AWS Neptune Benchmark Over Real-World Dataset

Ghislain Auguste Atemezing

Benchmarking AWS Neptune Over Real-World Datasets.

Ghislain Auguste Atemezing[1]

Mondeca, 35 Boulevard de Strasbourg, 75010, Paris, France.
ghislain.atemezing@mondeca.com

Abstract. Since the announcement by Amazon of its own graph database service Neptune in November 2017, there have been many expectations on how to compare Neptune with other state-of-the-art enterprise graph databases. Neptune is defined as a high-performance graph database engine supporting popular graph models: RDF and Property Graph Model (PGM). This paper aims at giving an empirical evaluation of AWS Neptune on real-world RDF datasets. We use three different versions of Neptune (Preview, Neptune 1.0, and Neptune 1.0.1) to evaluate how fast and reliable the engine is with real-world SPARQL queries. Additionally, we compare some of the results with our previous benchmark with other enterprise RDF database graphs, even though one should be careful with such comparison since the hardware settings are not completely equivalent. The results of this evaluation give some preliminary insights about AWS Neptune in the RDF benchmark task. The results demonstrate that Neptune is the fastest in loading 2 Billion triples, performs better on analytical queries, and outperforms on updates queries. However, Neptune performs poorly on SELECT queries with the shortest response time (60s).

Keywords: AWS Neptune, RDF, SPARQL, Benchmark.
Resource type: Benchmark
Permanent URL: https://doi.org/10.6084/m9.figshare.13414817

1 Introduction

The adoption of semantic technologies for data integration is continuing to gain attention and adoption in industry, after the first impact in the research community. Knowledge Graphs (KGs) have proven to be an efficient way to structure, connect and share knowledge within organizations by bridging data silos. SPARQL [7] is the W3C *lingua franca* recommendation to access to the KGs encoded in RDF stored in graph database management systems. Since the announcement by Amazon of its own graph database service Neptune in November 2017 [2], there have been many expectations on how to compare Neptune with other state-of-the-art enterprise graph Database Management Systems (DBMS). According to 2020 DB-Engines ranking of Graph DBMS[1], Amazon Neptune is

[1] https://db-engines.com/en/ranking/graph+dbms, accessed 2020-12-17

the second after Virtuoso for RDF Graph DBMS. Neptune is defined as a high-performance graph database engine supporting popular graph models: RDF and Property Graph Model (PGM). This paper aims at giving an empirical evaluation of AWS Neptune on real-world RDF datasets, hence dealing with its RDF support.

In industry, many business requirements dealing with data management are shifting to use cloud-based services. This benchmark is motivated by following our previous assessment [3] using real-world datasets from the Publications Office (PO)[2] using Neptune. We were also able to assess on three different versions of Neptune corresponding to three different periods in time. This paper contributes to an empirical evaluation of Neptune across the evolved versions (Preview, 1.0 and 1.0.1) and to give an insight with our previous benchmark without entering in the "fairness" debate with the reader. We argue that the resource has an impact in assessing RDF stores in general, hence supports the adoption of Semantic Web technologies in industry.

The remainder of the paper is structured as follows: Section 2 presents a brief review of some related works on benchmarking enterprise RDF stores, although none of them are cloud-based Graph DBMS. Section 3 describes the selected queries and datasets used for the experiments. Section 4 describes the settings. The report of the loading process is described in Section 5. Then, we provide with the results of the benchmark in Section 6, followed by a discussion in Section 7. Section 8 concludes the paper and highlights future work.

2 Related Work

In the literature, several general purpose RDF benchmarks were developed on both artificial data and real datasets. We briefly summarize them in this section. FEASIBLE [10] which is a cluster-based SPARQL benchmark generator, which is able to synthesize customizable benchmarks from the query logs of SPARQL endpoints.

The Lehigh University Benchmark (LUBM) [6] a dataset generated for the university domain. In the publication domain, the SP2Bench [11] benchmark uses a both a synthetic test data and artificial queries.

The Berlin SPARQL Benchmark (BSBM) [4] applies a use case on e-commerce in various triple stores. BSBM data and queries are artificial.

The DBpedia SPARQL Benchmark (DBPSB) [8] is another more recent benchmark for RDF stores. It uses DBPedia with up to 239M triples, starting with 14M to compare the scalability. The Waterloo SPARQL Diversity Test Suite (WatDiv) [1] addresses the stress testing of five RDF stores for diverse queries and varied workloads.

Iguana framework [5] provides with a configurable and integrated environment for executing SPARQL benchmark. It also allows a uniform comparison of results across different benchmarks. However, we use for this benchmark a

[2] `https://publications.europa.eu`

different tool and plan to use Iguana for a more systematic benchmark with cloud-based RDF stores.

All the above-mentioned benchmarks are not in the cloud environment as it is the case of this work. We aim at pushing the benchmark comparison into the cloud since many stakeholders are transiting to adopt the Software-as-a-Service (SaaS) paradigm. To the best of our knowledge, this is the first publicly available benchmark of AWS Neptune on real-world datasets.

3 Dataset and Queries

3.1 Datasets

Two datasets are used for the loading experiment, a dump dataset used in production with 2,195 nquads files [9] and an augmented version based on the previous dataset to 2B triples. The dataset is available in Zenodo.[3]. For comparison in the loading process, we use a dump version of Wikidata[4] with 9,43B triples. Table 1 summarizes the statistics of the datasets.

Table 1. Datasets statistics and RDF serializations

Dataset	#Files	#Triples	RDF Format
PO Dataset	2195	727 959 570	NQUADS
PO Augmented Dataset	6585	2 183 878 710	NQUADS
Wikidata Dump	1	9.43B	Turtle

3.2 SPARQL Queries

We use three different types of SPARQL queries according to their usage at Pulications Office. Each time has a different goal with respect to the required time to complete.

- *Instantaneous queries*: These queries are generally used to dynamically generate dynamic visualizations on the website. Thus, they should be faster. In this group, we have a total of 20 queries, divided into 3 types of SPARQL queries: SELECT with 16 queries, DESCRIBE with 3 queries and CONSTRUCT with one query.
- *Analytical queries*: These queries are used for validation and mapping purposes at PO, where the most important feature is the quality of the results, not only the time to answer the query. In a total of 24 validation and mappings queries, all of them are `SELECT` SPARQL queries.

[3] `https://doi.org/10.5281/zenodo.1036738`

[4] https://dumps.wikimedia.org/wikidatawiki/entities/20190729/wikidata-20190729-all.ttl.gz

– *Update queries*: This set is composed of 5 SPARQL queries with 1 CONSTRUCT ; 1 DELETE/INSERT and 3 INSERT with a limit time to get the results in 10s.

4 Neptune Benchmark Settings

We now cover the settings for configuring AWS Neptune instance, and the benchmark settings.

4.1 AWS Neptune Configuration

Neptune is a service in the Amazon Web services (AWS). This means you need to first have an account. Once logged into the profile, the following steps are the ones required specifically for creating an instance of Neptune:

– Configure an Amazon Virtual Private Cloud (VPC), which is important to secure the access to your endpoint.
– Configure an Amazon Elastic Compute Cloud (EC2) instance. This is an important step because it is the location of the scripts to access the Neptune instances. Two information are also useful for security reason, a private key (.pem) and a public DNS.
– Configure a S3 bucket. It is the container to host the data to be loaded in Neptune, with the corresponding Identity and Access Management (IAM) role.
– Create an instance DB Neptune: This is where we actually create an endpoint in the same VPC than the Bucket S3. In our case, we choose the EAST-1 region. We use a db.r4.4xlarge (16 vCPU, 122 GB RAM)[5], which is somewhat closed to the settings on the previous benchmark[6]. However, for the purpose of comparing the effects of varying the size of the instances during the loading time, we use other types of instances, respectively `db.r4.8xlarge` and db.r5.12xlarge.

4.2 Benchmark Settings

The benchmark starts once the datasets are loaded into the AWS Neptune. We do not take into account the time of loading the source files in S3 bucket. The benchmark comprises the following steps:

1. **Configuration step**: We set in the corresponding configuration file the timeout value for the queries. This forces the store to abort or kill the process running the query.

[5] `https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.DBInstanceClass.html`
[6] Hardware configuration: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz, 6C/12T,128 GB RAM with SATA disk

2. **Warm-up step**: To measure the performance of a triple store under operational conditions, a warm-up phase is used. In the warm-up phase, query mixes are posed to the triple store. We used a warm-up set to 20, meaning that we run 20 times the set of queries in each category before starting the run phase.
3. **Hot-run step**: During this phase, the benchmark query mixes were sent to the tested store. We keep track of each run and output the results in a CSV file containing the statistics. We perform 5 runs in this stage and also set the max delay between query is set to `1000s`.

5 Results Bulk Loading

We go through the results obtained during the loading process, querying both three sets of SPARQL queries and a stress test.

The loading in Neptune is possible once the dataset is already available in a S3 Bucket. In our case, we had to first transfer it into the bucket, without reporting the time taken for this task. Hence, we assume the dataset is ready to be loaded into Neptune.

```
1  curl -X POST \
2      -H 'Content-Type: application/json' \
3    http://opocegen2bio.c1hdbvigzcza.us-east-1.neptune.amazonaws.com:8182/
         loader -d '
4      {
5        "source" : "s3://mdk-neptune-gen2bio",
6        "format" : "nquads",
7        "iamRoleArn" : "arn:aws:iam::672418254241:role/
             neptuneFroms3LoaderRole",
8        "region" : "us-east-1",
9        "failOnError" : "FALSE"
10     }'
```

Listing 1.1. Loading call process with AWS Neptune

In the listing 1.1, line 3 specifies the endpoint for the loader, and lines 4-8 the source S3 bucket, format, IAMRole and the location of the endpoint.

Loading on db.r4.4xlarge instance Table 2 summarizes the time to load different sizes of datasets. Wikidata Dump is used to estimate the time for almost 10 Billion. The results show an increase of 1 hour compared to Neptune Preview for loading 727.95 Million triples. Overall, the order of magnitude (less than 5 hours) is like Virtuoso (3.8h) and Stardog (4.59h). However, when it comes to load 2.18 Billion triples, Neptune is faster than Virtuoso (13.01h) and Stardog (13.30h)

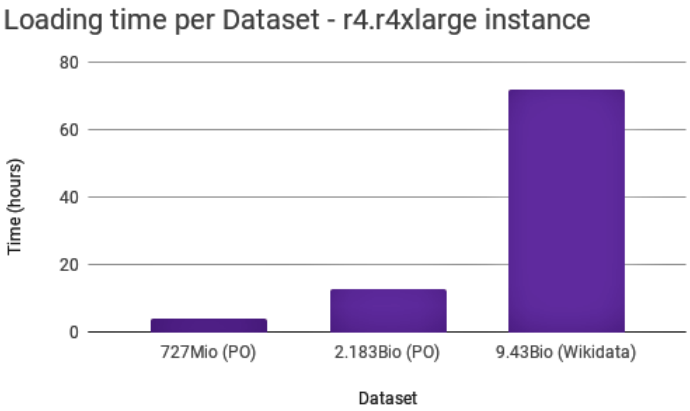
This result is useful in an emergency case of a database corruption with the need to reload the dataset from scratch within a reasonable exploitation time frame maintenance.

Figure 1 depicts the performance time in hours taken by Neptune 1.0.1 on a db.r4.xlarge instance.

Table 2. Loading time per dataset in Neptune 1.0.1

Dataset	Size	Time(h)
PO Dataset	727.95 Million	4.2
PO Augmented Dataset	2.183 Billion	12.9
Wikidata Dump	9.43 Billion	72

Fig. 1. Loading time in hours with Neptune. Mio=Million ; Bio=Billion



Loading on other EC2 instances We evaluate the loading with bigger instances of Neptune to evaluate the impact of the loader with respect to the hardware. We observe a high correlation (0.999) between the times on loading the above-mentioned datasets in both 4xlarge and 8xlarge. Table 3 shows the corresponding values obtained during this sub task.

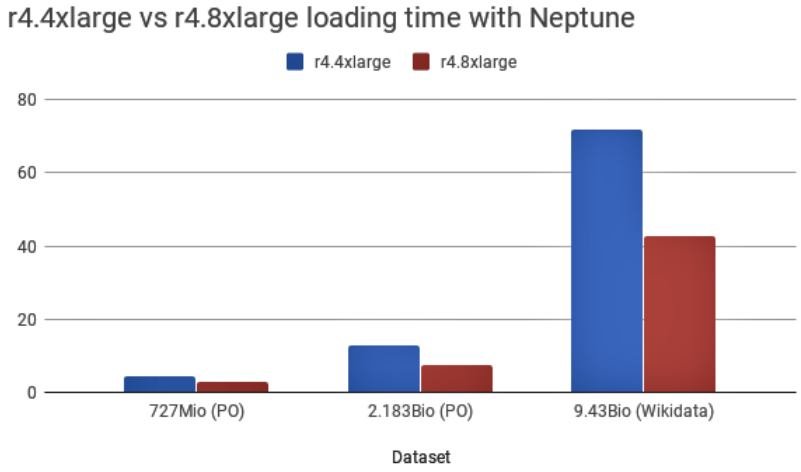
Table 3. Loading on a db.r4.4xlarge vs r4.8xlarge

Dataset size	Time 4xlarge (h)	Time 8xlarge (h)
727 Million	4.2	2.7
2.18 Billion	12.95	7.29
9.43 Billion	72	42.63

We even decide to load Wikidata Dump on a r5.12xlarge[7] (vCPU: 48 RAM: 384 GiB), and the task completed in $75,932s$, that is $21.09h$. Thus, reducing to almost one day compared to its predecessor, r4.8xlarge.

[7] `https://aws.amazon.com/ec2/instance-types/r5/`

Fig. 2. Overview of the completion time on different EC2 instances



6 Benchmarking AWS Neptune

We use the Sparql Query Benchmarker tool[8], an open-source tool based on Jena to run our experiment. We set 20 runs of mix queries per category to warm up the server. Furthermore, we make 5 additional runs to compute the average of the time taken for each query. Each group of queries has different time out which was decided based on functional requirements by experts at Publications Office. The timeout settings are `60s` for instantaneous queries and `600s` for analytical queries.

Listing 1.2 represents a script to launch the benchmark with a timeout set to 600s on a set of queries in a TSV file, and the results gathered in a CSV file.

```
./benchmark -s 0 -t 600 -q \ http://mdk.cluster-cai4oe44teuh.us-east-1-
    beta.rds.amazonaws.com:8182/sparql -m mixqueries_cat2_neptune.tsv \
-r 5 -w 20 -c results/results_neptune_group2.csv
```

Listing 1.2. sample command used to run an instance of AWS Neptune with analytical queries

6.1 Evaluating Instantaneous Queries

We proceed to compare the results of querying the RDF stores using the benchmarker tool. We gathered the output of each run in CSV files as explained in the previous section. We use set two substasks: (i) first by using a single thread and (ii) by emulating multiple clients with the same set of queries.

[8] `https://github.com/rvesse/sparql-query-bm`

Single Thread Table 4 shows the results obtained by the 3 versions of Neptune. Neptune 1.0.1 timed out with 8 queries, which an improvement over the preview version (9 timed out), but not with Neptune 1.0 (7 timed out). While query IQ10 was under $60s$, the same query with Neptune 1.0.1 timed out.

Comparing the results obtained with other enterprise RDF stores, such are Virtuoso, Oracle 12c, Stardog and GraphDB, we conclude that Neptune performs poorly with respect to this set of queries. Table 5 summarizes the number of queries with timeout obtained by all five RDF stores. Neptune is at the bottom of the ranking with Virtuoso the clear winner.

Query	Neptune P.	Neptune 1.0	Neptune 1.0.1
IQ1	.04	.07	.08
IQ2	.05	.07	.08
IQ3	.09	.07	.09
IQ4	.12	.05	.07
IQ5	**60**	**60**	**60**
IQ6	**60**	20.47	59.96
IQ7	**60**	**60**	**60**
IQ8	**60**	**60**	**60**
IQ9	**60**	**60**	**60**
IQ10	**60**	55.99	**60**
IQ11	.12	.05	.06
IQ12	**60**	**60**	**60**
IQ13	.09	.07	.11
IQ14	**60**	**60**	**60**
IQ15	.08	.04	.04
IQ16	.09	.05	.04
IQ17	**60**	**60**	**60**
IQ18	.21	.15	.21
IQ19	.14	.11	.14
IQ20	.09	.04	.05

Table 4. Average response time in second per queries over different versions of Neptune. Neptune P. = preview version.

It indicates that Neptune 1.0.0 solved the problem with IQ10, but it appeared in version 1.0.1. We observe also in general that Neptune 1.0 performed better than the other two versions.

Table 5. Number of timeouts per RDF stores

RDF Stores	nbTimeOut	Rank
Virtuoso	0	1
Oracle	2	2
Stardog	2	2
GraphDB	4	3
Neptune 1.0.1	8	4

Next, we manually rewrite seven queries (IQ5, IQ6, IQ7, IQ8, IQ12, IQ14 and IQ17) using the `EXPLAIN`[9] feature of Neptune. The main strategy is to incorporate `hint:Group hint:joinOrder "Ordered"`. Interestingly, Neptune is 4x faster (QMpH[10]=6.65 w.r.t. 26.59), and with a reduced number of queries reaching the limits. Table 6 shows the differences using optimized queries.

Query	Avg. time (s)	Previous bench (s)
IQ5r	.04	**60**
IQ6r	.68	59.96
IQ7r	.057	**60**
IQ8r	.03	**60**
IQ12r	.14	**60**
IQ14r	.03	**60**
IQ17r	.05	**60**

Table 6. Average response time in second per queries with optimized queries in Neptune 1.0.1

Multi-Thread In real-world settings, a SPARQL endpoint usually receives concurrent queries. We test this feature in the benchmark by emulating multi-threading to AWS Neptune 1.0.1 with respectively 5 clients, 20 clients, 50 clients, 70 clients and 100 clients. We observe a constant value of QMPH of **6.65**. Moreover, it places Neptune in second position after Virtuoso, and before Oracle, GraphDB and Stardog. Table 7 presents the results of QMpH values in case of multi-thread benchmark for instantaneous queries.

RDF Store	5clients	20clients	50clients	70clients	100clients
Neptune 1.0.1	6.653	6.654	6.655	6.654	6.654
Virtuoso 7.2.4.2	367.22	358.27	371.76	354.60	341.02
GraphDB EE 8.2	2.13	2.12	2.13	2.12	2.13
Stardog 4.3	1.973	1.94	1.97	1.96	1.95
Oracle 12c	2.10	1.99	2.01	2.01	2.02

Table 7. QMpH values in multi-threading bench for instantaneous queries

6.2 Evaluating Analytical Queries

We set 600s for timeout because the queries in this category are more analytical-based queries, and so need more time to complete. This value is based on the business requirement at PO. Table 8 presents the results of Neptune throughout the different versions. Surprisingly, there is no time out with these set of queries, and the latest version of Neptune is 4x faster than the two previous versions. Regarding the comparison

[9] `https://docs.aws.amazon.com/neptune/latest/userguide/sparql-explain-operators.html`

[10] QMpH = Query Mixed per Hour

with related triple stores, Table 9 reports the 3rd position for Neptune, compared to Virtuoso, GraphDB, Oracle and Stardog.

Table 8. Average response time in second per analytical SPARQL queries with Neptune versions.

Query	Neptune Preview	Neptune 1.0	Neptune 1.0.1
AQ1	15.94	13.69	12.98
AQ2	36.12	30.10	20.98
AQ3	56.62	43.71	28.81
AQ4	1.08	2.89	1.12
AQ5	63.71	102.75	19.70
AQ6	1.12	3.56	1.14
AQ7	.04	.14	.05
AQ8	.84	2.20	.62
AQ9	.06	.24	.05
AQ10	25.80	102.71	62.71
AQ11	.97	3.74	1.37
AQ12	28.16	594.60	87.79
AQ13	.24	.91	.74
AQ14	120.36	300.85	120.60
AQ15	14.01	28.39	15.42
AQ16	5.24	6.89	2.69
AQ17	2.63	17.16	1.56
AQ18	12.89	21.58	14.35
AQ19	5.23	8.09	2.72
AQ20	2.60	1.37	1.50
AQ21	17.78	25.42	19.71
AQ22	5.36	12.71	2.96
AQ23	2.77	5.14	1.78
AQ24	3.04	7.34	2.21

Table 9. Ranking of Neptune with other RDF stores using analytical queries

RDF Store	QMpH	Rank
Virtuoso 7.2.4.2	80.23	1
GraphDB EE 8.2	19.94	2
Neptune 1.0.1	8.4	3
Oracle 12c	2.41	4
Stardog 4.3	.89	5

6.3 Evaluating Updates Queries

Single Thread We set the queries in this group of queries to finish in 10s. Table 10, with a total of **14,694** QMpH. This result is almost 2K more than the results obtained with Virtuoso under the same queries. This is the first scenario where Neptune outperforms any other RDF store in this benchmark, with all the precaution with the comparison as we stated in the previous section. Table 11 summarizes the ranking for this set of queries.

Table 10. Average time in seconds with update queries

Query	Avg time (s)
UQ1	0.05
UQ2	0.07
UQ3	0.01
UQ4	0.01
UQ5	0.01

It also shows that the winner for this task is Neptune. Additionally, Neptune shares more or less the same order of magnitud with regards to numbers of QMpH. However, there is a huge gap with the three other RDF stores.

Table 11. Ranking of Neptune

RDF Stores	Avg time (s)	QMpH	Rank
Neptune	.24	14,594.81	1
Virtuoso	.29	12,372.49	2
GraphDB EE	.87	4,133.53	3
Stardog	11.83	304.14	4
Oracle	50	71.99	5

Multi-Thread In this scenario, we observe a non constant values when varying the number of clients. Figure 3 presents the evolution of QMpH, which starts with QMpH=5225 on 5 clients to reach the value of 461 with 100 simultaneous clients.

Fig. 3. Evolution of QMpH for different clients on updates queries

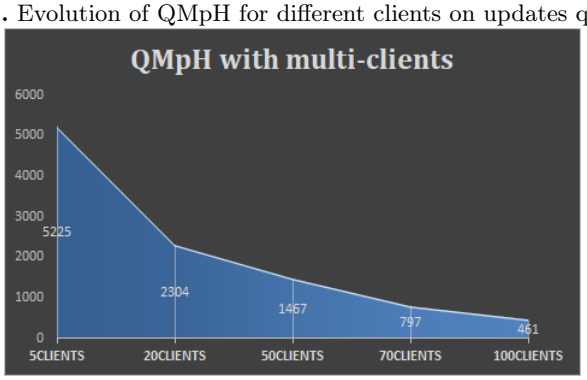


Table 12 presents an overview values with other RDF stores. Neptune follows Oracle in the highest numbers of QMpH. Surprisingly, this is the only situation where Oracle performs better than the rest of the RDF stores.

RDF Store	5clients	20clients	50clients	70clients	100clients
Neptune 1.0.1	5225	2304	1467	797	461
Virtuoso 7.2.4.2	71.32	48.34	48.51	48.32	48.26
GraphDB EE 8.2	273.18	146.87	79.52	52.18	56.24
Stardog 4.3	35.47	36.65	35.18	32.42	26.17
Oracle 12c	8382.21	8358.48	6980.34	7771.74	8718.96

Table 12. QMpH values in multi-threading bench for updates queries

6.4 Stability Test

We perform a stress test on the triple stores to have a quantitative indication related to stability. For this purpose, all the set of instantaneous queries are run continuously under a progressively increasing load to see how the engine reacts to high load. We use this test to empirically evaluate how stable is the RDF store.

The test starts by specifying the number of parallel clients within the script. Each client completes the run of the mix queries in parallel. The number of parallel clients is then multiplied by the ramp up factor and the process is repeated. This is repeated until either the maximum runtime or the maximum number of threads are reached. We set the maximum runtime to `180` minutes and set the maximum parallel threads to `128`.

Listing 1.3 displays a sample command used to run the stress test on a given triple store.

```
./stress -q <https://my/neptune.location.amazonaws.com:8182/sparql> -m
    mixqueries_cat1.tsv --max-runtime 180 --max-threads 128 --ramp-up 2.
```

Listing 1.3. Sample command used to run the stress test

Table 13. Results of the stress test on triple stores using instantaneous queries.

RDF Store	#mix runs	#op. run	Max.//.threads	# HTTP 5xx Errors
GraphDB	255	5,100	256	139
Virtuoso	255	5,100	256	4,732
Neptune 1.0.1	127	2,540	256	1,136
Stardog	92	1,840	128	576
Oracle	63	1,260	128	1,009

Neptune finishes with the limit of the parallel threads, unlike Virtuoso and GraphDB that completed the test after 180 minutes, reaching 256 parallel threads. The results in Table 13 suggest that Neptune is less stable than GraphDB and Virtuoso based on the total mix runs, the parallel threads and the total errors.

7 Discussion

We proceed to compare the results across the versions of Neptune, as well as with non cloud-based solutions of our previous work. We also briefly highlight some arguments for the potential impact of the resource.

7.1 Comparison across Neptune versions

The loader of Neptune 1.0.1 is less faster compared to Neptune Preview, at least with the experiment on 727 Million datasets, with almost the same behaviour with the previous version of 1.0. We observe a regression in terms of engine optimization when upgrading the minor version of Neptune 1.0. in the case of querying instantaneous queries. We agree on a faster engine after the preview release on the same set of queries.

7.2 Comparison with non-cloud-based RDF stores

We use Neptune 1.0.1 to proceed with some of our previous results with non-cloud RDF stores. Table 14 presents the results when querying those 8 queries with at least 6 Basic Graph Patterns (BGPs) in instantaneous queries. Neptune falls in 80% of the total queries.

Table 14. Comparison results time execution (in second) of the seven instantaneous queries with at least seven six BGPs.

Query	Neptune	Virtuoso	GraphDB	Stardog	Oracle	#BGP
IQ5	**60**	.09	.01	.82	31.35	7
IQ6	59.96	.28	.01	.10	39.35	6
IQ7	**60**	.06	.01	.01	34.82	7
IQ8	**60**	.10	.01	1.35	31.88	7
IQ9	**60**	.05	.01	.20	**60**	7
IQ10	**60**	.12	**60**	**60**	3.64	7
IQ19	.04	.11	**60**	**60**	.04	11

Now, we consider in the analytical queries, those queries containing a combination of at least three of the SPARQL features REGEX, DISTINCT, FILTER, OPTIONAL and GROUP BY. Table 15 shows the results by comparing Neptune with other RDF stores. While AQ14 hits a highest time reported by Neptune, it is rather surprising the huge difference with GraphDB, Virtuoso and Stardog.

7.3 Potential Impact

The main dataset used in production by the Publications Office for this benchmark has been granted usage to three persons from different organizations and RDF vendors (Allegrograph, RDFox) with the goal of replicating this benchmark. The recent justification of usage last September was by someone who wanted to:

Table 15. Comparison results time in seconds of querying analytical queries with a mix of SPARQL features.

Query	Neptune	Virtuoso	GraphDB	Stardog	Oracle
AQ13	.74	.06	.01	26.28	85.19
AQ14	120.60	.06	.01	.10	206.308
AQ15	15.42	.72	.06	**60**	3.74
AQ18	14.35	.06	.01	292.24	2.93

- *Perform more benchmarks against Neptune*
- *See how Neptune Query Plans hints can influence the response of complex SPARQL queries*

We imagine that the person is either using Neptune or works closely with Amazon. Therefore, we argue that the resource has an impact in assessing RDF stores in general, hence supports the adoption of Semantic Web technologies in industry.

In summary, this benchmark shows the following insights:

- Neptune loader has been slower since the Preview version, probably a design choice while gaining new features.
- Neptune 1.0 is faster than the upgraded minor version with instantaneous queries.
- Neptune 1.0.1 engine is 4x faster than the previous versions when it comes to analytical queries.
- In general, Neptune performs well in multi threading scenario, and outperforms in updates queries.
- The stability test reveals that Neptune is less stable than GraphDB and Virtuoso, which are the most stables in this task.
- âĂŞ Benchmarking a cloud-based solution comes with a financial cost, that has be taken into consideration when planning such task.

8 Conclusion

We have presented in this paper an empirical evaluation of AWS Neptune on real-world RDF datasets. We have described the steps to do such a benchmark with the goal to ease reproducibility. To this end, we used the same resource across three different versions of Neptune, which span for almost two years.

We compare the results obtained with our previous work on benchmarking enterprise RDF stores. This comparison is used to put in perspective, knowing the limitations of a strict comparison. For example, the settings of the hardware used in that work (SATA disk and 128 GB RAM) are not *stricto sensu* comparable with Amazon instance (db.r4.4xlarge), or that Neptune is a multi-modal data graph on the cloud.

The resource for this benchmark and the results are accessible online on Zenodo[11] with a CC-BY-4.0[12] license attached to it. We hope this work will show a growing interest in a more rigorous assessment of AWS Neptune with existing latest versions of RDF Graph Databases on the cloud.

[11] `https://doi.org/10.6084/m9.figshare.13414817`
[12] `https://creativecommons.org/licenses/by/4.0`

Acknowledgments. We would like to thank the AWS team based in Paris, in particular Jean-Philippe Pinte and Alice Temem for granting us credits to perform our test on Neptune.

References

1. G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee. Diversified stress testing of rdf data management systems. In *International Semantic Web Conference*, pages 197–212. Springer, 2014.
2. Amazon. Amazon neptune: Fast, reliable graph database built for the cloud, 11 2017.
3. G. A. Atemezing and F. Amardeilh. Benchmarking commercial rdf stores with publications office dataset. In *European Semantic Web Conference*, pages 379–394. Springer, 2018.
4. C. Bizer and A. Schultz. Benchmarking the performance of storage systems that expose sparql endpoints. *World Wide Web Internet And Web Information Systems*, 2008.
5. F. Conrads, J. Lehmann, M. Saleem, M. Morsey, , and A.-C. Ngonga Ngomo. IGUANA: A generic framework for benchmarking the read-write performance of triple stores. In *International Semantic Web Conference (ISWC)*, 2017.
6. Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182, 2005.
7. S. Harris, A. Seaborne, and E. PrudâĂŹhommeaux. Sparql 1.1 query language. *W3C recommendation*, 21(10):778, 2013.
8. M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. DBpedia SPARQL Benchmark—Performance Assessment with Real Queries on Real Data. In *ISWC 2011*, 2011.
9. O. Publications and Mondeca. Dump of rdf dataset used for rdf benchmark, 2017. http://doi.org/10.5281/zenodo.1036739.
10. M. Saleem, Q. Mehmood, and A.-C. N. Ngomo. Feasible: A feature-based sparql benchmark generation framework. In *International Semantic Web Conference*, pages 52–69. Springer, 2015.
11. M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. Spˆ 2bench: a sparql performance benchmark. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 222–233. IEEE, 2009.