SQL and NoSQL: Comparing Modern Databases

Ryan McArthur
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
McArt046@morris.umn.edu

ABSTRACT

Choosing a database for software applications has become more difficult in recent years due to the sheer number of databases that exist and their plethora of options to choose from. This difficult choice can have a substantial effect on a project, especially one that is large in size. Since decisions such as this are very important, making a well informed choice on which database to use is crucial. Here we discuss two options for databases, Relational Database Management Systems (RDBMS) and Not Only SQL (NoSQL) with much of the focus being on NoSQL. We will also look at a comparison between a RDBMS and a NoSQL database as well as a comparison among different NoSQL databases. We find that between RDBMS and NoSQL it depends heavily as to what is being done to the data to decide the 'better' option. When looking at multiple NoSQL databases for a specific distributed framework and a specific large data set we see that throughput varied from 225 to 3200 operations per second and latency varied by factors of 4 and 5, with the highest throughput giving the highest latency.

1. INTRODUCTION

Databases are used in many aspects of our lives. Important and popular services such as online banking, video game accounts, Facebook, and many more use databases constantly. Many of these systems work very well, but recently companies like Facebook and Google have also started investing into options other than the typical RDBMS that are used by most applications. The main driving force for this is the increasing amount of data these companies are collecting. NoSQL databases are not the be all and end all but with the way data is growing in recent years NoSQL seems to be a viable option to handle it. In 2013 we had 4.4 zettabytes in the digital universe, it is estimated to reach 44 zettabytes by 2020. [2] With this insane growth in data the typical scaling up approach that RDBMS take becomes way too expensive and possibly not even possible with current technology. "While there is rarely a single 'right' answer in selecting a complex component for an application, selection of inappropriate components can be costly, reduce downstream productivity due to rework, and even lead to project cancellation." [4] Deciding on a database that works well for your application is important. With NoSQL the ability to scale out offers a way to handle the increasing data while keeping cost reasonable.

When the size of the data and the number of requests increases there comes a point when structured databases can no longer handle these loads. One solution to this problem is to shift data centers to NoSQL databases. [7] NoSQL is a type of database that can have similarities with SQL databases but often times they sacrifice something to provide more on another front. Even further there are a wide range differences among NoSQL databases. Some will sacrifice more than others, and some will sacrifice different things than others. When selecting the database for an application one needs to look at the properties of the data to see if it will work well with a SQL database. If the answer is no, exploring different databases within the scope of NoSQL may be an option for that data. The differences between NoSQL databases really does matter and one may fit the data really well while another is terrible at handling that data. When selecting a database, research and testing are the most important things. As mentioned before there is not a definitive right answer. There are resources out there to help one decide on the best database for certain sets of data and applications. Some popular applications that specifically use NoSQL databases are things such as Facebook Messenger, Electronic Arts Simpson's Game App, Google Maps, and Google Earth.

Throughout the rest of this paper we will be going over a little background in Section 2 followed by some features of RDBMS in Section 3, then some of the features NoSQL has in Section 4. Finally we will look at a few performance comparisons in Section 5 , two between RDBMS and NoSQL and another comparing different NoSQL databases.

2. BACKGROUND

Most people's ideas of databases are Relational Database Management Systems (RDBMS). RDBMS are database management systems that are based on the relational model introduced by E.F . Codd in 1971. [10] Most RDBMS use the Structured Query Language or SQL to access the database. Popular RDBMS include MySQL, Microsoft SQL Server, and SQLite. Most databases used throughout business are RDBMS. The shift towards something like NoSQL only recently started with tech giants like Amazon and Google

UMM CSci Senior Seminar Conference, November 2016 Morris, MN.

needing something to handle their large amounts of data. The concept of a NoSQL-style database has existed since the 1960's but the term NoSQL became popular in the mid to late 2000's.

Often times when data needs to be searched often or quickly, items will be indexed. Indexing is used to increase speed and performance with searches. Each item is assigned its own unique identifier which allows for easy search instead of parsing through all the information until what is being looked for is found. Indexes will take up more storage but in some cases the extra storage is worth the performance increase.

3. FEATURES OF RDBMS

3.1 ACID

RDBMS support the ACID properties on transactions of their database. ACID means Atomicity, Consistency, Isolation, Durability. A transaction is a single logical operation on the data in the database. Atomicity requires that each transaction be "all or nothing": if one part of the transaction fails, then the entire transaction fails, and the database state is left unchanged. Consistency ensures that any transaction will bring the database from one valid state to another. Isolation ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially. Durability ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. ACID is very important to ensure data does not get lost or evaluated incorrectly. [12]

3.2 Structured Data

RDBMS databases deal with a very structured form of data, a table with column and rows. This is how most data that people deal with is organized. Spreadsheet with columns and rows, This row relates to this column in this way. That is how RDBMS store their data. One spreadsheet would be a table in RDBMS. There can be as many tables as one needs. An example of this would be two tables, one has a list of books as the rows, with the number of pages as the first column and the author as the second column. The author column has a number in it, lets say its 4 for the first book. That number refers to the second table which is a table of all the authors the fourth author in this table corresponds to the number 4 in our book table. It is a very simple way to look at data and works very well in plenty of cases. This structured data in RDBMS are also meant to be normalized which means "organizing the columns, and tables to reduce data redundancy and improve data integrity" [13] This normalization of the database often reduces data storage but in some cases data duplication can actually improve performance drastically while only sacrificing a little bit of storage.

4. FEATURES OF NOSQL

Using certain NoSQL databases has different upsides; the intention here is to stay general over many different NoSQL databases. Some of the upsides may not apply to all of the databases or apply to as great of a degree as others. There are new products constantly emerging in the NoSQL field as well as existing products having multiple versions and releasing new features all the time. There are two instances in the papers we discuss where MongoDB actually changes between testing and the writing of the paper which could have changed the outcomes of the benchmarks. There are at least 225 NoSQL databases in existence today so NoSQL databases really need to be viewed individually for ones purpose and data set. One example of how MongoDB stores data is in a document called a JSON object. There can be as many documents as needed within the database. Looking at the JSON object in Figure 1 we see this looks very different compared to a spreadsheet.

```
{
        "id":"JSLE23MN",
        "personal":{
                "name" : "John Smith",
                "gender" : "Male",
                "age" : "22",
                "address" : {
                        "streetaddress" : "654 8th St",
                        "city" : "Plano",
                        "state" : "TX",
                        "postalcode" : "12345",
                }
        }
}
```

Figure 1: An example of a JSON object

4.1 BASE

An alternative set of properties to ACID is BASE which means Basic Availability, Soft-state, and Eventually Consistency. Basic Availability supports partial failures without total system failure. Soft-state means the data could change over time without any input. Eventually Consistency means the consistency of the database will be fluctuating. This gives leeway in the strict set of properties of ACID. Two of the NoSQL databases we look at later actually give the option to have eventually consistency or strong consistency. [8]

4.2 Consistency

Not everything needs a database that complies entirely to ACID, which is where NoSQL databases come into play. NoSQL databases often follow the properties of BASE, by sacrificing ACID compliance they can decrease latency which is the time between request and response. For example, looking at eventually consistency.

If a change is made to a web page from a server in China that server will be updated right away. One may try to access that web page in the United States and get out of date information, until maybe thirty seconds later when the server in the United States gets updated. The consistency was not there the entire time but it eventually became consistent. Seeing the non-updated page allowed one to have access instantly instead of waiting thirty seconds for the changes to be propagated. Amazon's Dynamo NoSQL database actually "pioneered the idea of eventual consistency as a way to achieve higher availability and scalability: data fetched are not guaranteed to be up-to-date, but updates

Table 1: Insertion times of MongoDB and MySQL

Number of Entries	MySQL(ms)	MongoDB(ms)
500,000	16,064,999	17,860

are guaranteed to be propagated to all nodes eventually." [1]

Another option to deal with consistency is the Multi-Versional Consistency Control. This is similar to the previous example except both people would have access to the updated document. One person may be editing it while the other person is viewing it so pulling up the web page will give you a web page that is fully functional even though someone may be making changes to it at that very moment. Once the changes are made and propagated throughout the database and you refresh or revisit the page the updates will be there.

4.3 Horizontal Scalability

Distributed systems are an easy way to increase Horizontal Scalability. Horizontal scalability "means the ability to distribute both the data and the load of these simple operations over many servers, with no RAM or disk shared among the servers." [11] The ability to scale horizontally allows one to meet very large data and processing needs as well as keeping costs lower than increasing single machine resources (RAM, CPU, storage capacity)

One way NoSQL databases deal with scaling is the master/slave replication. In the master/slave architecture there is one master that handles all writes that get requested and send updates to the slaves. The slaves handle all read requests. This can be a problem if the master gets more requests than it can handle. The problem can be solved by what is called sharding, which is having two or more masters. The masters then split up the writes across however many their are essentially each master has its set of slaves. They have the same information on them. There is a router that sends the reads and writes to each master and slave and keeps track of what is where.

Distributed systems were lightly touched in the last section when talking about master/slave and sharding. These are techniques to allow distributed systems which is another way to increase horizontal scalability. It is the most common and cheapest option for scaling a NoSQL database and actually a huge attraction to NoSQL databases. Scaling a RDBMS almost always requires improving processor, memory, or storage which can be a lot more expensive than getting five servers that are much cheaper and spreading out the work.

4.4 Large Datasets

NoSQL databases can handle large datasets better than RDBMS. A big part of that is due to the ability to scale horizontally much better than a RDBMS. Another reason is that by allowing duplication and not focusing on normalization, performance can often be increased and duplication can either be left alone or dealt with later. Allowing for duplication that increases performance can often be very beneficial. Storage is becoming insanely cheap some data duplication may be worth it. In 2016 1GB on average costs $0.019 and in 1980 1GB on average costed $437,500 [9]

NoSQL handles large data much better than a RDBMS. Facebook Messages is a prime example of this. In 2010 Facebook posted about their underlying technology of Facebook Messages stating "MySQL proved to not handle the long tail of data well; as indexes and data sets grew large, performance suffered." [6] He also states that "current Messages infrastructure handles over 350 million users sending over 15 billion person-to-person messages per month." [6] MySQL was just not able to handle that amount of data so Facebook had to consider alternatives and ended up moving to a NoSQL database (HBase) because it offered good scalability and performance for the workload.

Table 2: Search Times of MySQL

Searched on	No of Entries	Ind. Queries	Time (ms)
Col w/o Index	500,000	4	1374.5
Col With Index	500,000	4	621.75

Table 3: Search Times of MongoDB

Searched on	No of Entries	Ind. Queries	Time (ms)
Col w/o Index	500,000	4	210.5
Col With Index	500,000	4	26.25

5. PERFORMANCE EVALUATION

5.1 MySQL vs MongoDB

Three individuals from Pune Institute of Computer Technology in India were looking at NoSQL and unstructured data as well as a comparison between MySQL and MongoDB. I will be using their comparison of MySQL and MongoDB for the next section. NoSQL databases are supposed to increase performance with big data in certain areas compared to RDBMS. Directly comparing MySQL vs MongoDB over distributed networks has some pretty interesting results, especially in terms of insertion. As we can see in Table 1 when doing an insert into the database of 500,000 records, MySQL took 1,606,499ms while MongoDB took 17,860ms, an insane difference in speed. One of the reasons we see this insane difference is actually due to lack of the durability part of ACID. MongoDB can have in-memory storage engine and "the in-memory storage engine is non-persistent and does not write data to a persistent storage." Thus "waiting for data to become durable does not apply to the in-memory storage engine." [5] This speeds up the write times of MongoDB but if power was to be lost while all that was in memory and not written to disk yet all that data that had not yet been written to disk would be lost. This lack of durability increases MongoDB's write time by freeing it up to do other things instead of waiting for the writes to happen to disk, but at the same time sacrifices durability.

Insertion is not the only place MongoDB was able to outperform MySQL. It also outperformed in terms of searches on other columns without index and other columns with index. Looking at Tables 2 and 3 which have the results on searches of the 500,000 inserted items for MySQL and MongoDB respectively. MySQL had a search time of 1,374.5ms on columns without index and 632.75ms on columns with index. MongoDB had a search time of 210.5ms on columns without index and 26.25 on columns with index. As we can

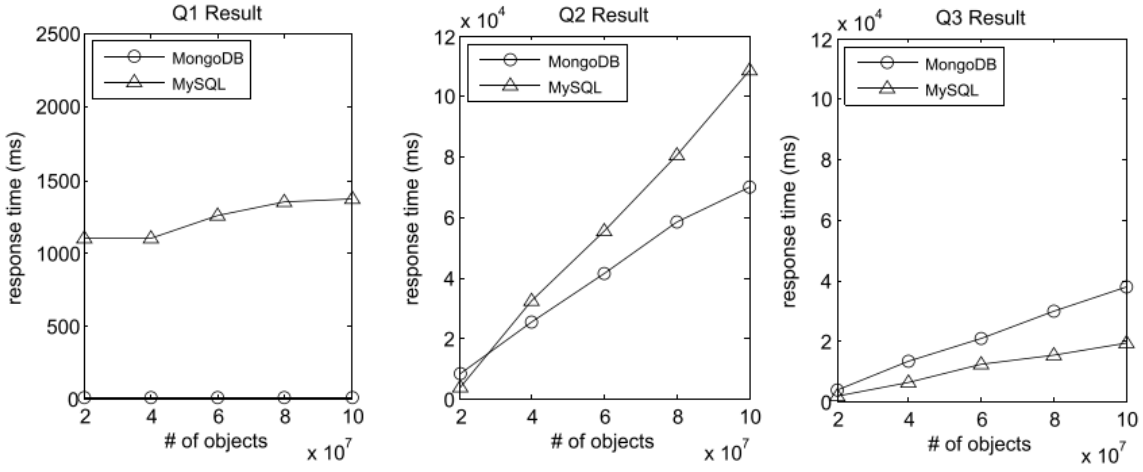

Figure 2: Three different comparisons between MongoDB and MySQL [3]

see here, MongoDB wins in both with and without index. Since these are read times there is no writing to memory instead of disk to show a performance increase so MongoDB is just faster in this case. [7]

Some more results comparing MySQL and MongoDB by four individuals from Dongguk University shows an aspect where MySQL is better than MongoDB. They were looking at having a MongoDB based repository for some big data they wanted to explore. One thing to note, it appears these results were tested on single sharded MongoDB and a single machine for MySQL.

In Figure 2 we can see three different queries done of RFID data (Q1, Q2, and Q3) within MongoDB and MySQL. Looking at Q1 we see a case where MongoDB clearly is the better choice, it is getting almost instance response time. Q2 MySQL is better with two objects but as the number of objects increases we see it's performance decrease at a rate faster than MongoDB. Finally in Q3 we get a case where MySQL actually out performs MongoDB. Do get a better idea of what was actually being queried look at [3].

5.2 Comparing NoSQL Databases

NoSQL databases have similarities amongst each other but they are not the same and a person needs to consider if their data is a good fit for a certain NoSQL database. How the data is used is another big factor when deciding which NoSQL database to choose. Some professors at Carnegie Mellon conducted a case study for a distributed healthcare organization that decided they would like to try a NoSQL database for their Electronic Health Record System. The goal of this case study was to look at different NoSQL databases and decide which would work best for this healthcare organization's needs. They did not look at SQL databases because the customer was familiar with RDBMS technology for this case already, but was looking to focus the technology evaluation only on NoSQL.

They first wanted to understand what kind of performance and scalability gains were made with each option that they were considering. They chose two driving cases, retrieval of recent medical tests for a certain patient, and strong consistency when writes take place. This case actually wants strong consistency instead of eventual consistency as mentioned before. Because they needed strong consistency they use *quorum consistency* to ensure consistency across shards; see [4] for more details. A decision was make to look at several different data models (key-value, column, and document). The choice to look at Riak (key-value), Cassandra (column), and MongoDB (document) was made due to their features and the fact that they are market leaders.

All tests were run using Amazon EC2 cloud. The data was input as follows: there are one million patient records, and each patient has between 0 and 20 lab test results with ten million lab test results in total. [4] Tests were run on two different configurations of the database: a single node and a nine node setup. The single node setup was a base case while the nine node setup was to represent an actual deployment. The "data was partitioned (i.e. "sharded") across three nodes, and replicated to two additional groups of three nodes each." [4] All of the data was split into three sections and that was replicated twice so it was a sort of 3x3 data distribution. This was the case for Cassandra and MongoDB but Riak was unable to support this configuration so on Riak the "data was sharded across all nine nodes, with three replicas of each shard stored across the nine nodes." [4]

5.3 NoSQL Test Results

Testing was performed under a workload of 80% read and 20% write due to the health care organization reporting this as their usual workload. A local cache was created that represented the patients that were to be seen that day. The workload execution was also executed on a different number of client threads (1, 2, 5, 10, 25, 50, 100, 200, 500, and 1000), three times for each amount of threads. The average of these three tests was used for evaluation. Results shown are from the three different databases running on each amount of threads and their respective throughput, or operations executed per second.

Looking at Figure 5, which is throughput of the databases with read/write workload, we see that Cassandra is clearly the best option here. Cassandra peaked around 3200 operations per second which is slightly better than the single node configuration that was also tested. Riak peaked around 480 operations per second but did see an increase for the distributed system of 4x compared to the single node configu-

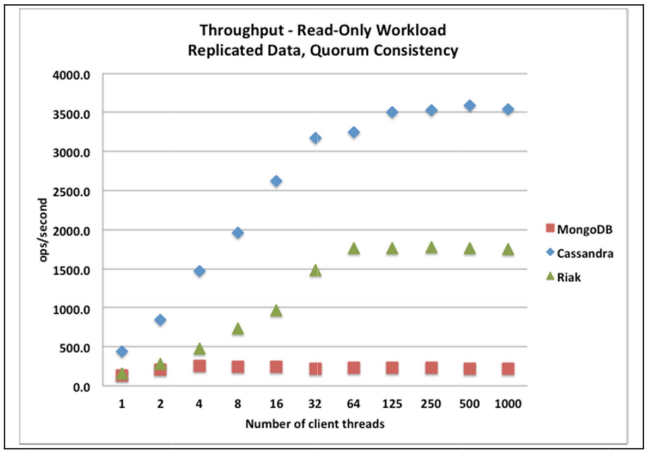

Figure 3: Throughput for Riak, Cassandra, and MongoDB on a read only workload [4]

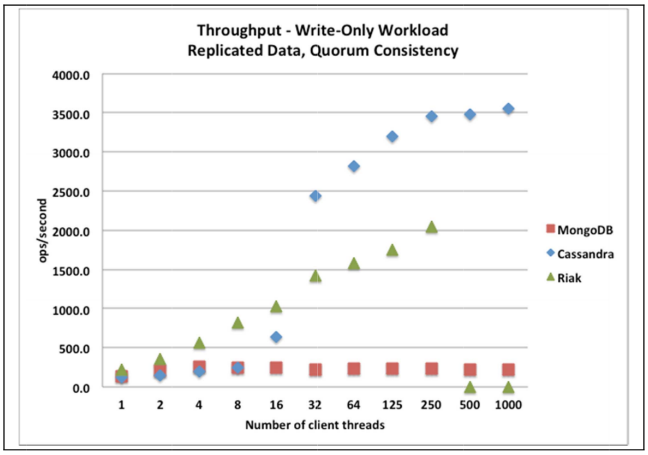

Figure 4: Throughput for Riak, Cassandra, and MongoDB on a write only workload [4]

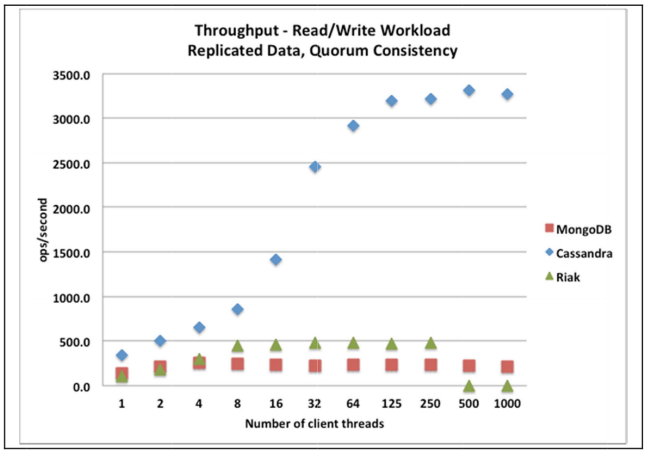

Figure 5: Throughput for Riak, Cassandra, and MongoDB on a read/write workload [4]

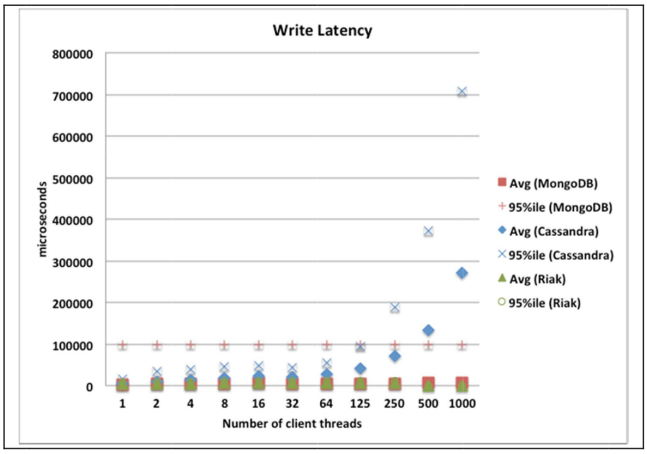

Figure 6: Write latency [4]

ration. Notice Riak drops to zero at 500 and 1000; that is due to "insufficient socket resources to execute the workload for 500 and 1000 concurrent sessions." [4] Looking at MongoDB we see that its performance is quite poor. It is actually achieving less than 10% of what the single node configuration of MongoDB is able to. A factor affecting this was "the interaction between the sharding scheme used by MongoDB and our workloads." [4] The way the workloads generated keys caused MongoDB to direct all write operations to the same shard which resulted in performance decreases. Conflicts between other systems and MongoDB made it so a different indexing scheme was unavailable. One thing to note is that after these tests concluded, MongoDB introduced a hash-based sharding which may be a viable option for this data but tests were not conducted for this. This is a great example of the constant changing field that is NoSQL.

Looking at the write latency of each in Figure 6 we see that Cassandra clearly has the highest latency, MongoDB had the next highest latency, and Riak offered the lowest for latency. MongoDB was 4x faster and Riak was 5x faster than Cassandra. The read latency in Figure 7 shows that MongoDB is the best in this case followed by Cassandra but Riak seems to lose performance as we add more client threads in. This is not good considering we have a 20% write workload and an 80% read workload.

After MongoDB has been excluded because it is ruled out as being a viable option for the tasks at hand, The second set of tests are similar to the the first set of benchmarks discussed. They are looking at the differences strong consistency versus eventually consistency has on throughput. Looking at the 32 client threads in Figure 8 throughput yields a 10% decrease when moving to strong consistency from eventual consistency with Riak. We see an even larger decrease when considering Cassandra in Figure 9. A decrease of 25% is observed when switching from eventually consistency to strong consistency with 32 client thread.

It was not clearly explicit which choice they went with in the end. They had two viable options with different drawbacks. Cassandra has a very high throughput. While Riak offers very low write latency the higher read latency is not ideal. Arguments can be made for both but it really depends on what the system is focusing on. The choice between high throughput and low latency is to be made by the customer.

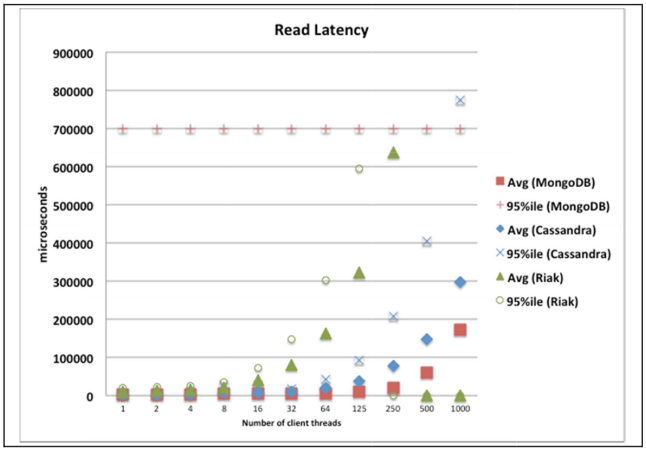

Figure 7: Read latency [4]

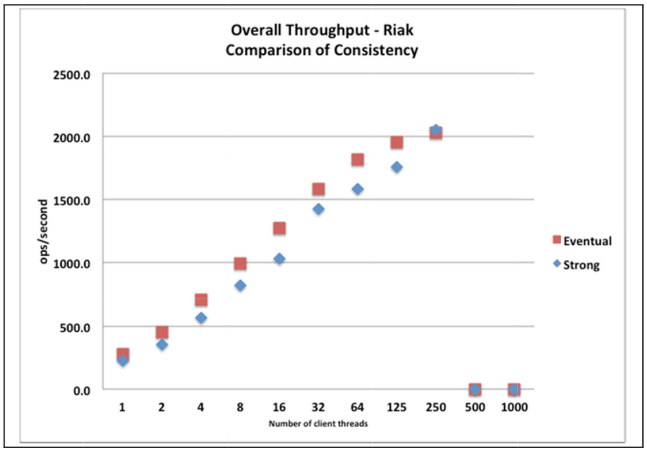

Figure 8: Consistency comparison in Riak [4]

6. CONCLUSION

Overall testing the data model and application is the most important when deciding which database will fit the best for each situation. As we saw MongoDB was clearly better in the first scenario but did not do nearly as well in the second. We can see that each database type, both RDBMS and NoSQL have their own place in the world. Currently RDBMS have control over most of the applications but we may see NoSQL taking up more of that space as we see this constant increase in data. As I mentioned each has their own place and I do not see one completely eliminating the other from existence. Facebook currently has teams working on both types of databases, Facebook Messages is build on HBase and Facebook itself is build on MySQL so it is evident they see each have their uses. I also do not foresee the be all and end all emerging in either field. There will not be an automatic go to when the word NoSQL is mentioned and that will not be the case for RDMBS either. Running benchmarks will allow one to choose the database that best fits the needs at hand.

7. REFERENCES

[1] R. Cattell. Scalable SQL and NoSQL Data Stores. *SIGMOD Rec.*, 39(4):12–27, May 2011.

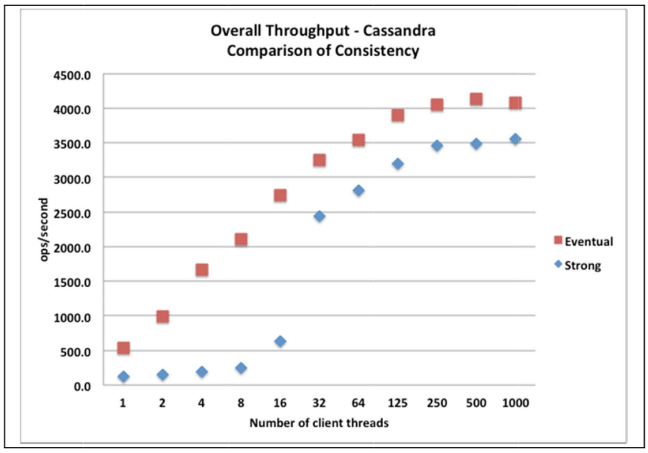

Figure 9: Consistency comparison in Cassandra [4]

[2] IDC. The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things, 2014.

[3] Y. S. Kang, I. H. Park, J. Rhee, and Y. H. Lee. MongoDB-Based Repository Design for IoT-Generated RFID/Sensor Big Data. *IEEE Sensors Journal*, 16(2):485–497, Jan 2016.

[4] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser. Performance Evaluation of NoSQL Databases: A Case Study. In *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*, PABS '15, pages 5–10, New York, NY, USA, 2015. ACM.

[5] Mongo. In Memory Storage Engine, 2016. https://docs.mongodb.com/v3.2/core/inmemory/.

[6] K. Muthukkaruppan. The Underlying Technology of Messages, 2010. https://www.facebook.com/notes/facebook-engineering/the-underlying-technology-of-messages/454991608919/ [Online; accessed 29-November-2016].

[7] S. S. Nyati, S. Pawar, and R. Ingle. Performance evaluation of unstructured NoSQL data over distributed framework. In *Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on*, pages 1623–1627, Aug 2013.

[8] D. Pritchett. BASE: An Acid Alternative. *Queue*, 6(3):48–55, May 2008.

[9] Statistic Brain. Average Cost of Hard Drive Storage, 2016. http://www.statisticbrain.com/average-cost-of-hard-drive-storage/.

[10] Tutorialspoint. SQL - RDBMS Concepts, 2016.

[11] M. N. Vora. Hadoop-HBase for large-scale data. In *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, volume 1, pages 601–605, Dec 2011.

[12] Wikipedia. Acid, 2016. [Online; accessed 15-November-2016].

[13] Wikipedia. Database normalization — Wikipedia, The Free Encyclopedia, 2016. [Online; accessed 11-November-2016].