Journal of King Saud University – Computer and Information Sciences (2017) 29, 520–525

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

A performance evaluation of in-memory databases

Abdullah Talha Kabakus [a,*], Resul Kara [b]

[a] Abant Izzet Baysal University, IT Center, 14030 Bolu, Turkey
[b] Duzce University, Faculty of Engineering, Department of Computer Engineering, 81620 Duzce, Turkey

Received 1 April 2016; revised 6 May 2016; accepted 29 June 2016
Available online 2 July 2016

KEYWORDS

NoSQL databases;
In-memory databases;
Database performance

Abstract The popularity of NoSQL databases has increased due to the need of (1) processing vast amount of data faster than the relational database management systems by taking the advantage of highly scalable architecture, (2) flexible (schema-free) data structure, and, (3) low latency and high performance. Despite that memory usage is not major criteria to evaluate performance of algorithms, since these databases serve the data from memory, their memory usages are also experimented alongside the time taken to complete each operation in the paper to reveal which one uses the memory most efficiently. Currently there exists over 225 NoSQL databases that provide different features and characteristics. So it is necessary to reveal which one provides better performance for different data operations. In this paper, we experiment the widely used in-memory databases to measure their performance in terms of (1) the time taken to complete operations, and (2) how efficiently they use memory during operations. As per the results reported in this paper, there is no database that provides the best performance for all data operations. It is also proved that even though a RDMS stores its data in memory, its overall performance is worse than NoSQL databases.
© 2016 The Authors. Production and hosting by Elsevier B.V. on behalf of King Saud University. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

The key reasons behind regarding "data storage mechanism" as the hearth of enterprise software systems can be listed as: (1) it is the most major part of softwares that determines how quick an application responds a request, and (2) the loss of data is mostly unacceptable since the key business operations. Until the rise of NoSQL (Not-only SQL) databases, the relational database management systems (RDMS') were the sole and exclusive remedy. However, with the constant growth of stored data, the limitations of relational database management systems such as scalability and storage, and efficiency losing of query due to the large volumes of data, and the storage and management of larger databases become challenging (Abramova et al., 2014). At the time of writing, there exists over 225 NoSQL databases that provide different features and characteristics (Edlich, 2016). NoSQL databases are more horizontally scalable and flexible when they are compared to RDMS' (Stonebraker, 2010). When it comes to processing vast amounts of data quickly taking the advantage of schema-free data structure and distributed architecture, NoSQL databases are preferred instead of RDMS' (Bartholomew, 2010; Li and

* Corresponding author. Fax: +90 3742534526.
E-mail address: talha.kabakus@ibu.edu.tr (A.T. Kabakus).
Peer review under responsibility of King Saud University.

ELSEVIER Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2016.06.007
1319-1578 © 2016 The Authors. Production and hosting by Elsevier B.V. on behalf of King Saud University.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Manoharan, 2013). Also, performance of RDMS' decrease with increase in size of data, which causes deadlocks and concurrency issues (Han et al., 2011). While RDMS relies on ACID (Atomicity, Consistency, Isolation, Durability) consistency model that ensures all the transactions are correctly committed and do not corrupt database, and the data are consistent, NoSQL databases are based on BASE (Basically Available, Soft-state, Eventually Consistent) consistency model in order to achieve scalability, high availability, and high performance (Bartholomew, 2010; Carro, 2014; Cook, 2009; Gajendran, 2012; Pritchett, 2008). NoSQL databases serves the data from volatile memory (i.e. random access memory – RAM) instead of non-volatile memory (i.e. hard drive) in order to increase the speed of querying since I/O (Input/Output) data access is slow (Abramova et al., 2014).

The rest of the paper is organized as follows: Section 2 describes categories of in-memory databases and their differences. Section 3 presents related works. Section 4 discusses the proposed experimental setup. Section 5 presents the experimental results and discussion. Finally, Section 6 concludes the paper.

2. NoSQL databases

NoSQL databases can be categorized into four classes according to different optimizations (Indrawan-Santiago, 2012):

- Key-value store: The data are stored as key-value pairs. This data structure is also known as "hash table" where the data are retrieved by keys. Most well-known examples of key-value stores are *Redis*[1], *Memcached*[2].
- Document store: The data are stored in collections that contain key-value pairs which encapsulate key value pairs in JSON (Javascript Object Notation) or JSON like documents (Hecht and Jablonski, 2011). Most well-known examples of document stores are *MongoDB*[3], *CouchDB*[4]. Since values are not opaque to the system, data can be queried by values as well as keys (Hecht and Jablonski, 2011).
- Column family: The data are stored as a set of rows and columns where columns are grouped according to the relationship of data (Abramova et al., 2014). Most well-known examples of document stores are *Cassandra*[5], *HBase*[6].
- Graph database: This type of databases is best used to represent data in the form of graph. The most well-known example of graph databases is *Neo4j*[7].

3. Related works

Bartholomew (Bartholomew, 2010) compares SQL and NoSQL databases with providing a brief history and the use case of each one. Tiwari provides a detailed introduction on NoSQL databases with a comparison on the basis of following features: (1) scalability, (2) transactional integrity and consistency, (3) data modeling, (4) query support, and (5) access and interface availability (Tiwari, 2011). Hecht and Jablonski present a use case oriented survey on NoSQL databases (Hecht and Jablonski, 2011). They compare NoSQL databases by their data models, query possibilities, concurrency controls, partitioning and replication opportunities.

Abramova et al. (2014) use Yahoo! Cloud Serving Benchmark (Cooper et al., 2010) in order to evaluate and compare the performance of NoSQL databases. They randomly generate 600,000 records and used them with different workloads by changing ratios of *read*, *update* and *insert* operations. The databases used in the experimental evaluation are *Redis*, *Cassandra*, *HBase*, *MongoDB*, and *OrientDB*[8]. They report that as overall the in-memory database *Redis* provides the best performance. Also, they report that column family databases *Cassandra* and *HBase* showed good update performance since they are optimized for *update* operations.

Li and Manoharan (2013) compare performances of NoSQL databases through five experiments: (1) Time to instantiate database bucket, (2) time to read values corresponding to given keys, (3) time to write key-value pairs, (4) time to delete key-value pairs, and (5) time to fetch all keys. These experiments are also tested for various data from 10 records to 100,000 records. The databases they tested are *MongoDB*, *RavenDB*[9], *CouchDB*, *Cassandra*, *Hypertable*[10], *Couchbase*[11], and *MS SQL Express*[12]. They report that *Couchbase* and *MongoDB* are the fastest two overall for *read*, *write*, and *delete* operations. They also note that *Couchbase* lacks fetching all keys from database.

Boicea et al. (2012)) compare *MongoDB* and *Oracle*[13] databases in order to compare NoSQL and SQL database performance through the three experiments: (1) Elapsed time to insert data, (2) elapsed time to update data, and (3) elapsed time to delete data. These experiments are also tested for various data from 10 records to 1,000,000 records. They report that for all operations, *MongoDB* provides better performance than *Oracle*.

Our contribution in this paper is developing our own software to measure performance of widely used in-memory databases for various experiments. Despite that memory usage is not a major criterion to evaluate performances of algorithms, since these databases serve the data from memory, it is necessary to reveal their memory usages especially when the size of data gets bigger. For this reason, unlike the related works, we also dig into the memory usages of in-memory databases alongside their performances in term of the time taken to complete different database operations.

4. Experimental setup

The in-memory databases that are experimented in this paper are listed in Table 1 with their database models and versions. There exists at least one database from each NoSQL database category (key-value store, document store, column family,

[1] http://redis.io.
[2] https://memcached.org.
[3] https://www.mongodb.org.
[4] http://couchdb.apache.org.
[5] http://cassandra.apache.org.
[6] https://hbase.apache.org.
[7] http://neo4j.com.
[8] http://orientdb.com.
[9] https://ravendb.net.
[10] http://www.hypertable.org.
[11] http://www.couchbase.com.
[12] https://www.microsoft.com/en-us/server-cloud/products/sql-server/overview.aspx.
[13] https://www.oracle.com/database/index.html.

Table 1 The in-memory databases that are experimented in this paper.

Database name	Database model	Version
MongoDB	Document store	3.2.4
Redis	Key-value store	3.0.7
Memcached	Key-value store	1.4.14
Cassandra	Column store	2.2.5
H2	Relational database	1.4.191

graph database) except graph database since it was discussed by Armstrong et al. (2013), graph databases cannot be compared with other NoSQL databases using the same experiments. All the databases used with the experimental setup are NoSQL databases except *H2*. We intentionally included *H2* into this list despite that it is a RDMS, *H2* differs from other RDMS by storing its data in the memory instead of hard disk. Therefore, the experimental results reveal the effect of the database model on the performance of database.

We developed our own software based on Java to measure performance of in-memory databases for various experiments. Execution time per operation is recorded by determining system time at the start and end of the method using *java.lang.System* class. Similarly, consumed memory per operation is recorded by determining the free memory in bytes at the start and end of the method using the *java.lang.Runtime* class which allows Java applications to interface with the environment that they run (Ricca, 2003). The execution time and consumed memory are also calculated in this way by Bergmann et al. (2010). During the experiments, all other processes of operating system (except the mandatory ones) are stopped in order to reveal sole performance of databases. All the experiments are evaluated on the same machine whose specifications are described in the Table 2.

5. Experimental results and discussion

The performance of in-memory databases is measured by four experiments: (1) performance to write a key-value pair, (2) performance to read value corresponding to a given key, (3) performance to remove the key-value pair corresponding to a given key, and (4) performance to get all the data. For each experiment, the size of data is exponentially increased in order to reveal how the size of data affects performance of each database.

5.1. Experiment 1 – performance to write a key-value pair

A service that generates random key-value pairs is implemented in order to measure write performance of in-memory databases. The calculated time to write the generated key-value pairs per each database is listed in Table 3. As it is shown in the result, the list of databases can be sorted by overall performance of *write* operation: *Memcached*, *H2*, *Redis*, *Cassandra*, *MongoDB*. It should be noted that there is a significant difference between *Memcached* and *MongoDB*. As the size of data increases, the performance differences between databases become evident. Performance of *MongoDB* significantly decreases when the size of the data increases since *MongoDB* uses locking mechanism as it is discussed by Abramova et al. (2014).

Since in-memory databases serve the data from memory, it is necessary to identify how much memory they consume during write operations. Table 4 presents memory usage of each database for *write* operation. As it is shown in the result, unlike the elapsed time result, *Redis* provides the best performance when it comes to efficient memory usage.

Table 2 Specifications of the machine that is used to evaluate experiments.

Operating system	Ubuntu 14.04 (64-bit)
Memory	16 GB
CPU	Intel Core i7-4710MQ 4-Cores; 6 MB L3; 2.50 GHz > 3.50 GHz
Java version	1.8.0_60
File system	ext4

Table 3 The calculated time to *write* key-value pairs (ms).

Database	Number of records			
	1,000	10,000	100,000	1,000,000
Redis	34	214	1666	14.638
MongoDB	904	3482	26.030	253.898
Memcached	23	100	276	2813
Cassandra	1202	4487	15.482	140.842
H2	147	475	1648	7394

5.2. Experiment 2 – performance to read value corresponding to a given key

Our second experiment measures the required time and consumed memory to read the value corresponding to the given key. As the experimental result is listed in Table 5, *H2* provides

Table 4 Memory usages of in-memory databases for *write* operation (MB).

Database	Number of records			
	1,000	10,000	100,000	1,000,000
Redis	2.5	3.8	4.3	62.7
MongoDB	56.9	263.6	365	155.9
Memcached	5.3	27.2	211	264.9
Cassandra	5.3	7.5	208.1	102.6
H2	2.3	33.2	103.4	540

Table 5 The elapsed time to *read* value corresponding to a given key per database (ms).

Database	Number of records			
	1,000	10,000	100,000	1,000,000
Redis	8	6	8	8
MongoDB	8	10	11	13
Memcached	9	14	14	30
Cassandra	2	2	3	6
H2	25	26	60	171

the worst *read* performance in terms of elapsed time. Despite that *H2* also stores the data on memory like the other databases, the architecture of database which is relational database management system decreases the performance of *read* operation. As it is shown in the result, the list of databases can be sorted by overall performance of read operation: *Cassandra*, *Redis*, *Memcached*, *MongoDB*, *H2*.

Cassandra provides the best *read* performance when it comes to efficient memory usage. There is no significant difference in memory usages between other databases except *H2*. When the size of data increases, the *read* performance of *H2* dramatically decreases (see Table 6).

5.3. Experiment 3 – performance to delete key-value pair corresponding to a given key

Our third experiment measures the required time to delete the data corresponding to a given key. As Table 7 summarizes the result, *Redis* clearly provides the best performance by completing operations in less than 1 ms while *MongoDB* provides the worst performance. Performance of *Cassandra* for *delete* operation is calculated as very similar to *Redis*.

When it comes to efficient memory usage, *Cassandra* and *Redis* provide better performance compared to other in-memory databases, and *H2* provides the worst for *delete* operation (see Table 8).

5.4. Experiment 4 – performance to fetch all the data

Our forth experiment measures database performance while fetching whole data. This experiment differs from the experiment 2 since the read query ends as soon as the data corresponding to the given key is found. Despite that the key is randomly chosen during the experiment 2, we think that fetching the whole data makes these measures clearer: (1) how fast is the database while fetching all the data available, and (2) how much memory does the database consume in order to store and fetch the whole data. As the result is listed in Tables 9 and 10, *MongoDB* clearly provides the best performance with consuming a lot less memory compared to *Redis* and *H2*. As the size of data increases, the performance difference between *MongoDB* and others becomes evident. *Memcached* is excluded for this experiment since it does not support fetching whole the data (Stackoverflow, 2016).

According to CAP (Consistency, Availability, Partition Tolerance) theorem, which is the base of both ACID and BASE consistency models, only two of three guarantees can be achieved (Han et al., 2011; Rahman et al., 2015). While RDMS mainly focus on consistency, the main idea behind NoSQL databases is giving up some consistency in order to provide more availability, scalability, and high performance (Bartholomew, 2010). Another advantage of using RDMS is that since they use a common language (which is SQL – Structured Query Language), migration from one relational database management system to another is always more possible when it is compared to NoSQL databases which have its own set of APIs to interact the data they contain (Bartholomew, 2010).

Table 6 Memory usages of in-memory databases for *read* operation (MB).

Database	Number of records			
	1,000	10,000	100,000	1,000,000
Redis	1.3	1.3	1.3	1.3
MongoDB	1.3	2.5	2.5	1.3
Memcached	1.3	2.5	1.3	2.5
Cassandra	0	0	0	0
H2	1.2	1.2	9.8	60

Table 7 The elapsed time to *delete* the data corresponding to a given key per database (ms).

Database	Number of records			
	1,000	10,000	100,000	1,000,000
Redis	0	1	0	0
MongoDB	75	88	92	355
Memcached	17	17	16	13
Cassandra	2	2	1	2
H2	10	13	68	174

Table 8 Memory usages of in-memory databases for *delete* operation (MB).

Database	Number of records			
	1,000	10,000	100,000	1,000,000
Redis	0	0	0	0
MongoDB	10	10	10	11.3
Memcached	2.2	2.1	2.2	2.2
Cassandra	0	0	0	0
H2	1.2	1.8	4.9	62.9

Table 9 The elapsed time to fetch all the data per database (ms).

Database	Number of records			
	1,000	10,000	100,000	1,000,000
Redis	10	9	11	11
MongoDB	9	15	9	8
Cassandra	9	32	24	54
H2	9	14	18	20

Table 10 Memory usages of in-memory databases to get all the data (MB).

Database	Number of records			
	1,000	10,000	100,000	1,000,000
Redis	2.1	2.1	2.2	2.2
MongoDB	1.3	1.3	1.1	0.8
Cassandra	0.6	0.6	1.3	1.3
H2	1.2	1.1	9.7	10.6

6. Conclusion

NoSQL databases are based on BASE consistency model instead of ACID consistency model which comes with the idea of giving up some consistency in order to provide more availability, scalability, and high performance. The popularity of NoSQL databases – which store the data in memory – has increased due to the need of (1) processing vast amount of data faster than the relational database management systems by taking the advantage of highly scalable architecture, (2) flexible (schema-free) data structure, and, (3) low latency and high performance. Currently there exists over 225 NoSQL databases that provide different features and characteristics. In this paper, we evaluate at least one in-memory database from each type: *Redis* and *Memcached* as key-value stores, *MongoDB* as a document store, *Cassandra* as column family, and *H2* as an in-memory relational database management system. Unlike the related works, alongside the time taken to complete various data operations, the memory usages of in-memory databases are also experimented in order to reveal memory usages of each database. Results obtained from experiments can be listed as:

- While *Memcached* clearly provides the best *write* performance in term of elapsed time, *Redis* uses the memory more efficiently than others. Performance of *MongoDB* significantly decreases when the size of the data increases due to locking mechanism of *MongoDB*.
- *Redis* fairly provides better performance than *Memcached* and *MongoDB* for the *read* operation. *H2* clearly provides the worst *read* performance. We think that the architecture of *H2* which is relational database is the main reason behind this difference.
- When it comes to delete a key-value pair corresponding to a given key, *Redis* clearly provides the best performance while *MongoDB* provides the worst performance. *Redis* and *Cassandra* consume the memory more effective than other databases for *delete* operation. *H2* uses the memory less efficiently than others and the difference becomes evident when the size of data increases.
- *MongoDB* provides significantly the best performance to fetch the whole data while *Cassandra* provides the worst performance. The disadvantage of using *H2* is that since it stores the data in the memory, it is volatile. *Redis* and *MongoDB* (*when the indexes are created for the queries*) serve the data from memory while the hard disk is used for storage. When it comes to memory consumption to fetch whole the data, *MongoDB* uses the memory more effective than others. *H2* uses the memory much more than others especially when the size of data increases. *Memcached* does not support fetching the whole data.

SQL and NoSQL databases provide different characteristics and one cannot replace another. If the system is not flexible in terms of consistency, then the relational database management system is the right choice. If the system can give up some consistency, then NoSQL databases can be a good choice in order to provide more availability, scalability, and high performance.

As future work, the effect of document types on performance for each database can be experimented. Also, reasons of latencies that are experienced during the experiments can be explained in detail by inspecting their architectures. Effect of distributed and parallel environments on database performances is another topic to be examined.

References

Abramova, V., Bernardino, J., Furtado, P., 2014. Which NoSQL database? A performance overview. Open J. Databases 1, 17–24.

Armstrong, T.G., Ponnekanti, V., Borthakur, D., Callaghan, M., 2013. LinkBench: a database benchmark based on the Facebook social graph. In: SIGMOD '13 Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, New York, NY, USA, pp. 1185–1196. http://dx.doi.org/10.1145/2463676.2465296.

Bartholomew, D., 2010. SQL vs. NoSQL. Linux J., 54–59 (doi: FacetMap: A Scalable Search and Browse Visualization)

Bergmann, G., Horváth, Á., Ráth, I., Varró, D., Balogh, A., Balogh, Z., Ökrös, A., 2010. Incremental evaluation of model queries over EMF models. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (Eds.), Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp. 76–90. http://dx.doi.org/10.1007/978-3-642-16145-2.

Boicea, A., Radulescu, F., Agapin, L.I., 2012. MongoDB vs oracle – database comparison. In: Proceedings – 3rd International Conference on Emerging Intelligent Data and Web Technologies, EIDWT 2012, pp. 330–335. http://dx.doi.org/10.1109/EIDWT.2012.32 (Bucharest, Romania).

Carro, M., 2014. NoSQL Databases. CoRR abs/1401.2.

Cook, J.D., 2009. ACID versus BASE for database transactions [WWW Document]. URL <http://www.johndcook.com/blog/2009/07/06/brewer-cap-theorem-base/> (accessed 29.03.16).

Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R., 2010. Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM Symposium on Cloud Computing – SoCC '10. ACM, New York, NY, USA, pp. 143–154. http://dx.doi.org/10.1145/1807128.1807152.

Edlich, S., 2016. NOSQL Databases [WWW Document]. URL <http://nosql-database.org> (accessed 28.03.16).

Gajendran, S.K., 2012. A Survey on NoSQL databases..

Hecht, R., Jablonski, S., 2011. NoSQL evaluation: a use case oriented survey. In: Proceedings – 2011 International Conference on Cloud and Service Computing, CSC 2011, pp. 336–341. http://dx.doi.org/10.1109/CSC.2011.6138544.

Indrawan-Santiago, M., 2012. Database research: are we at a crossroad? Reflection on NoSQL. In: Proceedings of the 2012 15th International Conference on Network-Based Information Systems, NBIS 2012, Melbourne, Australia, pp. 45–51. http://dx.doi.org/10.1109/NBiS.2012.95.

Jing, Han, Haihong, E., Guan, Le, Jian, Du, 2011. Survey on NoSQL database. In: 2011 6th International Conference on Pervasive Computing and Applications. IEEE, Port Elizabeth, South Africa, pp. 363–366. http://dx.doi.org/10.1109/ICPCA.2011.6106531.

Li, Y., Manoharan, S., 2013. A performance comparison of SQL and NoSQL databases. IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing – Proceedings., 15–19 http://dx.doi.org/10.1109/PACRIM.2013.6625441.

Pritchett, D., 2008. Base: an acid alternative. Queue 6, 48–55. http://dx.doi.org/10.1145/1394127.1394128.

Rahman, M.R., Tseng, L., Nguyen, S., Gupta, I., Vaidya, N., 2015. Characterizing and adapting the consistency-latency tradeoff in distributed key-value stores. Distributed, Parallel, and Cluster Computing.

Ricca, F., 2003. The DLV java wrapper. In: Proceedings ASP 2003 – Answer Set Programming: Advances in Theory and Implementation. Messina, Italy, pp. 305–316.

stackoverflow, 2016. retrieve all objects key store in memcached in java – Stack Overflow [WWW Document]. URL <http://stackoverflow.com/questions/8487153/retrieve-all-objects-key-store-in-memcached-in-java> (accessed 30.03.16).

Stonebraker, M., 2010. SQL databases v. NoSQL databases. Commun. ACM 53, 10–11. http://dx.doi.org/10.1145/1721654.1721659.

Tiwari, S., 2011. Professional NoSQL, 1st ed. John Wiley & Sons, Indianapolis, Indiana.