Comparison Between "Big Data – NoSQL" Data Models

Aldric Lopez
Software engineering
École Polytechnique de Montréal
Montreal, Canada
aldric.lopez@polymtl.ca

Andrew Duy-Anh Pham
Software engineering
École Polytechnique de Montréal
Montreal, Canada
andrew-duy-anh.pham@polymtl.ca

Thierry Fortin
Software engineering
École Polytechnique de Montréal
Montreal, Canada
thierry.fortin@polymtl.ca

Abstract*—NoSQL databases are becoming the norm when it comes to storage in most modern companies. Traditional relational databases offer a strict data model which allows for data integrity and efficient and straightforward transactions. However, when it comes to Big Data and the colossal amounts of information being stored and analyzed, they simply cannot compete with their modern NoSQL counterparts. The flexibility and scalability of NoSQL databases outperform traditional relational databases when it comes to managing large volumes of data. The formless data models used by NoSQL databases are more appropriate for the volatile nature of Big Data. Depending on one's needs, different types of NoSQL databases can be deployed. The goal of this paper is to determine which type of NoSQL database would be overall more performant. Firstly, the traditional relational databases and how NoSQL databases differ from them will be explored. Second, the different types of NoSQL databases will be presented as well as specific examples of each type of NoSQL database. Third, using the Yahoo! Cloud Service Benchmark tool, different tests will be executed on instances of each database in order to recover valuable metrics on each of them. Finally, a comparison of the results of these tests will be done in order to determine which type of database is more performant.

Keywords: Big Data, NoSQL Database, Key-Value, Document-Based, Wide-Column, Redis, Mongo, Azure Cosmos, YCSB

I. Introduction

With its numerous applications in different fields, Big Data is becoming more and more prevalent as the years go by. The popularization of web services and the development of cheaper data storage has resulted in the production of enormous volumes of data. Additionally, the developments on the cloud service front, has greatly accelerated the rise of Big Data as outsourcing computational efforts is now a possibility. Bigger corporations can now rely on technology giants like Google, Amazon, and Microsoft to host their data and process it without having to worry about managing the infrastructure itself. On a daily basis, it is estimated that 2.5 quintillion bytes of data are being produced across the world [1]. Analyzing all this data can help us identify patterns, it can give us insight on subjects that were once obscure and give us answers to questions that were previously unanswerable. Depending on the field of application, all this information has opened up doors to a multitude of new possibilities. In the service industry, companies can optimize their services to each client by analyzing different behavior patterns and habits shown by its users. Using all the information gathered with measuring tools and satellite imagery, scientific organizations and researchers can accurately develop models to predict the effects of climate change. In the financial sector, Investment firms can analyze trends to predict the future behavior of the market and confidently plan their future investments using these prediction models. With all these different applications of data, a strong infrastructure is needed to support it. To manage all this information, these different entities need modern ways of storing and processing the data that traditional databases cannot offer. Traditional relational databases require specific and predefined structure which is not suitable for the volatile nature of Big Data. To make up for these shortcomings, a new type of database was developed, NoSQL databases. These types of databases, which started emerging at the turn of the century, offer the possibility of having a schema-less data model and much better horizontal scaling. Modern companies can benefit from this as they can theoretically expand their operations endlessly and seamlessly without much effort other than paying for the services. Because of its flexibility, NoSQL databases have grown in tandem with Big Data and are now highly associated with one another. Several types of NoSQL databases exist, each having their own advantages based on the type of data being stored. In this study, we will be presenting and comparing the different types of NoSQL database. To compare them, an instance of each type of NoSQL database will be deployed and tested on their performance using the Yahoo! Cloud Service Benchmark tool (YCSB). Finally, using the results, arguments will be provided for each database as to why they should or should not be adopted.

II. NoSQL Databases

First and foremost, to understand NoSQL databases, a strong understanding of what relational databases are is needed. Traditional relational databases were the norm for many years. They required a precise data model which each entry would have to follow. This allowed for simple and clean structure for the databases. Where relational databases fell short, NoSQL

databases were able to NoSQL means non-SQL, with SQL being an acronym for Structured-Query-Language, the most popular query language for relational databases. NoSQL database thus refers to all non-relational databases. When it comes to NoSQL databases themselves, as mentioned, they offer a much more flexible use which is more appropriate for Big Data. These databases do not operate using a static data model, which allows all types of data to be ingested and analyzed. The way these databases are conceived allows for better expandability, or horizontal scaling. When used with cloud services, these databases allow for a great fault tolerance. Generally, these databases will be hosted on distributed systems, meaning that the entire database is broken into pieces and hosted on several machines. When put together, this cluster of machines, or nodes, makes up the entire database. This segmentation of data allows for different practices which allow more control over the availability of the information. Data can be replicated across different nodes to ensure that information is always accessible even if one node goes down. It is also possible to define where specific data is to be hosted geographically in order to allow commonly accessed data to be hosted closer to their main clientele, which lowers response times. This amount of control and precision is ideal for companies as it allows them to easily expand their operations based on the amount of data they manage. Based on the format of the data, different types of NoSQL databases can be used. The four types of NoSQL databases are key-value databases, document-based databases, wide-column databases, and graph databases. All four of these categories of databases have their benefits and drawbacks and offer a multitude of uses depending on one's needs. For this study, only the key-value, document-based and wide-column databases will be presented and analyzed. For each type of NoSQL database, a specific implementation of these databases. For the key-value databases, Redis will be used. MongoDB is the chosen database to represent document-based databases. And finally, Azure Cosmos DB will be used for the wide-column databases.

A. Key-Value

Key-value databases are now used more than ever. Key-value databases or key-values store is a NoSQL database which manages and stores key-values pairs. The key-value model uses a hash table where the key can be synthetic or auto generated and must be unique and where the values consist of large data fields. The value field follows the concept of objects in object-oriented-programming, however, contrary to these objects, the value field may have values with different attributes or even with missing attributes. Additionally, there is only one way to retrieve data from these databases and it is throughout its key. This also means that grouping key-values pairs can only be done via collections. Generally key-value databases store data in memory which makes them fast and enables high throughput data access [2]. For these reasons it has become a very attractive solution in several scenarios such as graph processing in social media, online transaction processing, machine learning pipelines and more. Furthermore, since these databases don't use a schema and do not operate using a static data model it enables a greater flexibility and scalability for storing data making them suitable for big data.

1) Redis DB

Nowadays, one of the most popular key-value databases is Redis. As with most key-value databases, its great performance is mainly because all data resides within the memory allowing low latency and high-speed data access. One of the features that make Redis so popular is its flexible data structure. "Redis modern data structure avoids the overhead associated with translation between application objects to database entities for every database operation" [3]. The data types supported by Redis are strings, hashes, lists, sets, sorted sets and more. As a result, the flexibility offered by Redis allows data to be stored more efficiently and allows the use of this database in various types of situations. Moreover, large organizations must ensure that data is always available for their users even in critical situations. Redis has built-in replication and uses asynchronous replication to guarantee high availability and reliability when this type of situation occurs. Asynchronous replication is the default replication mode, and it is the most suitable for most use cases. However, it is possible to use synchronous replication for certain data, but Redis cannot assure strong consistency. Another Redis feature is automatic partitioning with Redis Cluster. Redis Cluster is a deployment topology that allows horizontal scaling. Hence, it allows data to be shared across the multiple Redis nodes. Within a Redis cluster, a server can be referred to as a node.

B. Document-based

Document-based databases are designed to store and treat data as documents where a document is a record in a document database. Within this type of database, data is stored as a collection of key-value pairs [2]. Naturally, all keys must be unique, and every document must have a unique id within the document collection to ensure explicit identification of every document. A collection consists of a group of documents that have similar content. However, it is not mandatory that the content in a collection has the same schema for the reason that document databases do not have a schema restriction. This feature makes document-based databases very flexible and adaptable to multiple situations. Moreover, document-based databases can easily handle unstructured data and enable the ability to easily modify a document in a collection without affecting other documents [4]. As a result, this database is great to handle large volumes of data like in content management and catalogs situations where data evolves over time.

1) MongoDB

MongoDB is a document-based database and it stores data as BSON documents. MongoDB documents are made of a field value pair and as most document-based databases it is flexible

and adaptable since it doesn't have a schema restriction. In addition, MongoDB groups documents in collections. Some of the benefits of MongoDB are that it offers horizontal scalability, replication, and high availability. One of MongoDB core functionalities is horizontal scalability using sharding. Sharding is a method used to distribute data over a cluster which is composed of shards. Each shard will have a portion of sharded data. As a result, as the amount of data grows, MongoDB will distribute the read and write workload across the shards and by adding shards it will increase the storage capacity of the cluster. Additionally, to provide redundancy and high availability, MongoDB facilitates replication with replica sets. A replica set consists of two or more MongoDB instances that preserve the same data set. Each replica set unit can act as a primary or secondary replica at any time. The primary replica is responsible for recording all changes to its data sets and the secondary replica will maintain a copy of the primary. When the primary replica fails, the replica set automatically switches over to the secondary keeping the system up and running [5].

C. Wide-Column

Wide-column databases, also known as Column family stores, are a way to store data in a distributed multidimensional sorted map [2]. The data stored in this model is mostly key-value pairs stored within rows, this makes this data model very flexible, as the columns in it can be of any size or have as many attributes as it needs. They are also conceptually the closest model to a relational database. Furthermore, it supports versioning, meaning the values are sorted in a chronological order [2]. This type of data model is extremely efficient and encouraged for databases with huge amounts of data, because of its high performance on aggregation queries (MIN, MAX, AVG, etc.). It can also be noted that this model grandly facilitates the migration of data in relational databases to distributed ones. Moreover, these systems are made to work with MapReduce and Hadoop, distributed systems for data analyses. We can see this model used for HBase, Accumulo and even Cassandra, a hugely popular DB even used by Netflix to this day. Finally, this type of storage is used in priority when scalability and availability is a priority.

1) Azure Cosmos DB

Azure Cosmos DB is a globally distributed, multi-model database service that is designed to enable users to elastically and independently scale throughput and storage across any number of geographical regions. Speaking of, a multi-model database service is a type of database that allows users to access data using multiple data models, such as key-value, document, wide-column and graph. Cosmos DB allows users to access data using a variety of different APIs, including Cassandra, MongoDB, SQL, and Azure Table Storage. It offers automatic and instant scalability of throughput and storage, as well as comprehensive service level agreements (SLAs) for availability, latency, and throughput [6]. This service provides built-in global distribution, with the ability to replicate data to multiple regions with a single click, ensuring low-latency access to data from anywhere in the world. Also worthy of note, as opposed to the two other databases presented in this report, Azure Cosmos is not a local database, meaning the data is not stored locally, it is stored in the cloud. A token is then necessary to access the database correctly and safely. To use Azure Cosmos DB, and incidentally to run the YCSB benchmark, you have to create a database account in the Azure portal, and then use one of the supported APIs to create and manage your data. In this case, only the Command Line was used to Insert, Update and Read data in the database. An important detail to take in consideration, Azure Cosmos only lets its users and admins modify the numbers of clusters for a PostGreSQL implementation, so it wasn't possible to modify that number in the present benchmarks.

III. COMPARING THE MODELS

Let us now compare the three NoSQL solutions presented in this paper by discussing their advantages and disadvantages.

Key-value databases are very simple to use and can be quickly accessed using the right key. This makes them useful for storing large amounts of data that needs to be accessed quickly, such as in caching systems or real-time data processing applications. Additionally, key-value stores are often highly scalable, and can be easily distributed across multiple servers to support large amounts of data and high levels of concurrency. One disadvantage of key-value stores is that they do not support complex data structures or relationships between data. This means that they are not well-suited to applications that require complex data modeling or querying capabilities. Additionally, key-value databases do not support transactions, which can make them less reliable for applications that require data integrity.

Document-based solutions, on the other hand, have the advantage of supporting complex data structures and relationships between data. This makes them useful for applications that require sophisticated data modeling and querying capabilities, such as content management systems or e-commerce platforms. Document databases also support transactions, which can improve the reliability and data integrity of the system. One disadvantage of document stores is that they can be less efficient for storing and retrieving simple data. Because document stores store data in structured documents, accessing a single piece of data can require reading the entire document, which can be slower than using a key-value store. Additionally, document stores may not be as scalable as key-value stores, depending on the implementation.

Wide-column databases have the advantage of being highly scalable and efficient for storing large amounts of data. Because they store data in columns rather than rows, they can support very large data sets and high levels of concurrency.

Additionally, wide-column databases support transactions, which can improve the reliability and data integrity of the system. One disadvantage of wide-column databases is that they can be more difficult to use and require more sophisticated data modeling than other NoSQL solutions. Because data is stored in columns rather than rows, the schema for a column family database must be carefully designed to support the desired data access patterns. Additionally, wide-column databases may not be as flexible as document stores when it comes to supporting complex data structures and relationships.

All of these NoSQL solutions have their own use-cases, and there's not a better one for all situations. Companies select their desired model in accordance to their needs, financial situation and above all, the type of application they have. For example, key-value stores are mainly used for simple projects that require rapid response times but use basic transactions. On the other hand, document databases are more popular for convoluted projects that need to do lots of intricate exchanges to work correctly. Wide-column databases are better suited for the traditional Big Data operations which require large quantities of data and have complex analysis.

IV. Benchmarking Methodology

In order to test Redis, MongoDB and Azure Cosmos DB against one another, a standardized testing process is needed. Using the YCSB tool, the loading capacities and running capacities of each instance will be tested. The YCSB benchmarking tool is commonly used within the industry to measure the retrieval and maintenance capabilities of NoSQL databases systems. It comes with predefined workloads which can be easily used with our deployments and also offers a template to define our own custom workloads that will work with their testing suite. The workloads offer a multitude of properties that can be adjusted to create ideal tests to evaluate the target database. The main attributes used are the number of operations to be executed and the proportion properties which dictate how the operations should be split between different types of actions.

In this study, for each instance, three different workloads will be used with each database instance in order to test their capabilities. The chosen workloads are workloads A, B and D from the predefined workload sets offered by the YCSB tool. The data for each workload must first be loaded in the databases in order to run the various tests. Each workload is made up of a thousand operations varying between read, insert and update tasks. Workload A has its operations evenly split between reading and updating tasks. Workload B has a read proportion of 95% and an update proportion of 5%. Lastly, workload D has a reading proportion of 95% and an insert proportion of 5%. Each of these workloads will be run three different times on each database instance. With the results of each test, the average results will be calculated and used to compare each of the database instances between each other.

In order to have an even testing field, the Redis and MongoDB instances were deployed in a docker container with the same configuration which consisted of a single cluster, each holding six nodes These clusters were hosted on a virtual server provided by AWS. The server was running on 64-bit (x86) Ubuntu Server 22.04. The instance type was t2.medium with 1 vCPU, 4Gib of Memory and a 20gb of storage. Currently, Redis Cluster and Docker encounter some issues due to IP addresses or TCP ports remapping. However, some alternatives exist to solve this problem such as using Docker's host networking mode. Also, in a non-production environment, there are compatible docker containers that can be used with Redis Cluster. Therefore, one of these docker containers was used to run the YCSB. The configuration of this container consisted of 6 running Redis instances, 3 master and 3 slaves, one slave for each master. With MongoDB, there are many methods to create a cluster. In a development environment, one of the easiest and quickest ways to do so is by creating a cluster in the cloud. This feature is provided by MongoDB Atlas which is a database-as-service [13]. Another alternative is using Docker, this alternative is useful since it ensures that all members of a team run the same cluster. Consequently, this method was used to run the YCSB. The configuration used consisted of 6 instances of MongoDB in a replica set to match Redis Cluster environment.

However, it is not possible to deploy the Azure Cosmos database with this configuration as they do not offer such control over the instance. Using the free trial, we could not have as much control as with the Amazon VMs we used for the other database. The one setting that was available for modification that had value was the throughput control. In our case, we set the throughput configuration to autoscale, meaning that the instance would manage its throughput itself based on the amount of work. Additionally, from our understanding of the Azure Cosmos documentation, with the trial version, only when paired with PostgreSQL does Azure Cosmos offer cluster-level configuration. This means that we were unable to deploy an instance with the same cluster configuration as the other two instances. This creates an additional source of possible errors.

Furthermore, while local database instances were deployed on Amazon virtual machines for Redis and MongoDB, the Azure Cosmos database was hosted externally and required a connection using an access token. This added a significant amount of latency, which we believe was taken into account by YCSB when calculating the latency metrics. Therefore, when comparing the results of Azure Cosmos' results with the other instances, the analysis of the results must be done keeping in mind that they did not have the same testing baseline.

V. COMPARING RESULTS

A. Workload A Results

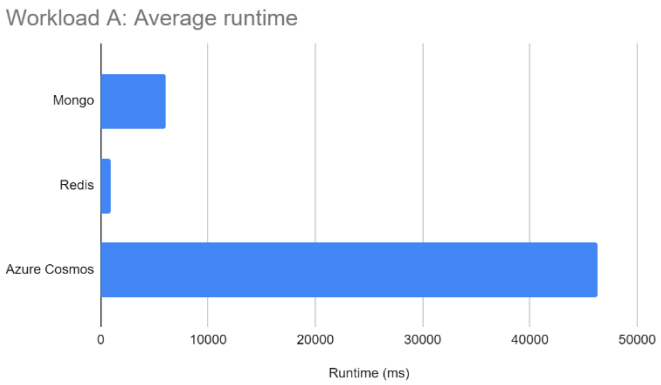

Figure 1. Average runtime of workload A

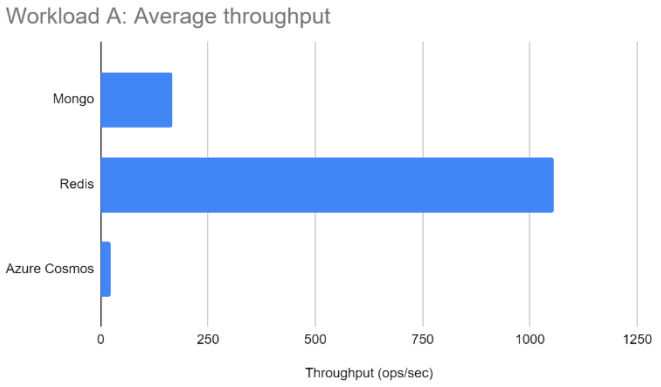

Figure 2. Average throughput of workload A

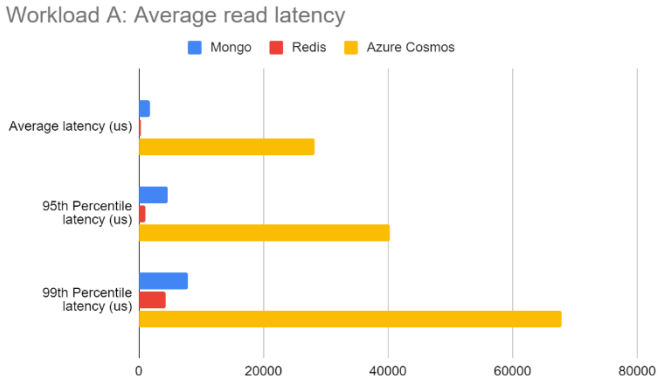

Figure 3. Average reading latency of workload A

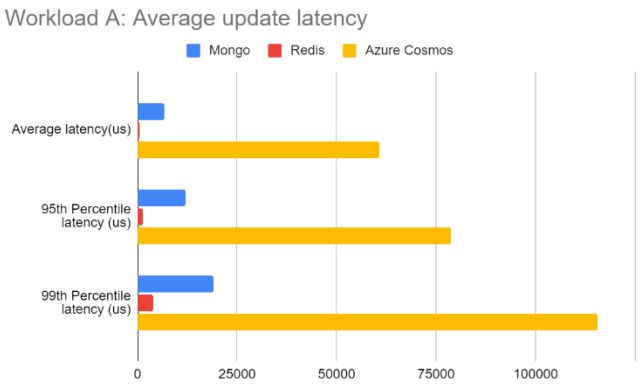

Figure 4. Average updating latency of workload A

By observing figure 1, 2, 3 and 4, we can immediately notice that Azure Cosmos lags behind Redis and MongoDB in terms of efficiency. Redis has the fastest runtime, the highest throughput and lowest latencies for both reading and updating. MongoDB stands in second place being behind in all metrics by a significant margin. As workload A is evenly split between both reading and writing operations, we cannot distinguish whether each database is particularly better at one of these two actions.

B. Workload B Results

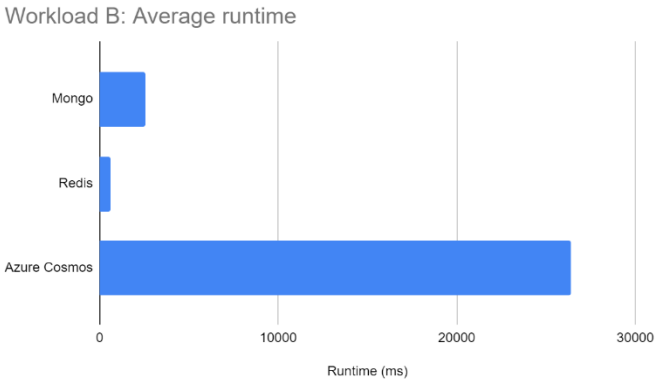

Figure 5. Average runtime of workload B

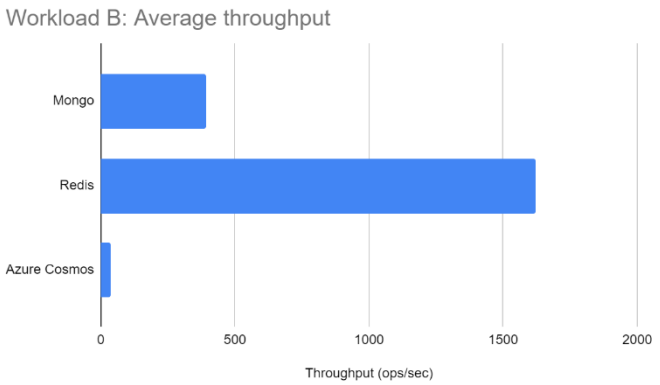

Figure 6. Average throughput of workload B

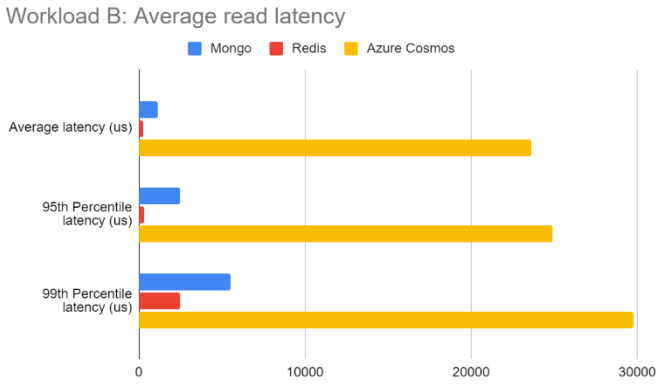

Figure 7. Average reading latency of workload B

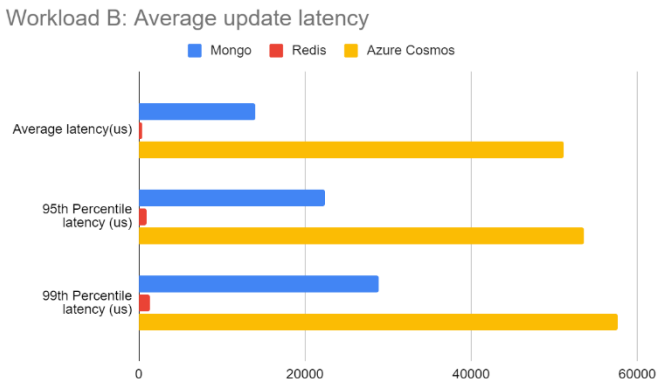

Figure 8. Average updating latency of workload B

Once again, Redis leads the group by a notable difference, MongoDB safely sits in second place while Azure Cosmos trails behind. One key difference that can be observed when comparing with the results of workload A is the overall increase in performance. Each instance has a lower runtime and reading latencies, while showing higher throughputs. Redis and Azure Cosmos also show a slight drop in updating latency, however MongoDB almost doubles its updating latency.

C. Workload D Results

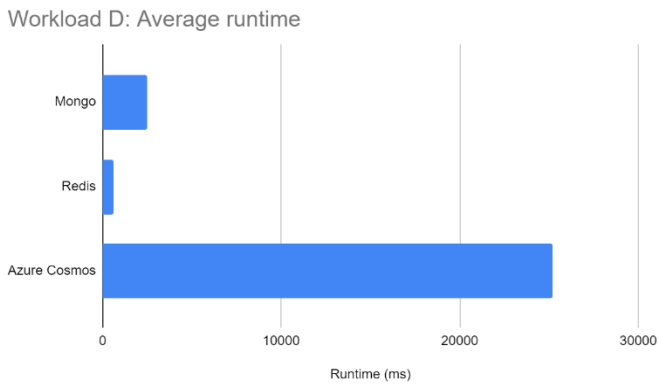

Figure 9. Average runtime of workload D

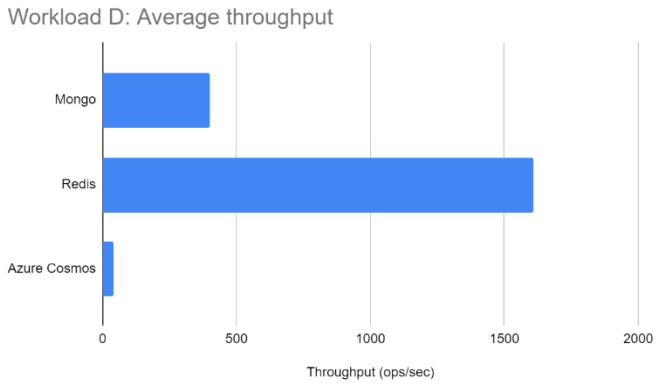

Figure 10. Average throughput of workload D

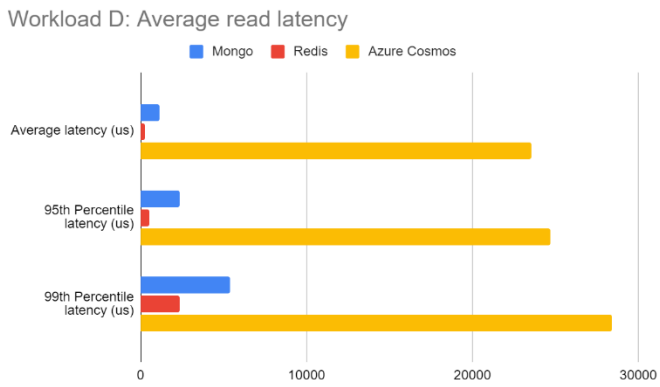

Figure 11. Average reading latency of workload D

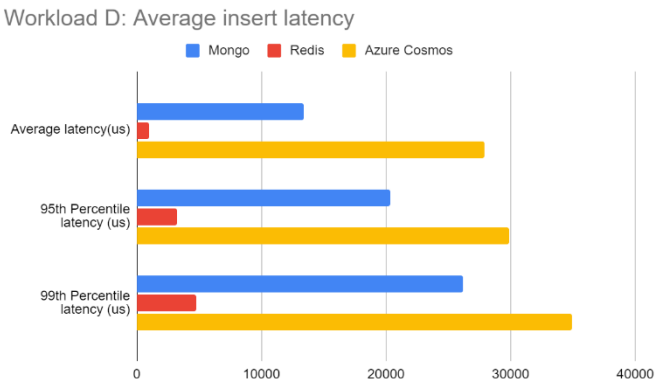

Figure 12. Average insert latency of workload D

Looking at the last set of figures, it is possible to observe that once again Redis outperforms its rivals as it has the fastest runtime, the best average throughput, the fastest average read latency and the fastest average update latency. MongoDB stayed in second place and Azure Cosmos obtained the worst results. One particularly interesting note is that in the insert latencies, Azure Cosmos seems to close the gap with MongoDB.

D. General Comparison

From looking at the different graphs, Azure Cosmos is clearly an outlier for all metrics except the cleanup latency. The low throughputs and high latency values would indicate that Azure Cosmos is much slower than its counterparts at reading and writing operations. As mentioned before, the externally hosted Azure Cosmos instance needed to securely establish a connection using a token, explaining the extreme differences with Redis and Mongo. This connection could have impacted heavily on Azure Cosmos' results as the YCSB metrics start measuring the total runtime as soon as the tests are launched. This would result in very high latency values and low throughputs as the connection time would be counted within those metrics, even though they are separate from the actual operations. Additionally, Azure Cosmos was not hosted on a machine with different configurations as the VMs used for Redis and MongoDB. This would clearly impact the performance of each machine and produce results that vary greatly based on their available resources. Therefore, it is impossible to confidently conclude that Azure Cosmos is truly slower than the other instances.

As for the other two, it is possible to confidently compare the two as they both had the same configuration and were both hosted in the same under same conditions. Redis outperforms MongoDB in every category by a significant margin. One would assume that Redis is outperforming MongoDB due to its key-value nature, however this would be false. Although Redis is explicitly a key-value database, MongoDB also uses a key-value system. Each document in the database is associated with a key which is used to search for it. This means that both instances are equally able to find elements of data using its key. However, the reason why Redis is so much faster despite the similar key-value approach, is that Redis hosts all of its data in memory and not on disk. Operating on the memory rather than the disk is much faster, which results in much lower latency times for all operations. MongoDB is primarily disk-based, although it can be configured to be memory focused, which explains the slower response times.

Overall it is clear to see that Redis outperforms MongoDB in all fields. We can conclude that Redis is definitely faster than Mongo, however we cannot confirm the same conclusion for all key-value and document-based databases. As for Azure Cosmos, our incompatible instance environment makes it impossible to confidently compare its results with the other two instances. Therefore, when it comes to the wide-column performance, our study is inconclusive.

VI. Conclusion

As relational databases cannot keep NoSQL offers great flexibility and scalability that is best taken advantage of in the context of Big Data. There are many NoSQL database options to be chosen from based on one's needs. Each type of NoSQL database is designed to answer specific use cases. Key-value databases are better for simple applications that use simple reading and writing operations. Document-based databases are more appropriate for projects that require complex transactions as their data models are more intricate. As for wide-column databases, they are better used when migrating from a relational database to a NoSQL database and when large volumes of data need to be analyzed and scaled up. Overall, from our own benchmarking results, we can conclude that Redis is faster than MongoDB due to the fact that all of the data held in the database is held within the memory and not the disks. However, our research on the efficiency of wide-column databases is inconclusive as we were not able to test our Azure Cosmos instance under reliable conditions. We hope that the information presented in this paper helps future users of NoSQL solutions when choosing the type of database, they will use.

References

[1] JJ. Ahmed and M. Ahmed, "A Study of Big Data and Classification of NoSQL Databases", 2020 IEEE International Conference on Technology, Engineering, Management for Societal impact using Marketing, Entrepreneurship and Talent (TEMSMET), 2020, pp. 1-8. [Online]. Available: https://ieeexplore.ieee.org/document/9557566

[2] A. Makris et al., "A classification of NoSQL data stores based on key design characteristics", Procedia Computer Science, vol. 97, pp. 94-103, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050916321007

[3] Redis Data Structure. [Online]. Available: https://redis.com/redis-enterprise/data-structures/

[4] What is a Document Database?. [Online]. Available: https://www.mongodb.com/document-databases

[5] What is MongoDB?. [Online]. Available: https://www.mongodb.com/docs/manual/

[6] D. Shukla. (2017) A technical overview of Azure Cosmos DB. [Online]. Available: https://azure.microsoft.com/en-us/blog/a-technical-overview-of-azure-cosmos-db/

[7] S. Kalid et al., "Big-data NoSQL databases: A comparison and analysis of "Big-Table", "DynamoDB", and "Cassandra"", 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), 2017, pp. 89-93. [Online]. Available: https://ieeexplore.ieee.org/document/8078782

[8] A. Oussous et al., "NoSQL databases for big data", International Journal of Big Data Intelligence, 4. 171. 10.1504/IJBDI.2017.085537, 2017. [Online]. Available: https://www.researchgate.net/publication/318796844_NoSQL_databases_for_big_data

[9] J. Bhogal and I. Choksi, "Handling Big Data Using NoSQL", 2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, 2015, pp. 393-398. [Online]. Available: https://ieeexplore.ieee.org/document/7096207

[10] B. F. Cooper. "Benchmarking cloud serving systems with YCSB", In Proceedings of the 1st ACM symposium on Cloud computing (SoCC '10), 2010, pp. 143–154. [Online]. Available: https://dl.acm.org/doi/10.1145/1807128.1807152

[11] L. Schaefer. What is NoSQL?. [Online]. Available: https://www.mongodb.com/nosql-explained

[12] What is a Relational Database (RDBMS)?. [Online]. Available: https://www.oracle.com/ca-en/database/what-is-a-relational-database/

[13] Introduction to Redis. [Online]. Available: https://redis.io/docs/about/