# A performance comparison of SQL and NoSQL databases

**Conference Paper** · August 2013

DOI: 10.1109/PACRIM.2013.6625441



**2 authors**, including:

Yishan Li
University of Auckland
**1** PUBLICATION **200** CITATIONS

# A performance comparison of SQL and NoSQL databases

Yishan Li and Sathiamoorthy Manoharan
Department of Computer Science
University of Auckland
New Zealand

***Abstract*—With the current emphasis on "Big Data", NoSQL databases have surged in popularity. These databases are claimed to perform better than SQL databases. In this paper we aim to independently investigate the performance of some NoSQL and SQL databases in the light of key-value stores. We compare read, write, delete, and instantiate operations on key-value stores implemented by NoSQL and SQL databases. Besides, we also investigate an additional operation: iterating through all keys. An abstract key-value pair framework supporting these basic operations is designed and implemented using all the databases tested. Experimental results measure the timing of these operations and we summarize our findings of how the databases stack up against each other. Our results show that not all NoSQL databases perform better than SQL databases. Some are much worse. And for each database, the performance varies with each operation. Some are slow to instantiate, but fast to read, write, and delete. Others are fast to instantiate but slow on the other operations. And there is little correlation between performance and the data model each database uses.**

***Keywords*-Database performance, SQL, NoSQL databases.**

## I. Introduction

Traditional database systems for storage have been based on the relational model. These are widely known as SQL databases named after the language they were queried by [1]. In the last few years, however, non-relational databases have dramatically risen in popularity. These databases are commonly known as NoSQL databases, clearly marking them different from the traditional SQL databases. Most of these are based on storing simple key-value pairs on the premise that simplicity leads to speed.

With the increase in accessibility of Internet and the availability of cheap storage, huge amounts of structured, semi-structured, and unstructured data are captured and stored for a variety of applications. Such data is commonly referred to as *Big Data* [2]. Processing such vast amount of data requires speed, flexible schemas, and distributed (i.e. non-centralized) databases. NoSQL databases became the preferred currency for operating Big Data they claim to satisfy these requirements. This also lead to a surge in the number of NoSQL database offerings. There are several commercial and open-source implementations of NoSQL databases (such as BigTable [3] and HBase [4]).

The large number of NoSQL offerings then leads to questions on differences between these offerings and their suitability in particular applications. A number of survey papers (such as that of Tudorica and Bucur [5] or Han et al. [6]) have therefore been published to address some of these questions. In addition, there are several online resources and blogs addressing these aspects as well.

The focus of our paper is to compare the key-value stores implementations on NoSQL and SQL databases. While NoSQL databases are generally designed for optimized key-value stores, SQL databases are not. Yet, our findings suggest that not all NoSQL databases perform better than SQL databases. We compare read, write, delete, and instantiate operations on the key-value storage. We observe that even within NoSQL databases there is a wide variation in the performance of these operations. We also observe little correlation between performance and the data model each database uses.

The rest of this paper is organized as follows. Section II introduces a selection of NoSQL databases. Section III reviews related work. In particular, it looks at other surveys comparing NoSQL offerings. Section IV discusses our experimental setup and what we evaluate. Section V presents and discusses our experimental results. The final section concludes with a summary.

## II. NoSQL Databases

It is widely believed that Google's BigTable [3] was the first of the NoSQL databases. BigTable is based on three keys: first one called the row key, second called the column key, and the third one is the timestamp. This is effectively a multidimensional map. Column keys can be classified into groups. A group is accessed as a single unit.

The success of proprietary non-relational databases such as BigTable and Amazon's Dynamo [7] initiated a number of other open-source and closed-source non-relational database developments. These NoSQL databases grew in popularity because of the ease of access, speed, and scalability.

Most of the NoSQL databases are based on storing key-value pairs. It is possible that the values could be a set of secondary keys which in turn contain values.

A special type of a key-value pair database is a column-family database. This consists of columns and super columns, addressable with keys. A super column groups a number of related columns and is accessed as a single unit.

The other special type of key-value pair database is a document-oriented database. Document-oriented databases go beyond simple values and have the ability to store objects. The


978-1-4799-1501-9/13/$31.00 

objects are in some serialized form such as XML, JSON, and BSON (binary encoded JSON).

In this paper we look at a selection of NoSQL databases in the view of comparing them against each other and against Microsoft SQL. The remainder of this section briefly describes the NoSQL databases we have selected for comparison.

MongoDB[1] is a document-oriented database written in C++ [8]. The objects are stored serialized as BSON. The objects do not need to have the same structure or fields and the common fields do not need to have the same type, thus allowing a flexible schema storage. MongoDB supports auto-sharding where it partitions the data collections and stores the partitions across available servers. This results in a dynamically balanced load.

Hypertable[2] is an open-source NoSQL database based on Google's BigTable [3]. It is a column-family database and is written in C++. Just like MongoDB, Hypertable too supports auto-sharding.

Apache CouchDB is a document-oriented database written in Erlang [9]. The objects are stored serialized as JSON. The database is accessed through a RESTful HTTP API. CouchDB is therefore claimed to be a "database that completely embraces the web"[3].

Apache Cassandra[4] is a column-family database and was originally developed by Facebook [10]. Some of its key features include de-centralization to reduce failures and data replication to increase fault-tolerance.

RavenDB[5] and Couchbase[6] are two other document-oriented databases we consider [11]. Like MongoDB and CouchDB, these too offer flexible schema storage and use JSON as its format for serialized objects. They also offer data replication and sharding.

## III. Related Work

While NoSQL databases have the speed and scalability advantage, there have a number of drawbacks compared to traditional relational databases. Leavitt lists these challenges [12]. He notes that NoSQL databases, even though fast for simple tasks, are time-consuming for complex operations. Besides queries for complex operations can be hard to form. The other drawback is the lack of native support for consistency. Leavitt also notes that NoSQL is a technology that many organizations are yet to learn and there is a lack of support and management tools to help.

Bartholomew gives a tutorial introduction to the history of and differences between SQL and NoSQL databases [13]. Sakr et al. discuss data management solutions, including NoSQL, for cloud-based platforms [14]. They discuss the challenges data management solutions face in the light of the cloud.

[1]www.mongodb.org
[2]www.hypertable.com
[3]couchdb.apache.org
[4]cassandra.apache.org
[5]ravendb.net
[6]www.couchbase.com

Tiwari provides a comprehensive treatise on NoSQL databases [15]. He covers the history, rationale, programmability, storage architecture, and performance-tuning of some of the implementations. He also notes the following as the basis of comparing implementations.

- Scalability
- Consistency
- Support for data models
- Support for queries, and
- Management tools

Similarly, Indrawan-Santiago notes the following as the basis of comparison: data model, transaction model, support for ad-hoc queries, indexing, shrading, and license type [16]. She compares ten NoSQL implementations including Cassandra, HBase, Dynamo, MongoDB and CouchDB on this basis. She also qualitatively compares relational databases to NoSQL databases, and concludes that NoSQL are likely to complement relational databases and enhance an organization's enhance database management capabilities.

Hecht and Jablonski provide a use-case oriented survey of NoSQL databases [17]. They identify the difficulties in choosing a NoSQL database to fit a particular use-case, and therefore focus their paper to address this. They use as the basis for their comparison the data model, support for queries, partitioning, replication, and concurrency controls. They compare in this light fourteen NoSQL databases, including MongoDB, CouchDB, Cassandra and HBase.

Boicea et al. compare a NoSQL database against a SQL database. They choose Oracle for the SQL implementation and MongoDB for the NoSQL implementation [18]. They report that, with a large number of records, insertion time is a factor more in Oracle and update and delete times are several factors more in Oracle.

Yahoo! Cloud Serving Benchmark is an open-source workload generator tool for comparing key-value stores [19].

Our contribution in this paper is a practical take on the comparison of the databases. This is based on some of the fundamental operations we undertake on databases. We outline these operations in the following section.

We do not compare architectural details such as supported data models, consistency, fault-tolerance, etc. These details are well-covered in other reviews [15], [16], [17].

## IV. Experimental Framework

Our comparison of the databasese involve four fundamental operations:

1) Instantiate. This instantiates a storage bucket for the key-value pairs.
2) Read. This reads the value corresponding to a given key from the key-value pair storage. This is the same as the *Read* operation of the CRUD (Create, Read, Update, Delete) model commonly used to describe database key database operations.
3) Write. If a given key-value pair is not found in the storage, then the pair is added to the storage. Otherwise,

it updates the value for the given key in the storage. This operation therefore combines *Create* and *Update* operations of the CRUD model.

4) Delete. This deletes the record (i.e. key-value pair) corresponding to a given key from the key-value pair storage. This is the same as the *Delete* operation of the CRUD model.

In addition to the above fundamental operations, there are two supplementary operations that are commonly used: iterating through all the keys and iterating through all values. To enable testing these, we time one more operation: *GetAllkeys* which fetches all the keys from the strorage. Note that, in practice, this operation may only return a specified number of keys (rather than all) when there are a very large number of key-value pairs in the storage.

```
abstract public class KeyValueStoreBase
{
        abstract public bool Instantiate(string bucketName);
        abstract public string Read(string key);
        abstract public bool Write(string key, string value);
        abstract public void Delete(string key);

        abstract public List<string> GetAllKeys();
}
```

Fig. 1. Base operations on key-value stores

Our experiments measure the performance of these five operations for the chosen databases. Figure 1 summarizes the operations as an abstract class. We specialize this abstraction for each database under test.

Note that there are advanced operations that databases support based on their type (for instance, object or column manipulation). We do not test these in this paper.

| Database | Version |
|---|---|
| MongoDB | 1.8.5 |
| RavenDB | 960 |
| CouchDB | 1.2.0 |
| Cassandra | 1.1.2 |
| Hypertable | 0.9.6 |
| Couchbase | 1.8.0 |
| Microsoft SQL Server Express | 10.50.1600.1 |

TABLE I
VERSION DETAIL OF DATABASE IMPLEMENTATIONS

Table I illustrates the version detail of the database implementations that are experimented with in this paper. For each of these databases, we run the chosen operation (such as Read or Write) five times and take the average time.

The data set for the experiment are auto-generated key-value pairs that are of the form $(k_N, v_N)$ where $N$ is a sequence number.

## V. RESULTS AND EVALUATION

Our first experiment measures the time taken to instantiate a database bucket. See Figure 2 which summarizes the results of this experiment.

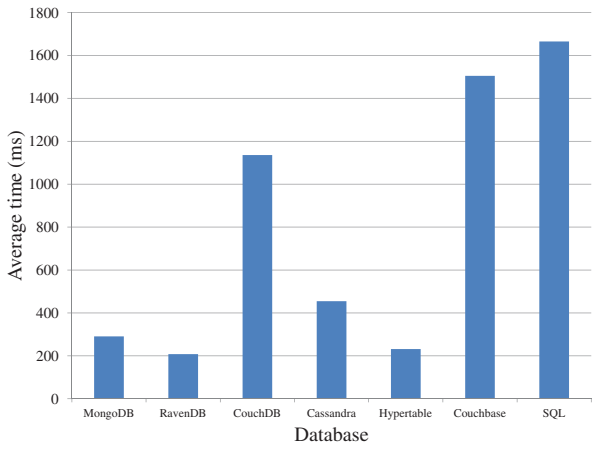


Fig. 2. Time for instantiating database bucket (ms)

Note that the times are averaged over five runs. The absolute time values are not significant; what is significant is the time values relative to one another.

We observe that RavenDB, Hyertable, MongoDB offer the fastest creation of database buckets. CouchDB, Couchbase, and SQL Express are among the slowest to create buckets.

Our second experiment measures the time taken to read values corresponding to given keys from the bucket. Table II summarizes the result.

In the tables, the number of operations refers to the number of times a given operation (such as *read*) is executed in the test. This is also equal to the number of key-value pairs (i.e. records) in the store.

| Database | Number of operations | | | | | |
|---|---|---|---|---|---|---|
| | 10 | 50 | 100 | 1000 | 10000 | 100000 |
| MongoDB | 8 | 14 | 23 | 138 | 1085 | 10201 |
| RavenDB | 140 | 351 | 539 | 4730 | 47459 | 426505 |
| CouchDB | 23 | 101 | 196 | 1819 | 19508 | 176098 |
| Cassandra | 115 | 230 | 354 | 2385 | 19758 | 228096 |
| Hypertable | 60 | 83 | 103 | 420 | 3427 | 63036 |
| Couchbase | 15 | 22 | 23 | 86 | 811 | 7244 |
| MS SQL Express | 13 | 23 | 46 | 277 | 1968 | 17214 |

TABLE II
TIME FOR READING (MS)

Sorted by read performance we have the list of databases: Couchbase, MongoDB, SQL Express, Hypertable, CouchDB, Cassandra and RavenDB. Of these Cassandra and Hypertable are column-family databases; and Couchbase, MongoDB, CouchDB, and RavenDB are document-oriented databases. There is no observable correlation between the data model and performance. We also see that the read performance of SQL Express is better than some, but not all, of the NoSQL databases.

Our third experiment measures the time taken to write key-value pairs to the bucket. If the key-value pair already exists in the bucket, this amounts to updating the existing value.

Otherwise it amounts to adding the key-value pair to the bucket. Table III summarizes the result.

| Database | Number of operations | | | | | |
|---|---|---|---|---|---|---|
| | 10 | 50 | 100 | 1000 | 10000 | 100000 |
| MongoDB | 61 | 75 | 84 | 387 | 2693 | 23354 |
| RavenDB | 570 | 898 | 1213 | 6939 | 71343 | 740450 |
| CouchDB | 90 | 374 | 616 | 6211 | 67216 | 932038 |
| Cassandra | 117 | 160 | 212 | 1200 | 9801 | 88197 |
| Hypertable | 55 | 90 | 184 | 1035 | 10938 | 114872 |
| Couchbase | 60 | 76 | 63 | 142 | 936 | 8492 |
| MS SQL Express | 30 | 94 | 129 | 1790 | 15588 | 216479 |

TABLE III
TIME FOR WRITING (MS)

Sorted by write performance we have the list of databases: Couchbase, MongoDB, Cassandra, Hypertable, SQL Express, RavenDB, and CouchDB. We see that the write performance of RavenDB and CouchDB is worse than that of SQL Express. But other NoSQL databases perform better than SQL express.

Our fourth experiment measures the time taken to delete key-value pairs from the bucket. Table IV summarizes the result.

| Database | Number of operations | | | | | |
|---|---|---|---|---|---|---|
| | 10 | 50 | 100 | 1000 | 10000 | 100000 |
| MongoDB | 4 | 15 | 29 | 235 | 2115 | 18688 |
| RavenDB | 90 | 499 | 809 | 8342 | 87562 | 799409 |
| CouchDB | 71 | 260 | 597 | 5945 | 67952 | 705684 |
| Cassandra | 33 | 95 | 130 | 1061 | 9230 | 83694 |
| Hypertable | 19 | 63 | 110 | 1001 | 10324 | 130858 |
| Couchbase | 6 | 12 | 14 | 81 | 805 | 7634 |
| MS SQL Express | 11 | 32 | 57 | 360 | 3571 | 32741 |

TABLE IV
TIME FOR DELETING (MS)

Sorted by delete performance we have the list of databases: Couchbase, MongoDB, SQL Express,Cassandra, Hypertable, CouchDB and RavenDB. We see that the delete performance of SQL Express is better than that of all NoSQL databases but Couchbase and MongoDB.

Boicea et al. reported that, for 100000 records, insertion time is a factor more in Oracle than in MongoDB and update and delete times are several factors more in Oracle [18]. We did not observe such large performance gaps between MongoDB[7] and SQL Express in our experiments here.

Our final experiment measures the time taken to fetch all the keys in the bucket. Table V summarizes the result.

Except for CouchDB, all databases are quick to fetch the keys. SQL Express was the fastest of all. Couchbase has no API to support fetching all the keys, and thus this was excluded in the experiment.

Note that fetching all the keys is substantially faster than reading values one after the other. This difference obviously is due to the database connection overhead.

[7]Boicea et al. did not include the versions of the databases tested, and therefore we do not therefore know if our version of MongoDB is different to theirs or not.

| Database | Number of keys to fetch | | | | | |
|---|---|---|---|---|---|---|
| | 10 | 50 | 100 | 1000 | 10000 | 100000 |
| MongoDB | 4 | 4 | 5 | 19 | 98 | 702 |
| RavenDB | 101 | 113 | 115 | 116 | 136 | 591 |
| CouchDB | 67 | 196 | 19 | 173 | 1063 | 9512 |
| Cassandra | 47 | 50 | 55 | 76 | 237 | 709 |
| Hypertable | 3 | 3 | 3 | 5 | 25 | 159 |
| MS SQL Express | 4 | 4 | 4 | 4 | 11 | 76 |

TABLE V
TIME FOR FETCHING ALL KEYS (MS)

Fetching all values would take a similar amount of time to fetching all keys so long as the values are small in size (i.e. comparabable to the size of the keys).

## VI. SUMMARY AND CONCLUSION

This paper compares key-value store implementations on NoSQL and SQL databases. While NoSQL databases are generally optimized for key-value stores, SQL databases are not. Yet, we find that not all NoSQL databases perform better than the SQL database we tested. We observe that even within NoSQL databases there is a wide variation in performance based on the type of operation (such as read and write). We also observe little correlation between performance and the data model each database uses.

Of the NoSQL databases RavenDB and CouchDB do not perform well in the read, write and delete operations. Casandra is slow on read operations, but is reasonably good for write and delete operations. Couchbase and MongoDB are the fastest two overall for read, write and delete operations. Couchbase, however, does not support fetching all the keys (or values). If iterating through keys and values is not required for an application, then Couchbase will be a good choice. Otherwise one may choose MongoDB who comes the close second to Couchbase in the read, write, and delete operations.

Note that we did not test the databases for more complex operations. The database rankings we noted may not hold when it comes to complex operations.

Like any application software, NoSQL implementations go through changes and thus performance improvements and degradations are likely to happen through these changes. Consequently, one will need to compare databases not only at the application design stage but also at regular intervals to enable switching to the most suitable database implementation. Tiered software development processes would help isolate the database backend to facilitate such switching when deemed necessary.

## REFERENCES


[1] K. Kline, *SQL in a nutshell*, 3rd ed. O'Reilly Media, November 2008.
[2] P. Warden, *Big Data Glossary*. O'Reilly Media, September 2011.
[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15.
[4] L. George, *HBase: The Definitive Guide*. O'Reilly Media, August 2011.

[5] B. Tudorica and C. Bucur, "A comparison between several NoSQL databases with comments and notes," in *Roedunet International Conference (RoEduNet), 2011 10th*, june 2011, pp. 1 –5.
[6] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, oct. 2011, pp. 363 –366.
[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007.
[8] K. Chodorow and M. Dirolf, *MongoDB: The Definitive Guide*. O'Reilly Media, September 2010.
[9] J. C. Anderson, J. Lehnardt, and N. Slater, *CouchDB: The Definitive Guide*. O'Reilly Media, January 2010.
[10] E. Hewitt, *Cassandra: The Definitive Guide*. O'Reilly Media, November 2010.
[11] M. Brown, *Getting Started with Couchbase Server*. O'Reilly Media, June 2012.
[12] N. Leavitt, "Will NoSQL databases live up to their promise?" *Computer*, vol. 43, no. 2, pp. 12 –14, feb. 2010.
[13] D. Bartholomew, "SQL vs. NoSQL," *Linux Journal*, no. 195, July 2010.
[14] S. Sakr, A. Liu, D. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *Communications Surveys Tutorials, IEEE*, vol. 13, no. 3, pp. 311–336, 2011.
[15] S. Tiwari, *Professional NoSQL*. Wiley/Wrox, August 2011.
[16] M. Indrawan-Santiago, "Database research: Are we at a crossroad? Reflection on NoSQL," in *Network-Based Information Systems (NBiS), 2012 15th International Conference on*, sept. 2012, pp. 45 –51.
[17] R. Hecht and S. Jablonski, "NoSQL evaluation: A use case oriented survey," in *Cloud and Service Computing (CSC), 2011 International Conference on*, dec. 2011, pp. 336 –341.
[18] A. Boicea, F. Radulescu, and L. I. Agapin, "MongoDB vs Oracle – database comparison," in *Emerging Intelligent Data and Web Technologies (EIDWT), 2012 Third International Conference on*, sept. 2012, pp. 330 –335.
[19] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, ser. SoCC '10. ACM, 2010, pp. 143–154.