



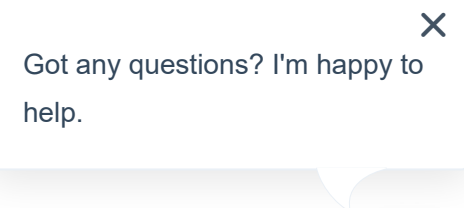
NoSQL Performance Benchmark 2018 – MongoDB, PostgreSQL, OrientDB, Neo4j and ArangoDB

February 14, 2018 *General* (<https://www.arangodb.com/category/general/>) *Tags: NoSQL* (<https://www.arangodb.com/tag/nosql/>), *Performance* (<https://www.arangodb.com/tag/performance/>), *RocksDB* (<https://www.arangodb.com/tag/rocksdb/>)

([/#twitter](#)) ([/#linkedin](#)) ([/#reddit](#)) ([/#hacker_news](#))

ArangoDB, as a native multi-model database, competes with many single-model storage technologies. When we started the ArangoDB project, one of the key design goals was and still is to at least be competitive with the leading single-model vendors on their home turf. But does a native multi-model database make sense. To prove that we are meeting our goals, we are competitive, we run and publish occasionally an update to the benchmark series. In this we included MongoDB, PostgreSQL (tabular & JSONB), OrientDB and Neo4j.

In this post we will cover the following topics:



1. Introduction and thanks (/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#Introduction)
2. Test Setup (/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#Performance)
3. Description of tests (/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#Description)
4. Results (/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#Overall)
 - Overall results (/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#Overall)
 - Aggregation Test (/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#Aggregation)
 - Neighbor Search (<https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#Neighbor>)
 - Shortest Path Test (/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#Path)
 - Memory Usage (/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#Memory)
 - Limiting Main Memory for ArangoDB with RocksDB (/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#Rocks)
5. Conclusion (/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#Conclusion)
6. Appendix – Details about Data, Machines, Products and Tests (/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#Appendix)

Got any questions? I'm happy to help.


Introduction to the Benchmark and Acknowledgements

This article is part of ArangoDB's open-source performance benchmark series. Since the previous post, there are new versions of competing software on which to benchmark. Plus, there are some major changes to ArangoDB software.

For instance, in latest versions of ArangoDB, an additional storage engine based on Facebook's RocksDB has been included. So we waited until its integration was finished before conducting a new benchmark test. Besides all of these factors, machines are now faster, so a new benchmark made sense.

Before I get into the benchmark specifics and results, I want to send a special **thanks to Hans-Peter Grahl for his fantastic help with MongoDB** queries. Wrapping my head around the JSON notation is for sure not impossible but boy can querying data be complicated. Thanks Hans-Peter for your help! **Big thanks as well to Max De Marzi and "JakeWins" both team Neo4j** for their contributions and improvements (<https://news.ycombinator.com/item?id=16375503>) to the 2018 Edition of our benchmark. Also big thanks to Spain **and ToroDB CEO/Founder Alvaro Hernandez for** contributing your knowledge for PostgreSQL (<https://news.ycombinator.com/item?id=16375503>). **Deep thanks to my teammates Mark, Michael and Jan** for their excellent and tireless work on this benchmark. Great teamwork crew!

After we published the previous benchmark, we received plenty of feedback from the community — thanks so much to everyone for their help, comments and ideas. We incorporated much of that feedback in this benchmark. For instance, this time we included the JSONB format for PostgreSQL.

 Got any questions? I'm happy to help.

Test Setup



For comparison, we used three leading single-model database systems: Neo4j for graph; MongoDB for document; and PostgreSQL for relational database. Additionally, we benchmarked ArangoDB against a multi-model database, OrientDB.

Of course, performing our own benchmark can be questionable. Therefore, we have published all of the scripts necessary for anyone to repeat this benchmark with minimum effort. They can be found here on Github:

- Setup and Import-Script (<https://github.com/weinberger/nosql-tests/blob/master/setupAll.sh>)
- Run-Benchmark-Script (<https://github.com/weinberger/nosql-tests/blob/master/runAll.sh>)


We used a simple client/server setup and instances AWS recommends for both relational and non-relational databases (<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/storage-optimized-instances.html>). We used the following instances:

- Server: i3.4xlarge on AWS with 16 virtual cores, 122 GB of RAM
- Client: c3.xlarge on AWS with four virtual CPUs, 7.5 GB of RAM and a 40 GB SSD

The setup costs ~35 US dollars a day.

To keep things simple and easily repeatable, all products were tested as they were downloaded. So you'll have to use the same scripts and instances if you want to compare your numbers to ours.

Got any questions? I'm happy to help.

We used the latest GA versions (as of January 26, 2018) of all database systems and not to include the RC versions. Below are a list of the versions we used for each product: 


- Neo4j 3.3.1
- MongoDB 3.6.1
- PostgreSQL 10.1 (tabular & jsonb)
- OrientDB 2.2.29
- ArangoDB 3.3.3

For this benchmark we used NodeJS 8.9.4. The operating system for the servers was Ubuntu 16.04, including the OS-patch 4.4.0-1049-aws — this includes Meltdown and Spectre V1 patches. Each database had an individual warm-up.

Descriptions of Tests

We use this benchmark suite internally for our own assessment, our own quality control, to see how changes in ArangoDB affect performance. Our benchmark is completely open-source. You can download all of the scripts necessary to do the benchmark yourself in our [repository](https://github.com/weinberger/nosql-tests) (<https://github.com/weinberger/nosql-tests>).

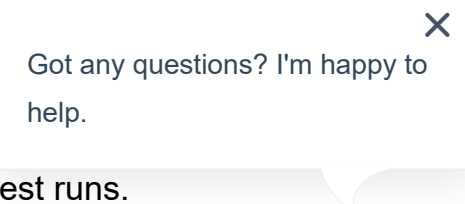
The goal of the benchmark is to measure the performance of each database system there is no query cache used. To be assured of this, we disabled the query cache for software that offered one. For our tests we ran the workloads twenty times, averaging results. Each test starts with an individual warm-up phase that allows the database systems to load data in memory.

 Got any questions? I'm happy to help.

For the tests, we used the Pokec dataset provided by the Stanford University SNAP (<https://snap.stanford.edu/data/soc-pokec.html>). It contains 1.6 million people (vertices) connected via 30.6 million edges. With this dataset, we can do basic, standard operations like single-reads and single-writes, but also graph queries to benchmark graph databases (e.g., the shortest path).

The following test cases have been included, as far as the database system was capable of performing the query:

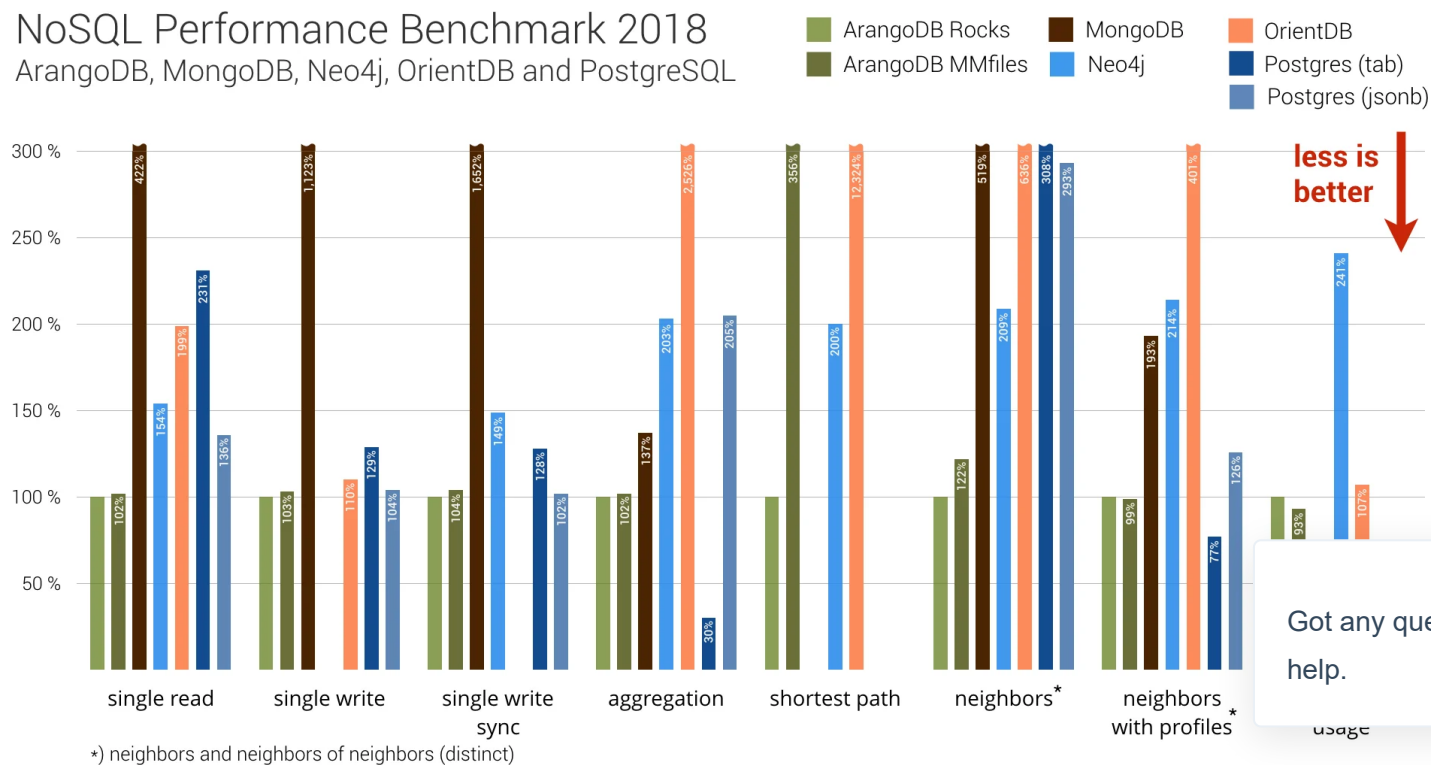
- **single-read:** these are single document reads of profiles (i.e., 100,000 different documents).
- **single-write:** these are single document writes of profiles (i.e., 100,000 different documents).
- **single-write sync:** these are the same as single-writes, but we waited for fsync on every request.
- **aggregation:** these are ad-hoc aggregation over a single collection (i.e., 1,632,803 documents). We computed the age distribution for everyone in the network, simply counting how often each age occurs.
- **neighbors second:** we searched for distinct, direct neighbors, plus the neighbors of the neighbors, returning ID's for 1,000 vertices.
- **neighbors second with data:** we located distinct, direct neighbors, plus the neighbors of the neighbors and returned their profiles for 100 vertices.
- **shortest path:** this the 1,000 shortest paths found in a highly connected social graph answers the question how close to each other two people are in the social network.
- **memory:** this is the average of the maximum main memory consumption during test runs.



The throughput measurements on the test machine for ArangoDB — with RocksDB as storage engine — defined the baseline (100%) for the comparisons. Lower percentages indicate a higher throughput. Accordingly, higher percentages indicate lower throughput.

Overall Results

The graph below shows the overall results of our performance benchmark. In the sub-sections after this graph, we provide more information on each test.



Learn more about ArangoDB with our technical white paper on [What is a Multi-model Database and Why Use It?](/arangodb-white-papers/white-paper-multi-model-database/) (/arangodb-white-papers/white-paper-multi-model-database/)



As you can see, a native multi-model can compete with single-model database systems. We are especially pleased that our new RocksDB-based storage engine performed well against the competition. I think the whole team can be proud of this integration.

In fundamental queries like single-read, single-write, as well as single-write sync, we achieved positive results and performed even better than PostgreSQL. The shortest path query was not tested for MongoDB or PostgreSQL since those queries would have had to be implemented completely on the client side for those database systems.

Please note that in previous benchmarks, MongoDB showed better results in single read/write tests. The table below shows the results of the most recent setups (database+driver on benchmark day) for all databases.

Got any questions? I'm happy to help.

NoSQL Performance Benchmark 2018

Absolute & normalized results for ArangoDB, MongoDB, Neo4j and OrientDB



	single read (s)	single write (s)	single write sync (s)	aggregation (s)	shortest (s)	neighbors 2nd (s)	neighbors 2nd data (s)	memory (GB)
ArangoDB 3.3.3 (rocksdb)	100%	100%	100%	100%	100%	100%	100%	100%
	23.25	28.07	28.27	01.08	0.42	1.43	5.15	15.36
ArangoDB 3.3.3 (mmfiles)	102.16%	102.55%	103.89%	102.40%	816.06%	122.07%	99.32%	92.87%
	23.76	28.79	29.37	1.10	3.40	1.75	5.12	14.27
MongoDB 3.6.1 (Wired Tiger)	422.38%	1123.36%	1652.09%	136.65%		518.83%	192.88%	50.64%
	98.24	315.33	466.99	1.47		7.42	9.94	7.70
Neo4j 3.3.1	153.65%		149.37%	203.45%	199.94%	208.96%	214.22%	240.68%
	35.73		43.22	2.18	0.83	2.99	11.04	37.00
PostGRES 10.1 (tabular)	231.17%	129.03%	127.70%	29.62%		307.96%	76.87%	26.68%
	53.77	36.22	36.10	0.32		4.41	3.96	4.10
PostGRES 10.1 (jsonb)	135.96%	104.34%	101.55%	204.55%		292.57%	126.14%	35.36%
	31.62	29.29	28.70	2.20		4.19	6.50	5.43
OrientDB 2.2.29	198.84%	110.37%		2526.29%	12323.67%	636.45%	400.97%	107.04%
	46.25	30.98		27.19	51.34	9.11	20.67	16.45

These are just the results. To appreciate and understand them, we'll need look a little deeper into the individual results and focus on the more complex queries like aggregations and graphy functionalities.

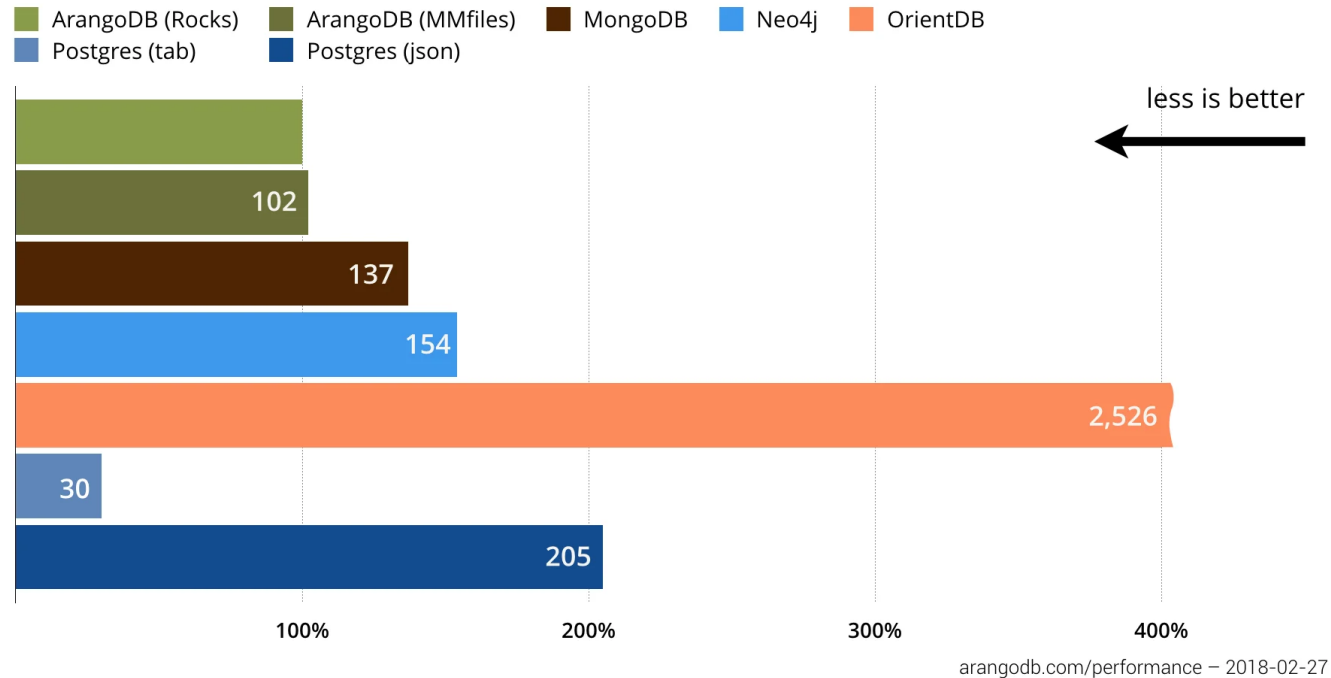
Age Distribution in a Social Network — Aggregation Test

In this test, we aggregated over a single collection (i.e., 1,632,803 documents). We statistics about the age distribution for everyone in the network by simply counting h each age occurs. We didn't use a secondary index for this attribute on any of the databases

Got any questions? I'm happy to help.

that they all have to perform a full-collection scan and do a counting statistics — this is a typical ad-hoc query. = 🔍

Aggregation



New to multi-model and graphs? Check out our free [ArangoDB Graph Course \(/arangoDB/graph-course/\)](#).

Got any questions? I'm happy to help.

Computing the aggregation is efficient in ArangoDB, taking on an average of 1.07 seconds and defining the baseline. Both storage engines of ArangoDB show acceptable performance. As expected, PostgreSQL as the representative of a relational world, performs best with only 0.3 seconds, but only when the data is stored as tabular. For the same task, but with data stored as

a JSONB document, PostgreSQL needs much more time compared to MongoDB and more than twice the time compared to ArangoDB. Since our previous benchmark, OrientDB ~~doesn't~~^Q seem to have improved much and is still slower by a factor of over 20x.

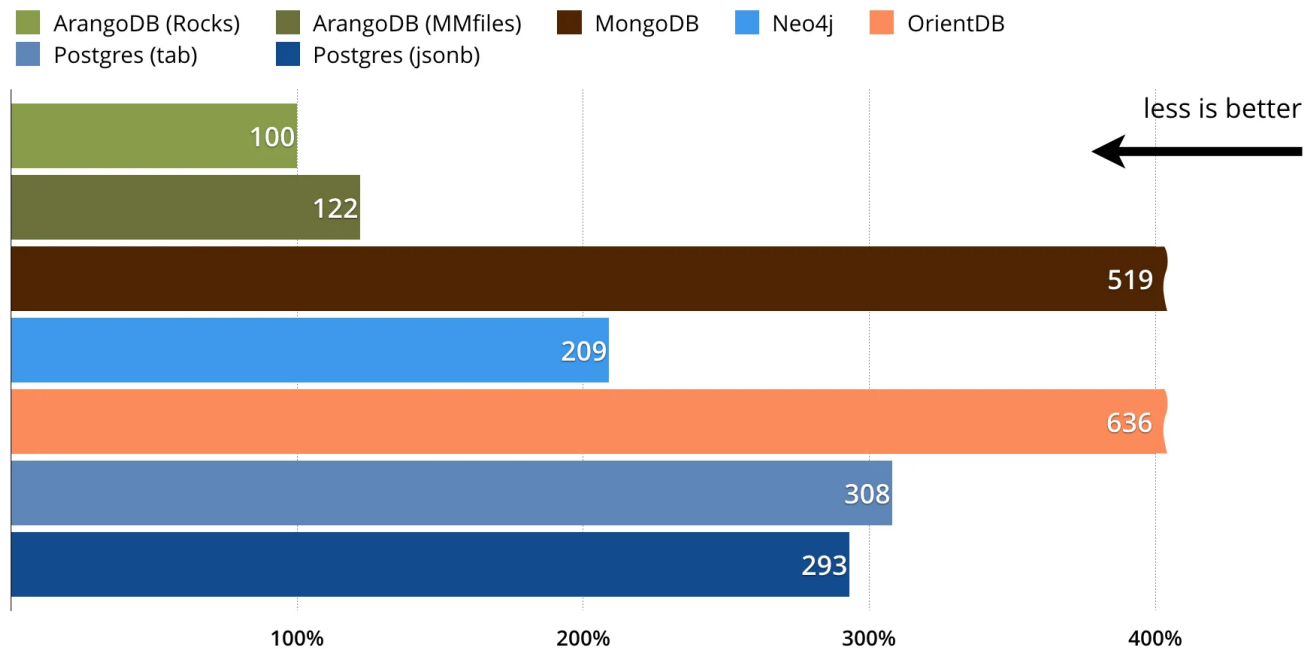
We didn't create special indices for JSONB in PostgreSQL since we didn't create additional indices for any other products. Since we wanted to test ad-hoc queries, it's valid to assume that no indices are present in the case of ad-hoc queries.

Extended Friend Network —Neighbor Search, Neighbors with Profile Data Test

This may sound like a pure graph query but as we searched within a known depth, other databases can also perform this task to find neighbors. We tested two different queries. First, a simple distinct lookup of the neighbors of neighbors and second the distinct neighbors of neighbors with the full profile data.

Got any questions? I'm happy to help.

Neighbors of Neighbors



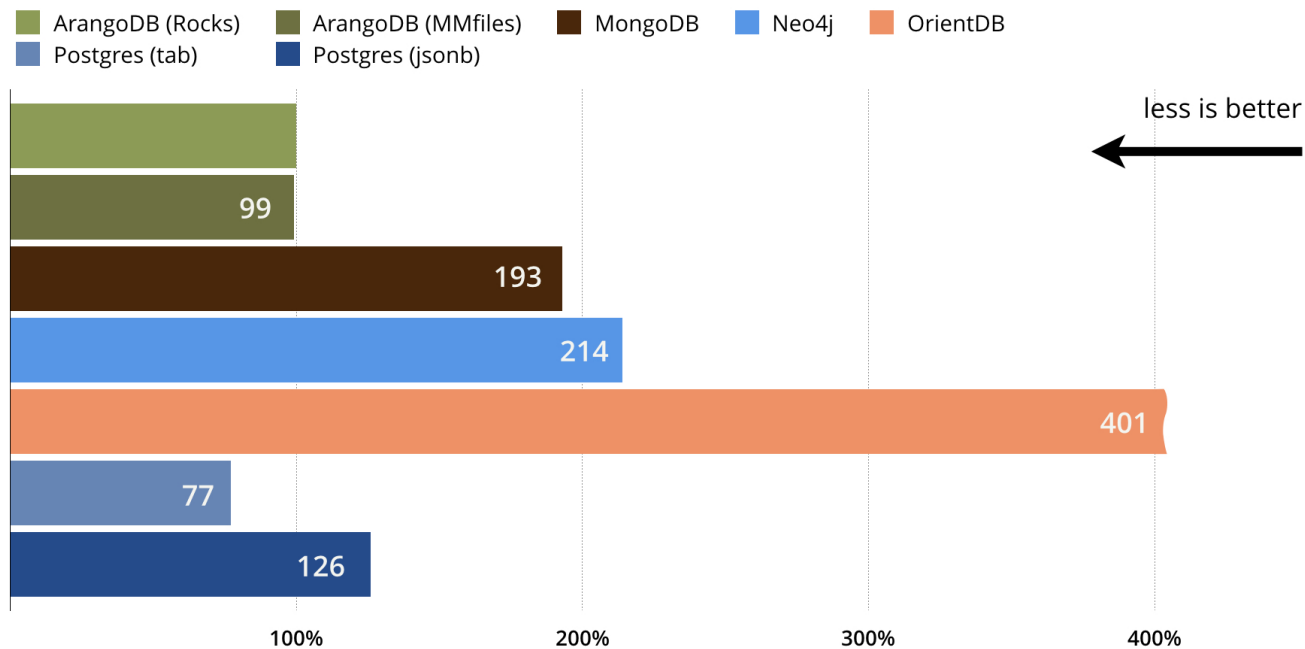
arangodb.com/performance – 2018-02-27

OrientDB and MongoDB didn't perform well in this test. ArangoDB shows comparatively good performance for neighbors of neighbors search.

A more challenging task for a database is of course retrieving also the profile data of neighbors. ArangoDB also works efficiently at this tasks but PostgreSQL is still 23 p (see below).

Got any questions? I'm happy to help.

Neighbors of Neighbors with Profile Data



arangodb.com/performance – 2018-02-27

The reason for the good performance of ArangoDB is the optimized edge index which allows for fast lookup of connected edges and vertices for a certain node, this is presumably faster than general index lookups.

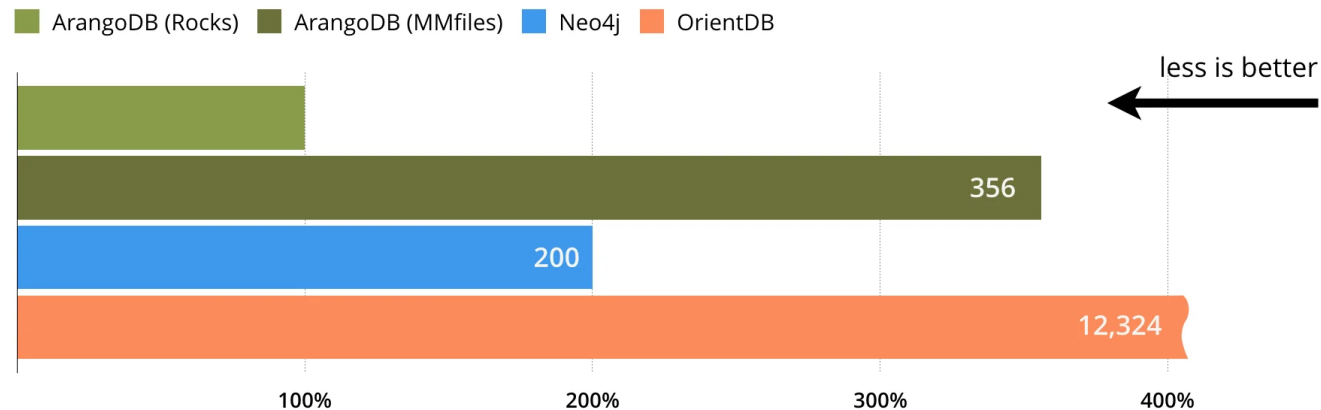
Shortcut between Two Entities — Shortest Path Test

The shortest path algorithm is a speciality of graph databases. The algorithm searches for the shortest distance between a start vertex and an end vertex. It returns the shortest path with all edges and vertices. With this you can determine the outcome of such queries to be used, for example, on LinkedIn when it shows the “Mutual Connections” on someone’s profile page.

Got any questions? I'm happy to help.

The task for this test was to find 1,000 shortest paths in a highly connected social network to answer the question how close two persons are in the network. = 🔍

Shortest Path



arangodb.com/performance – 2018-02-27

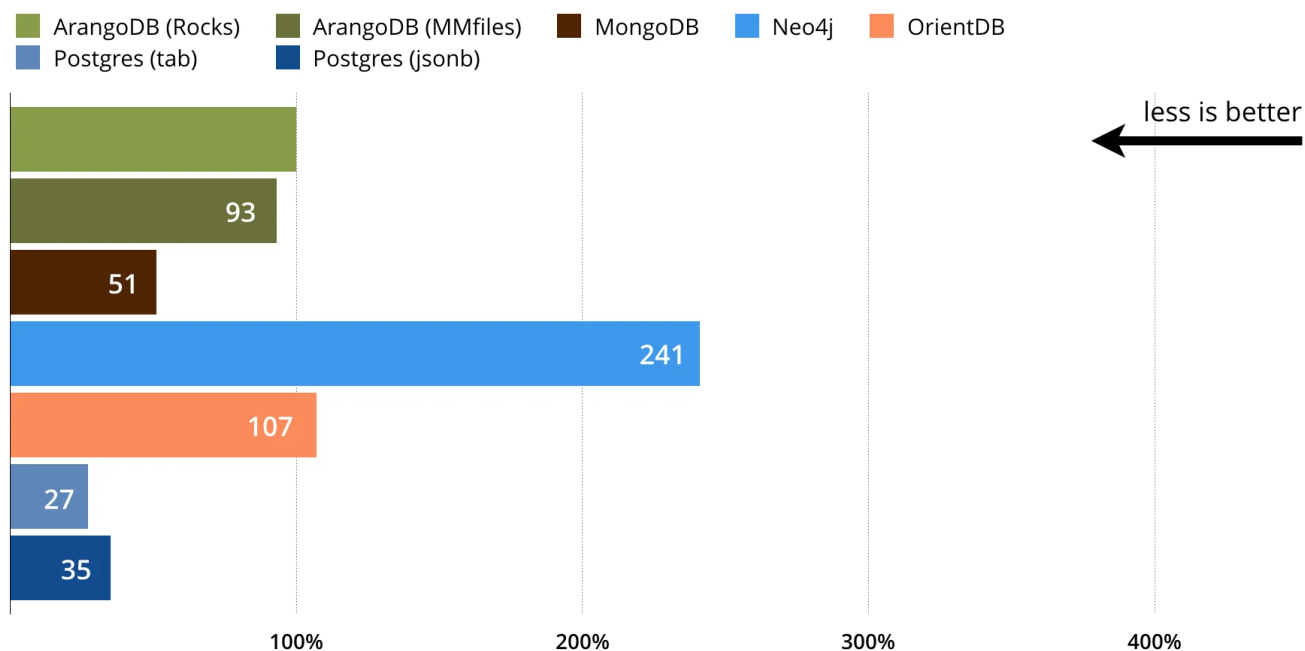
Since the integration of RocksDB in ArangoDB, shortest path queries have become as fast as 416ms to find 1,000 shortest paths. ArangoDB is twice as fast as Neo4j and one-hundred times faster than OrientDB. The RocksDB engine compared to the MM of ArangoDB is much better because it also has improved graph capabilities.

✕
Got any questions? I'm happy to help.

Memory Usage

In the previous benchmark, main memory usage was a challenge for ArangoDB — it still is to some extent. In this benchmark, we measured a higher memory footprint of up to 3.7 ^Q times the main memory consumption, compared to the best measured result of PostgreSQL (tabular).

Memory Consumption



arangodb.com/performance – 2018-02-27

Neo4j seems to have improved on the performance side by increasing the memory. Compared to the previous benchmark, they went from second best to last place.

Without any configuration, RocksDB can consume up to two-third of the available memory and does so until this limit is reached. It's until then that RocksDB starts to throw unneeded data out of main memory. This is also a reason for ArangoDBs high memory consumption with

Got any questions? I'm happy to help.

RocksDB.



Interested in trying out ArangoDB? Fire up your cluster in just few clicks with ArangoDB
Oasis: the Cloud Service for ArangoDB. [Start your free 14-day trial here](https://cloud.arangodb.com/home?utm_source=arangodb_website&utm_medium=top_blogpost&utm_campaign=traffic)
([https://cloud.arangodb.com/home?](https://cloud.arangodb.com/home?utm_source=arangodb_website&utm_medium=top_blogpost&utm_campaign=traffic)
[utm_source=arangodb_website&utm_medium=top_blogpost&utm_campaign=traffic](https://cloud.arangodb.com/home?utm_source=arangodb_website&utm_medium=top_blogpost&utm_campaign=traffic)).

Limiting Main Memory for ArangoDB with RocksDB

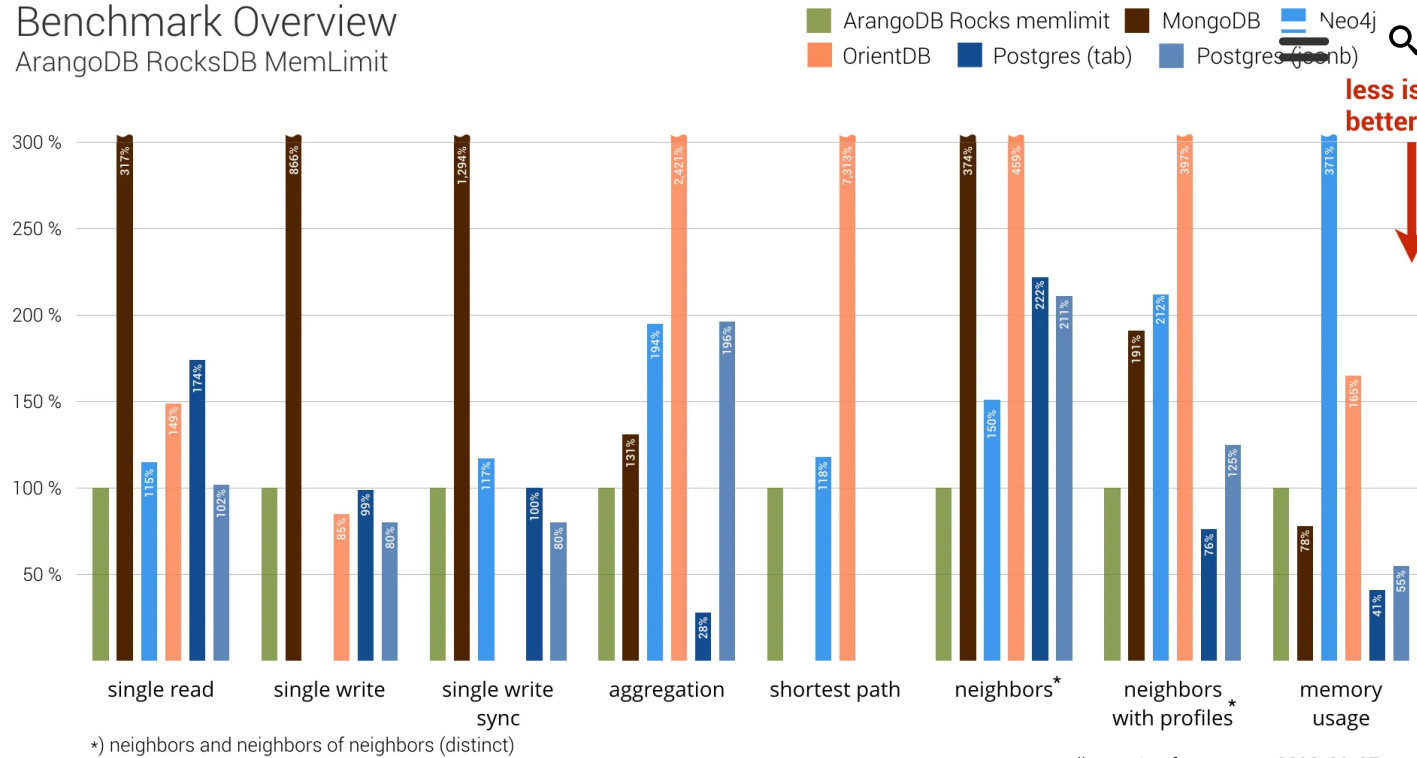
The great thing about RocksDB is that it's highly configurable. You can define the upper limit of the allowed memory usage. We were curious, though, what would happen if we set the memory limit to 10 GB and ran the complete benchmark again.



Got any questions? I'm happy to help.

Benchmark Overview

ArangoDB RocksDB MemLimit



arangodb.com/performance – 2018-02-27

Overall, ArangoDB with a memory limit on RocksDB is still fast in many test cases. ArangoDB loses a bit in single-writes and single-reads, but achieves nonetheless an acceptable overall performance.

Memory usage is still not optimal on ArangoDB. However, with the RocksDB storage you have plenty of options so that you can optimize for your use case. Plus, we suspect there are more tweaks we can do to get even better performance. RocksDB is still key to ArangoDB: we haven't yet tapped into all that it offers.

Got any questions? I'm happy to help.

Conclusion

If you're not yet convinced, take a look at the [Github repository](https://github.com/weinberger/nosql-tests) (<https://github.com/weinberger/nosql-tests>). Do your own tests — and please share your results if you do. Keep in mind when doing benchmark tests that different hardware can produce different results. Also, keep in mind that your performance needs may vary and your requirements may differ. Because of all of this, you should use our repository as a boilerplate and extend it with your own tests.

In this benchmark we could show again, that ArangoDB can compete with the leading single-model database systems on their home turf. And we've demonstrated again that we can also compete with another multi-model database, OrientDB.

In conclusion, the excellent performance and superior flexibility of a native multi-model is a key advantage of ArangoDB. It's a good reason to try ArangoDB for your use case (<https://www.arangodb.com/download/>).

Learn more with our technical white papers [What is a Multi-model Database and Why Use It? \(/white-paper-multi-model-database/\)](/white-paper-multi-model-database/), or check out [Switching from Relational to ArangoDB \(/white-paper-switching-relational-database/\)](/white-paper-switching-relational-database/) .

Appendix – Details about Data, Machines, Products and

For this NoSQL performance benchmark, we used the same data and the same hardware to test each database system. If you want to check or understand better our results, in this appendix we provide details on the data, the equipment, and the software we used. We also

✕
Got any questions? I'm happy to help.

provide more details on the tests we performed, as well as describe some of the adjustments made to accomodate the nuances of some database systems.



Data

Pokec is the most popular online social network in Slovakia. We used a snapshot of its data provided by the Stanford University SNAP. It contains profile data from 1,632,803 people. The corresponding friendship graph has 30,622,564 edges. The profile data contain gender, age, hobbies, interest, education, etc.

However, the individual JSON documents are very diverse because many fields are empty for many people. Profile data are in the Slovak language. Friendships in Pokec are directed. The uncompressed JSON data for the vertices need around 600 MB and the uncompressed JSON data for the edges require around 1.832 GB. The diameter of the graph (i.e., longest shortest path) is 11, but the graph is highly connected, as is normal for a social network. This makes the shortest path problem particularly hard.

Hardware

All benchmarks were done on a virtual machine of type i3.4xlarge (server) on AWS with 16 virtual cores, 122 GB of RAM and a 1900 GB NVMe-SSD. For the client, we used a on AWS with four virtual CPUs, 7.5 GB of RAM and a 40 GB SSD.

Got any questions? I'm happy to help.

Software

We wanted to use a client/server model for the benchmark. For this, we needed a language to implement the tests. Therefore, we decided that it has to fulfill the following criteria:

- Each database in the comparison must have a reasonable driver.
- It's not one of the native languages our contenders has implemented. This would potentially give an unfair advantage for some. This ruled out C++ and Java.
- The language must be reasonably popular and relevant in the market.
- The language should be available on all major platforms.

This essentially left JavaScript, PHP, Python, Go and Ruby. We decided to use JavaScript with node.js 8.9.4. It's popular and known to be fast, in particular with network workloads.

For each database we used the most up-to-date JavaScript driver that was recommended by the respective database vendor. We used the following Community Editions and driver versions:

- ArangoDB V3.3.3 for x86_64 (arangojs@5.8.0 driver)
- MongoDB V3.6.1 for x86_64, using the WiredTiger storage engine (mongodb@3.0.1 driver)
- Neo4j V3.3.1 running on openjdk 1.8.0_151 (neo4j@1.5.3 driver)
- OrientDB 2.2.29 (orientjs@2.2.7 driver)
- PostgreSQL 10.1.1 (pg-promise@7.4.1 driver)

All databases were installed on the same machine. We did our best to tune the configuration parameter. For example, we switched off transparent huge pages and configured up open file descriptors for each process. Furthermore, we adapted community and vendor provided configuration parameters from Michael Hunger of Neo4j and Luca Garulli of ArangoDB to improve individual settings.

Tests

Got any questions? I'm happy to help.

We made sure for each experiment that the database had a chance to load all relevant data into RAM. Some database systems allow explicit load commands for collections, while others do not. Therefore, we increased cache sizes where relevant and used full collection scans as a warm-up procedure.

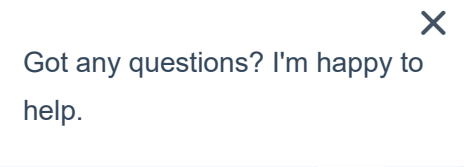
We didn't want to benchmark query caches or likewise — a database might need a warm-up phase, but you can't compare databases based on cache size and efficiency. Whether a cache is useful or not depends highly on the individual use case, executing a certain query multiple times.

For single document tests, we used individual requests for each document, but used keep-alive and allowed multiple simultaneous connections. We did this since we wanted to test throughput rather than latency.

We used a TCP/IP connection pool of up to 25 connections, whenever the driver permitted this. All drivers seem to support this connection pooling.

Single Document Reads (100,000 different documents)

In this test we stored 100,000 identifiers of people in the node.js client and tried to fetch the corresponding profiles from the database, each in a separate query. In node.js, everything happens in a single thread, but asynchronously. To load fully the database connection we submitted all queries to the driver and then waited for all of the callbacks using the node.js event loop. We measured the wallclock time from just before we started sending queries until the last answer arrived. Obviously, this measures throughput of the driver and database combination and not latency. Therefore, we gave as a result the complete wallclock time for all requests.



Single Document Writes (100,000 different documents)

For this test we proceed similarly: We loaded 100,000 different documents into the `node.js` client and then measured the wallclock time needed to send all of them to the database, using individual queries. We again first scheduled all requests to the driver and then waited for all callbacks using the `node.js` event loop. As above, this is a throughput measurement.

Single Document Writes Sync (100,000 different documents)

This is the same as the previous test, but we waited until the write was synced to disk — which is the default behavior of Neo4j. To be fair, we introduced this additional test to the comparison.

Aggregation over a Single Collection (1,632,803 documents)

In this test we did an ad-hoc aggregation over all 1,632,803 profile documents and counted how often each value of the AGE attribute occurred. We didn't use a secondary index for this attribute on any of the databases. As a result, they all had to perform a full collection scan and do a counting statistics. We only measured a single request, since this is enough to get an accurate measurement. The amount of data scanned should be more than any CPU cache can hold. We should see real RAM accesses, but usually no disk accesses because of the above warm-up procedure.

Finding Neighbors and Neighbors of Neighbors (distinct, for 1,000 vertices)

This was the first test related to the network use case. For each of 1,000 vertices we found all of the neighbors and all of the neighbors of all neighbors. This requires finding the friends of the friends of a person and returning a distinct set of friend ID's. This is a typical graph matching problem, considering paths of length one or two. For the non-graph database

Got any questions? I'm happy to help.

MongoDB, we used the aggregation framework to compute the result. In PostgreSQL, we used a relational table with id `from` and id `to`, each backed by an index. In the Pokec dataset, we found 18,972 neighbors and 852,824 neighbors of neighbors for our 1,000 queried vertices.

Finding Neighbors and Neighbors of Neighbors with Profile Data (distinct, for 100 vertices)

We received feedback from previous benchmarks that for a real use case we need to return more than ID's. Therefore, we added a test of neighbors with user profiles that addresses this concern and returns the complete profiles. In our test case, we retrieved 84,972 profiles from the first 100 vertices we queried. The complete set of 853,000 profiles (1,000 vertices) would have been too much for nodejs.

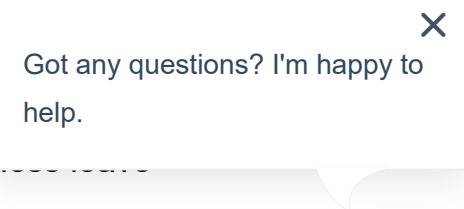
Finding 1000 Shortest Paths (in a highly connected social graph)

This is a pure graph test with a query that is particularly suited for a graph database. We asked the databases in 1000 different requests to find the shortest path between two given vertices in our social graph. Due to the high connectivity of the graph, such a query is hard, since the neighborhood of a vertex grows exponentially with the radius. Shortest path is notoriously bad in more traditional database systems, because the answer involves an *a priori* unknown number of steps in the graph, usually leading to an *a priori* unknown number of joins.

Nuances

The section above describes the tests we performed with each database system. However, each has some nuances that required some adjustments. One cannot always in fair comparison have all factors constant.

ArangoDB



ArangoDB allows you to specify the value of the primary key attribute `_key`, as long as the unique constraint is not violated. It automatically creates a primary hash index on that attribute, as well as an edge index on the `_from` and `_to` attributes in the friendship relation (i.e., the edge collection). No other indexes were used.

MongoDB

Since MongoDB treats edges just as documents in another collection, we helped it a bit for the graph queries by creating two more indexes on the `_from` and `_to` attributes of the friendship relation. For MongoDB, we had to avoid the `$graphlookup` operator to achieve acceptable performance. We tested the `$graphlookup`, but performance was so slow that we decided not to use it and wrote the query in the old way, as suggested by Hans-Peter Grahsl. We didn't even try to do shortest paths.

Please note that as the stats for MongoDB worsened significantly in comparison to what we measured in 2015, we reran the test for MongoDB with the same NodeJS version that we used in the 2015 benchmark. Results for single-reads and single-writes were slightly better with the old NodeJS version, but with no effect on the overall ranking. Since we tested the latest setup for all products, we didn't publish the results.

Neo4j

In Neo4j, the attribute values of the profile documents are stored as properties of the nodes. For a fair comparison, we created an index on the `_key` attribute. Neo4j claims to use "index-free adjacency" for the edges. So we didn't add another index on edges.

OrientDB

Got any questions? I'm happy to help.

For OrientDB, we couldn't use version 2.2.31, which was the latest one, because a bug in version 2.2.30 in the `shortest_path` algorithms hindered us to do the complete benchmark. We reported the bug on Github (<https://github.com/orientechnologies/orientdb/issues/8013>) and the OrientDB team fixed it immediately but the next maintenance release was published after January 26.

Please note that if you are doing the benchmark yourself and OrientDB takes more than three hours to import the data, don't panic. We experienced the same.

PostgreSQL (tabular & JSONB)

We used PostgreSQL with the user profiles stored in a table with two columns, the Profile ID and a JSONB data type for the whole profile data. In a second approach, for comparison, we used a classical relational data modelling with all profile attributes as columns in a table. As PostgreSQL starts per default with a main memory limit of only 128MB, we used a PostgreSQL tuning configurator (<http://pgtune.leopard.in.ua/>) to provide fair conditions for everyone.

Resources and Contribution

All code used in these tests can be downloaded from our Github repository. The repository contains all of the scripts to download the original data set, and to prepare it for all databases and import it. We welcome all contributions and invite you to test other data and other workloads. We hope you will share your results and experiences. Thanks to the contributions the benchmark project received so far! Open-source is awesome 😊

Got any questions? I'm happy to help.

(/ #twitter)

(/ #linkedin)

(/ #reddit)

(/ #hacker_news)



February 14, 2018 Author Claudius Weinberger
(<https://www.arangodb.com/author/c-weinberger/>)

Claudius studied economics with business informatics as key aspect at the University of Cologne. Together with his co-founder, he builds databases for more than 20 years; from in-memory to mostly memory databases and from K/V stores over multi-dimensional cubes to graph databases. His responsibility was mostly the product and project management. Since 2012 he is the CEO of ArangoDB.

8 Comments



Posted by Deepak Manoharan on February 14, 2018 at 6:37 pm

(<https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#comment-1031>) | |

Next time would like to see a comparison with dgraph.io

Got any questions? I'm happy to help.

Posted by Kiswono Prayogo on February 23, 2018 at 3:27 am (<https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#comment-1032>) | |

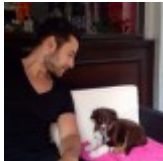


nice man



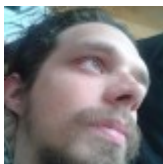
Posted by Vitor Buzinaro on February 24, 2018 at 4:42 am (<https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#comment-1033>) | |

It would be awesome if you can include Dgraph in your next benchmark !



Posted by Lazhar on March 1, 2018 at 8:24 pm (<https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#comment-1038>) | |

My favorite graph database – the team is responsive and listens to the community and well, the product is amazing so far!



Posted by ShalokShalom on March 4, 2018 at 7:09 pm (<https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#comment-1041>) | |

+1



Posted by Sunghyoun Bae on March 28, 2018 at 2:44 pm (<https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#comment-1047>) | |



Got any questions? I'm happy to help.

Very helpful for me.

Could you add Couchbase ?



Posted by Kyle on April 4, 2020 at 12:10 am (<https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#comment-1410>) | |

Do please rerun this, publish the results in a new blog post, and include couchbase!



Posted by Jan Stücke on April 9, 2020 at 6:31 pm (<https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/#comment-1413>) | |

We will try to publish an updated version again and might also take a look into Couchbase. It is just so much work to do it right and fair for every product, that it might take a bit for the next version



Leave a Reply

Your email address will not be published. Required fields are marked *

✕
Got any questions? I'm happy to help.



Comment



Name *

Email *

Website

☐ Je ne suis pas un robot

reCAPTCHA
Confidentialité - Conditions

Post Comment

✕
Got any questions? I'm happy to help.



Get the latest tutorials, blog posts and news:

Subscribe now

Tags

API (<https://www.arangodb.com/tag/api/>)

AQL (<https://www.arangodb.com/tag/aql/>)

arangoml (<https://www.arangodb.com/tag/arangoml/>)

arangosearch (<https://www.arangodb.com/tag/arangosearch/>)

arangosh (<https://www.arangodb.com/tag/arangosh/>)

Benchmark (<https://www.arangodb.com/tag/benchmark/>)

Cloud (<https://www.arangodb.com/tag/cloud/>)

Cluster (<https://www.arangodb.com/tag/cluster/>)

CNFC (<https://www.arangodb.com/tag/cnfc/>)

Got any questions? I'm happy to help.

Community (<https://www.arangodb.com/tag/community-2/>)



Documentation (<https://www.arangodb.com/tag/documentation/>)

driver (<https://www.arangodb.com/tag/driver/>)

Enterprise (<https://www.arangodb.com/tag/enterprise/>)

Foxx (<https://www.arangodb.com/tag/foxx/>)

geo (<https://www.arangodb.com/tag/geo/>)

GitHub (<https://www.arangodb.com/tag/github/>)

Graph (<https://www.arangodb.com/tag/graph/>)

HTTP (<https://www.arangodb.com/tag/http/>)

Java (<https://www.arangodb.com/tag/java/>)

JavaScript (<https://www.arangodb.com/tag/javascript/>)


Kubernetes (<https://www.arangodb.com/tag/kubernetes/>)

Machine Learning (<https://www.arangodb.com/tag/machine-learning/>)

Managed Service (<https://www.arangodb.com/tag/managed-service/>)

ML (<https://www.arangodb.com/tag/ml/>)

monitoring (<https://www.arangodb.com/tag/monitoring/>)

Got any questions? I'm happy to help. 

multi-model (<https://www.arangodb.com/tag/multi-model/>)



NoSQL (<https://www.arangodb.com/tag/nosql/>)

open-source (<https://www.arangodb.com/tag/open-source/>)

Performance (<https://www.arangodb.com/tag/performance/>)

PHP (<https://www.arangodb.com/tag/php/>)

plugin (<https://www.arangodb.com/tag/plugin/>)

Pregel (<https://www.arangodb.com/tag/pregel/>)

python (<https://www.arangodb.com/tag/python/>)

Release (<https://www.arangodb.com/tag/release/>)

RocksDB (<https://www.arangodb.com/tag/rocksdb/>)

Ruby (<https://www.arangodb.com/tag/ruby/>)

storage (<https://www.arangodb.com/tag/storage/>)

testing (<https://www.arangodb.com/tag/testing/>)

Tutorial (<https://www.arangodb.com/tag/tutorial/>)

video (<https://www.arangodb.com/tag/video/>)

✕
Got any questions? I'm happy to help.

WebUI (<https://www.arangodb.com/tag/webui/>)

Windows (<https://www.arangodb.com/tag/windows/>)



Win a t-shirt! Survey »

(/community-survey/)

Got any questions? I'm happy to help.




Social Links

Quick Links

Info

Meet
ArangoDB



[Quick Start](#)
[Subscriptions](#)
[About us](#)

[Roadmap](#)
[Community](#)
[Jobs](#)

[Download/Drivers](#)
[Events](#)
[Imprint](#)

[Case Studies](#)
[Online Meetup](#)
[Contact](#)

[Bi Connectors](#)
[Logo/Resources](#)
[News](#)

[Certification Exam](#)
[Win a t-shirt](#)

[Keep up with the latest news from the ArangoDB database](#)

Your data is your data.
We will use it respectfully
according to the terms of
our [Privacy Policy](#)
(<https://www.arangodb.com/data-privacy-policy/>).

© 2020 ArangoDB, Inc.
411 Borel Ave. Suite 512,
San Mateo, CA 94402,
United States

Got any questions? I'm happy to help.



9,567

Star ArangoDB on GitHub

(<https://github.com/arangodb/arangodb/stargazers>)



Got any questions? I'm happy to help.