univ-lyon1.fr
CNRS, LIRIS UMR 5205, Lyon 1 University
Lyon, France

## ABSTRACT

In this work, we demonstrate how to setup a Wikidata SPARQL endpoint on commodity hardware resources. We achieve this by using a novel triple store called qEndpoint, which uses a read-only partition based on HDT and a write partition based on RDF4J. We show that qEndpoint can index and query the entire Wikidata dump (currently 17 billion triples) on a machine with 600GB SSD, 10 cores and 10GB of RAM, while keeping the query performance comparable with other SPARQL endpoints indexing Wikidata.

In a nutshell, we present the first SPARQL endpoint over Wikidata that can run on commodity hardware while preserving the query run time of existing implementations. Our work goes in the direction of democratizing the access to Wikidata as well as to other large-scale Knowledge Graphs published on the Web. The source code of qEndpoint along with the query workloads are publicly available.

## CCS CONCEPTS

• **Information systems → DBMS engine architectures**.

## KEYWORDS

qEndpoint, HDT, Wikidata, SPARQL

**ACM Reference Format:**

Antoine Willerval, Dennis Diefenbach, and Angela Bonifati. 2023. qEndpoint: A Wikidata SPARQL endpoint on commodity hardware. In *Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion), April 30-May 4, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3543873.3587327

## 1 INTRODUCTION

Wikidata is growing to a central data hub on the Web. This resulted also in a dramatic increase in its size. As of today, Wikidata contains 17 billion triples making it one of the biggest Knowledge Graphs (KGs) on the Web.

The access to this KG is mainly guaranteed by the public Wikidata SPARQL service[1]. To maintain a stable service the Wikimedia Foundation is imposing a limit on the number of queries that can be fired in a certain time interval, as well as a timeout for each SPARQL request. To circumvent these limits, setting up third-party SPARQL endpoints is becoming interesting and necessary both for research and industry.

Due to the size of the dataset, indexing the whole Wikidata dataset is challenging. There is a limited number of successful attempts to index the whole of Wikidata on existing triple stores[5]. In this paper, we demonstrate how to use a novel triple store, called qEndpoint, for indexing Wikidata. Such a triple store is capable of running on commodity hardware, while at the same time achieving query performances similar to server-side solutions. Democratization is one of the reasons why triple stores on low hardware resources are urgently needed. This goes in the direction of designing the Web for an open society [2], where linked open data is not only publicly available, but also accessible by a wide range of people, no matter what their resources are. In the following, we outline the principles underpinning the architecture of qEndpoint (Section 2). We then present an overview of the system (Section 3) by touching upon the following main functionalities: (1) the data loading and the indexing of the whole Wikidata corpus; (2) the query component, showcasing the query performances of queries in the Wikidata query logs and their comparison with existing publicly deployed systems. We then present the demonstration scenarios of qEndpoint (Section 4).

## 2 TRIPLE-STORE ARCHITECTURE

The qEndpoint leverages a novel architecture that is based on two partitions: a read-optimized main partition and a write optimized delta partition (see Figure 1 for an illustration). The first partition is used to store larger parts of the data. The data is read-only and can be highly compressed. This partition uses HDT[6] as an indexing structure allowing high compression of the data coupled with high triple pattern resolution. The delta partition allows reads and writes and is currently using a RDF4J native store. Over time, the delta partition grows and is therefore merged with the main partition with the help of qEndpoint 's merger.

For the purpose of this demo, we focus on the main partition since we consider the querying aspect of Wikidata and thus disregard update operations on the knowledge graph. HDT is known as a good data structure for storing large-scale knowledge graphs and access them at high speed. On the other side, the original implementation suffered from the fact that the generation of HDT had high memory consumption assuming that the whole dataset can be entirely stored

[1] https://query.wikidata.org


*WWW '23 Companion, April 30-May 4, 2023, Austin, TX, USA*

ACM ISBN 978-1-4503-9419-2/23/04…$15.00
https://doi.org/10.1145/3543873.3587327

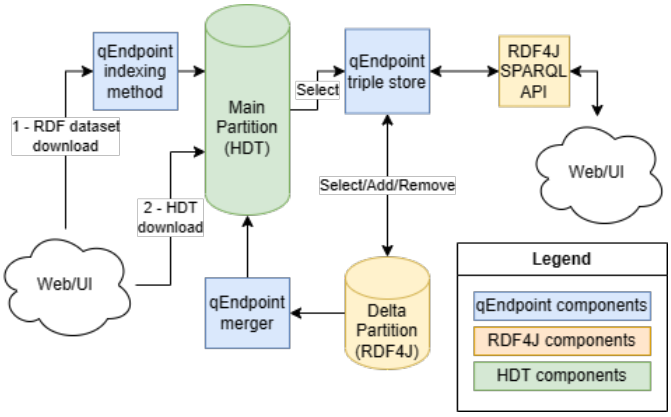


**Figure 1: High level qEndpoint architecture**

in memory. Two solutions have been proposed to circumvent this problem HDT-MR[8][2] and HDTCat[3]. The drawback of the first solution is that it requires a Map Reduce cluster which is an infrastructure not accessible to everyone. The second solution suffers from the fact that it does not scale well with growing size of the dataset. To this end, we have introduced a new indexing strategy that we integrated into the official RDFHDT Java library[3]. It allows to index HDT using the disk instead of the memory, drastically reducing the memory usage and allowing to index bigger datasets. The qEndpoint offers a SPARQL query interface based on YASGUI[10]. A screenshot is shown in Figure 2. The code of the qEndpoint is publicly available at https://github.com/the-qa-company/qEndpoint.

## 3 INDEXING AND QUERYING WIKIDATA

In the following we describe how to index and query Wikidata as well as a comparison with existing alternatives.

### 3.1 Loading data

To load the full Wikidata dump into the qEndpoint, one needs to run a few instructions. This is done using our indexing method to efficiently create the HDT partition (Point 1 in the figure 1)

```
# Download the executable from https://github
#.com/the-qa-company/qEndpoint/releases/latest/
wget https://github.com/the-qa-company/qEndpoint
/releases/latest/download/qendpoint.jar
# Download the Wikidata dump
wget https://dumps.wikimedia.org/wikidatawiki/
entities/latest-all.nt.gz
# Start the qEndpoint executable (jar)
java -Xmx10G -jar qendpoint.jar
# Index the dataset (latest-all.nt.gz)
curl "http://127.0.0.1:1234/api/endpoint/load"
  -F "file=@latest-all.nt.gz"
```

These simple commands allow everyone to quickly have a SPARQL endpoint over Wikidata without the need of special configurations.

[2]https://github.com/rdfhdt/hdt-mr
[3]https://github.com/rdfhdt/hdt-java/pull/179

| Task | Time | Description |
|---|---|---|
| Dataset download | 7 h | Download the dataset[4] |
| HDT compression | 45 h | Creating HDT |
| HDT co-index gen | 5 h | Creating OPS/PSO/POS indexes |
| Loading the index | 2 min | Start the endpoint |

**Table 1: Time split during the loading of the Wikidata dataset.**

| System | Loading Time | #Triples | RAM | Index size | Doc |
|---|---|---|---|---|---|
| Apache Jena | 9d 21h | 13.8 B | 64 GB | 2TB | 1 |
| Virtuoso | several days[5] (preprocessing) + 10h | 11.9 B | 378 GB | NA | 2 |
| Blazegraph | ~5.5d | 11.9 B | 128 GB | 1.1 T | 3 |
| QLever | 14.3 h | 17 B | 128 GB | 823 GB | 4 |
| qEndpoint | 50 h | 17.4 B | 10 GB | 294 GB | 5 |

**Table 2: Wikidata Index characteristics for different endpoints**

(1) https://wiki.bitplan.com/index.php/WikiData_Import_2020-08-15
(2) https://community.openlinksw.com/t/loading-wikidata-into-virtuoso-open-source-or-enterprise-edition/2717
(3) https://addshore.com/2019/10/your-own-wikidata-query-service-with-no-limits/
(4) https://github.com/ad-freiburg/qlever/wiki/Using-QLever-for-Wikidata
(5) https://github.com/the-qa-company/qEndpoint/wiki/Use-qEndpoint-to-index-a-dataset

Note that the Wikidata dump is not uncompressed during this process. In particular, the disk footprint needed to generate the index is lower than the uncompressed dump (which is around 1.5 Tb). In the Table 1, we can see the time spent for the different indexing steps.

For comparison, to date, there exists only few attempts to successfully index Wikidata [5]. Since 2022, when the Wikidata dump exceeded 10B triples, only 4 triple stores are reported to be capable of indexing the whole dump, namely: Virtuoso [4], Apache Jena [6], QLever[1] and Blazegraph. We report in Table 2 the loading times, the number of indexed triples, the amount of needed RAM, the final index size and the documentation for indexing Wikidata.
Overall, we can see that the qEndpoint has the lowest RAM footprint as well as the lowest disk footprint. Also, it offers an easy documentation for compressing the whole Wikidata dump. The time to index is the second best compared to the other systems. Most notably, the qEndpoint is the only setup that allows currently to index Wikidata on commodity hardware.

### 3.2 Loading a pre-computed index

HDT is meant to be a format for sharing RDF datasets[7] and was therefore designed to have a particularly low disk footprint. Table 3 shows the sizes of the components needed to currently set up a Wikidata SPARQL endpoint using only HDT, which amounts

[6]https://jena.apache.org

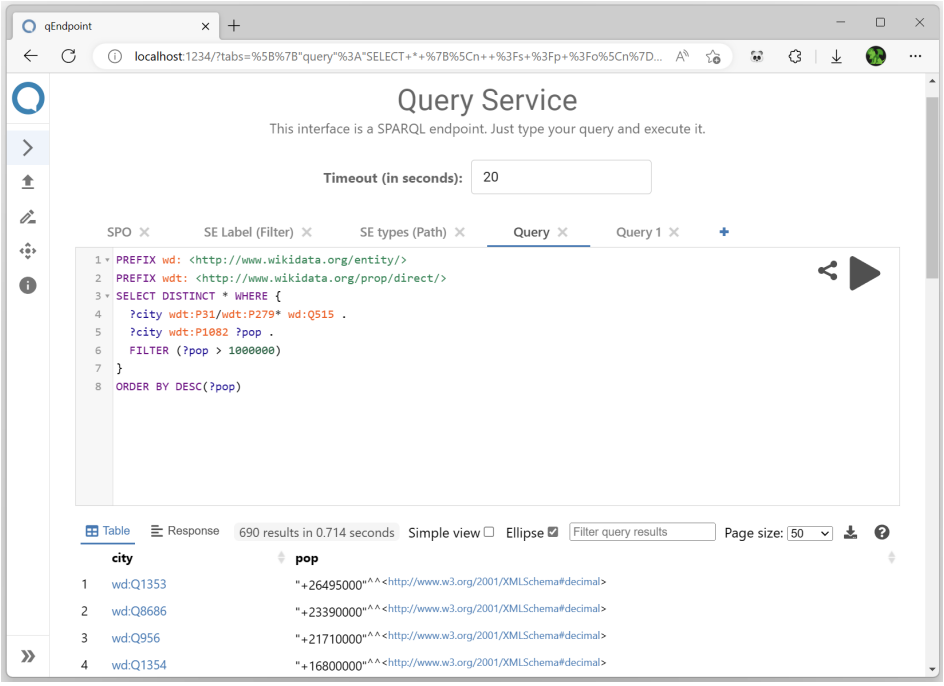


**Figure 2: qEndpoint interface with a basic query**

| File name | File size | Usage |
|---|---|---|
| index_dev.hdt | 183GB | Dictionary + SPO index |
| index_dev.hdt.index.v1-1 | 113GB | OPS/PSO/POS indexes |
| native-store | 16KB | RDF4J store |
| qendpoint.jar | 82MB | qEndpoint |

**Table 3: Sizes of each components of qEndpoint (total: 296GB)**

to the first two rows in the table, i.e. 300GB in total. The RDF4j counterpart, which is the third row in Table 3, corresponds to 16KB. Compared with the other endpoints in Table 2, the whole data can be easily downloaded in a few hours with any high-speed internet connection.

The second component in Table 1 of 113GB can be avoided in setups with slow connections since this co-index can be computed in 5h. As a consequence, it is possible to further reduce the amount of time required to deploy a full SPARQL endpoint. This time turns to be shrunk to a few minutes after downloading the files in Table 1[7]. Note that the whole bzip2 Wikidata dump is more than 150GB[8]. This means that by sharing the index, the setup time can be reduced to the amount of time that is necessary to download double of the size of the plain compressed Wikidata dump.

## 3.3 Queries

In the following, we discuss the evaluation of the query performance of the qEndpoint with other available systems. To the best of our knowledge, ours is also the first evaluation on the whole Wikidata dump using historical query logs.

As described above, while there are successful attempts to set up a local Wikidata endpoint, these are difficult to reproduce and depending on the cases the needed hardware resources are difficult to find [5]. Therefore, in order to compare with existing systems, we restrict to those whose setups are publicly available, namely:

- Blazegraph: the current system that is used in production by Wikimedia Foundation available at https://query.wikidata.org/sparql;
- Virtuoso: a live demo was set up in 2019[9] that is available at https://wikidata.demo.openlinksw.com/sparql;
- QLever: a live demo is available at https://qlever.cs.uni-freiburg.de/wikidata and was set up in 2022.

To benchmark the different systems, we performed a random extraction of 10K queries from Wikidata query logs [9] and ran them on the qEndpoint and on the above endpoints. Table 4 shows the query types for each query. The 10k queries where selected as follows. We picked 10k random queries of the interval 7 dump[10] matching these conditions:

(1) No usage of http://www.bigdata.com/ functions, internal to Blazegraph and not supported by other endpoints
(2) No MINUS operation, currently not supported by the qEndpoint.

The resources for the compared systems are the same as the one reported for indexing in Table 2.

[7]The index files are currently available at the URL https://qanswer-svc4.univ-st-etienne.fr/ into the RDF HDT open format
[8]https://dumps.wikimedia.org/wikidatawiki/entities/
[9]https://community.openlinksw.com/t/loading-wikidata-into-virtuoso-open-source-or-enterprise-edition/2717
[10]https://iccl.inf.tu-dresden.de/web/Wikidata_SPARQL_Logs/en

| Query type | Count |
|---|---|
| Triple Pattern (TP) | 5285 |
| Recursive / Path queries | 2852 |
| TP + Filter | 986 |
| TP + Union | 213 |
| Other | 664 |

**Table 4: Query count per type**

The results are presented in Figure 3 and Table 6.
We observe that the various systems have varying level of SPARQL support (see Table 6) and that qEndpoint via RDF4J can correctly parse all the queries. As shown in Figure 5, it achieves better performances than QLever in 44% of the cases (despite 4x lower memory footprint). It outperforms Virtuoso in 34% of the cases (despite 10x lower memory footprint) and it outperforms Blazegraph (the production system used by Wikimedia Deutschland with Wikidata) in 46% of the cases (despite a 4x lower memory footprint). Overall, we see that the median execution time between the qEndpoint and the other systems is -0.05. This means that, modulo a few outliers, we can achieve comparable query speed with reduced memory and disk footprint. By manually investigating some queries, we believe that the outliers are due to query optimization problems that we plan to tackle in the future.
Overall, despite running on commodity hardware, qEndpoint can achieve query speeds comparable to other existing alternatives. The dataset and the scripts used for the experiments are available online[11].

## 4 DEMONSTRATION OVERVIEW

During the demonstration, we plan to show the following capabilities of qEndpoint:

- how it is possible to index Wikidata on a commodity hardware;
- how it is possible to set up a SPARQL endpoint by downloading a pre-computed index; (Point 2 in the figure 1)
- the performance of qEndpoint on typical queries over Wikidata from our test dataset (see Table 4). Query types that are relevant for showcasing are simple triple pattern queries (TP), TPs with Unions, TPs with filters, Recursive path queries and other query types found in the Wikidata query logs. We will also be able to load queries formulated by the visitors of our demo booth.

The objective is to demonstrate that the qEndpoint performance is overall comparable with the other endpoints despite the considerable lower hardware resources. As such, it represents a suitable alternative to current resource-intensive endpoints.

| Endpoint | Max | Min | Mean | Median | % outperf. |
|---|---|---|---|---|---|
| QLever | 60 | -60.0 | -2.8 | -0.05 | 44.4 |
| Blazegraph | 4.74 | -60.0 | -2.8 | -0.04 | 45.9 |
| Virtuoso | 60.0 | -60.0 | -2.88 | -0.04 | 34.4 |

**Table 5: Statistics over the time differences in seconds**

| Endpoint | parsing errors | timeout errors | evaluation errors |
|---|---|---|---|
| QLever | 4174 | 0 | 0 |
| Virtuoso | 146 | 1 | 0 |
| qEndpoint | 0 | 808 | 0 |
| Blazegraph | 0 | 10 | 1 |

**Table 6: Errors per endpoint on 10K random queries of the Wikidata query logs.**

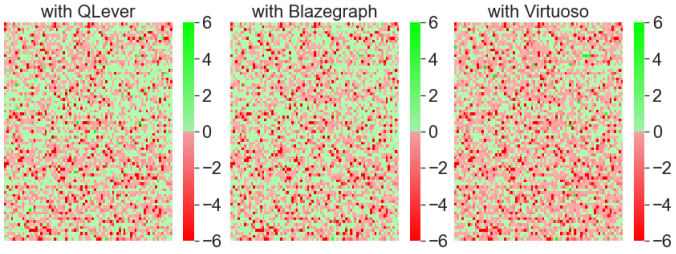


**Figure 3: qEndPoint time difference with 3 different endpoint (QLever, Virtuoso, Blazegraph from left to right) over 10K random queries without considering errors, one square is one query. A square is green if qEndPoint is faster and red if the baseline is faster. The intensity is based on the time difference which is between -6 and +6 seconds**

## REFERENCES


[1] Hannah Bast and Björn Buchhold. 2017. QLever: A Query Engine for Efficient SPARQL+Text Search. In *series = CIKM '17* (Singapore, Singapore). New York, NY, USA, 647–656. https://doi.org/10.1145/3132847.3132921

[2] Tim Berners-Lee. 2011. Designing the web for an open society. In *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar (Eds.). ACM, 3–4.

[3] Dennis Diefenbach and José M Giménez-García. 2020. HDTCat: let's make HDT generation scale. In *International Semantic Web Conference*. Springer, 18–33.

[4] Orri Erling and Ivan Mikhailov. 2009. *RDF Support in the Virtuoso DBMS*. Springer Berlin Heidelberg, Berlin, Heidelberg, 7–24. https://doi.org/10.1007/978-3-642-02184-8_2

[5] Wolfgang Fahl, Tim Holzheim, Andrea Westerinen, Christoph Lange, and Stefan Decker. 2022. Getting and hosting your own copy of Wikidata. In *3rd Wikidata Workshop @ International Semantic Web Conference*. https://zenodo.org/record/7185889#.Y0WD1y0RoQ0

[6] Javier D Fernández, Miguel A Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. 2013. Binary RDF representation for publication and exchange (HDT). *Journal of Web Semantics* 19 (2013), 22–41.

[7] Javier D Fernandez, Miguel A Martınez-Prieto, Claudio Gutierrez, Axel Polleres, and Mario Arias. [n. d.]. Binary RDF Representation for Publication and Exchange (HDT). ([n. d.]).

[8] José M Giménez-García, Javier D Fernández, and Miguel A Martínez-Prieto. 2015. HDT-MR: A scalable solution for RDF compression with HDT and MapReduce. In *ESWC 2015*. Springer, 253–268.

[9] Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. 2018. Getting the most out of Wikidata: semantic technology usage in Wikipedia's knowledge graph. In *International Semantic Web Conference*. Springer, 376–394.

[10] Laurens Rietveld and Rinke Hoekstra. 2013. YASGUI: not just another SPARQL client. In *ESWC 2013*. Springer, 78–86.


[11] https://github.com/the-qa-company/qEndpointWDQueries

Received 3 February 2023; revised ??; accepted ??