Comparison Between "Big Data – NoSQL" Data Models

Petr Smejkal
Department of Software Engineering
Polytechnique Montréal
Montréal, Canada
petr.smejkal@polymtl.ca

Lucia Cahojova
Department of Software Engineering
Polytechnique Montréal
Montréal, Canada
lucia.cahojova@polymtl.ca

Mikulas Vesely
Department of Software Engineering
Polytechnique Montréal
Montréal, Canada
mikulas.vesely@polymtl.ca

***Abstract*—Understanding the various types of data, their structure and how they can be used to extract insights is crucial in every application being developed. Our main goal is to provide a comprehensive comparison between 3 NoSQL data models and their respective implementations. To do so, we firstly provide a theoretical overview of the related topic and benchmark three chosen NoSQL implementations. Our results show todo. We conclude that when choosig an NoSQL implementation, it's important to be aware of both disadvantages and advantages of the specification, to be able to choose wisely. todo**

***Index Terms*—Big Data, NoSQL, MongoDB, Redis, Apache Cassandra, Benchmarking, Database**

I. Introduction

With the fast-changing approaches to store and access data of large volumes, different types and scales many different methods are being developed and it sometimes gets difficult to keep track of all of them.

Our work focuses on software architectures for Big Data and on recognising their suitability for different use cases. We discuss NoSQL databases, which are widely used to deal with Big Data datasets, but it's important to be able to pick the right implementation for a given application.

Our main goal is to provide an overview of Big Data, NoSQL databases, three NoSQL data models, their specific implementations of our choice and benchmark them to get a comparative evaluation.

We have chosen to evaluate MongoDB, Redis, Apache Cassandra and do so, by using the Yahoo! Cloud Serving Benchmark framework, to help us compare different workload performance on each implementation.

II. Related Work

In this section, we will focus on pointing out related work. We will focus on papers very similar to ours, to identify patterns and reoccurring findings.

In the first related work, *Cooper et al.* present the Yahoo! Cloud Serving Benchmark framework, which is designed to compare different serving data stores. It can also be used to execute different kinds of datasets using different kinds of data serving systems, where the developer can either use predefined workloads or design them according to their own needs. The paper also contains detailed description of those datasets provided by YCSB. They also benchmark the performance of four cloud serving systems and conclude that since there are visible tradeoffs between read and write performance, they highlight the the importance of a standard framework for examining system performance, to be able to choose the right fit [1].

Abubakar et al. provide an evaluation of performance of four NoSQL data models, ElasticSearch, OrientDB, Redis and MongoDB usinf YCSB. The monitor the execution time of insert, update and read operations within these data models and highlight the importance of targeting the specific use cases and needs of each application. The researchers conclude, that Redis is the most performant for insert, ElasticSearch for update (but followed very tightly by Redis) and also for read operations [2].

Experimental comparative study conducted by *H. Mattalah*, uses the YCSB framework to benchmark the performance of MongoDB vs HBase NoSQL data models. His purpose is to provide assistance and support in deciding which Big Data and Cloud

Computing solutions they should adopt. Furthermore, he provides overview of NoSQL databases in general, the types of NoSQL databases and uses all the YCSB predefined workloads to compare the two data models. His findings show that MongoDB is more efficient in the read operations, whereas HBase is more performant in write operations. He claims that the estimated frequencies of reading, inserting, updating and data size are the essential factors in determining the selection of an alternative among others [3].

Khazaei et al. present a comprehensive theoretical and experimental survey on choosing the right NoSQL solution. In their extensive study, they present Big Data characteristics, NoSQL System features and a in depth explanation of each major NoSQL class. For each class, they elaborated on the main functionalities and characteristics and then introduced three to four dominant solutions. They concluded that selecting the right datastore is not a trivial task due to diversity and lack of standard benchmarks in this domain [4].

The last work examined, was the NoSQL databases comparison done by *N. Seghier and O. Kazar*, in which they used the YCSB tool once again. They focused on performance of three very commonly used databases, Redis, MongoDB and Cassandra, which is our objective, too. Their main goal wa to help developrs choose the right NoSQL solution for their application in the future. They discovered that that Redis is more efficient in read operations than MongoDB and Cassandra, whereas MongoDB proved superior performance in write operations.

III. Methodology

In this section we describe the approaches and strategies to complete this academic paper, in order to both understand the topic and provide experimental data and results.

Descriptive Research

At first, aiming to gain an understanding of the existing research and debates relevant to our topic, we are going to execute a descriptive research. We are going to provide a brief overview of the definitions, advantages, use cases and other important information.

At first, we are going to focus on Big Data in general, define their main properties and attributes and present challenges they bring.

Secondly, in order to understand the suggested Big Data challenges and to be able to address them, we are going to introduce basic characteristics of the NoSQL databases. We are going to focus on their advantages and mainly on how they solve the Big Data processing issues opposed to traditional SQL databases.

Thirdly, we will examine three different NoSQL data models: Document, Key-Value and Wide Column. We will provide their details, specifications and properties with respect to the general attributes of the Big Data architectures.

At last, to provide more detailed explanation of the implementations of data models we are going to experiment with, we will provide the implementation characteristics and general Big Data architecture goals of MongoDB, Redis and Apache Cassandra.

Benchmarking Experiment

After researching the topic and providing the needed information, we will focus on the practical part of our academic paper, which is benchmarking the performance of our selected implementations, MongoDB, Redis and Apache Cassandra. To do so, we will use the Yahoo! Cloud Serving Benchmark, which is a tool for benchmarking cloud-based database and storage systems. It was developed by Yahoo! Labs to allow users to evaluate the performance of different databases and storage systems in a standardized way. YCSB provides a set of workloads that can be used to measure performance, scalability, and availability of cloud-based systems. It also provides an extensible framework for the addition of new workloads and other features.

To develop our project, we will use GitHub to store our implementation, and Docker and bash-scripting to automate our solution, to be able to run it all at once and get the desired results.

We explore todo kinds of workloads, to get the performance results of all the three implementations. We will examine both latency, which is the time taken to execute different kinds of workloads, and the throughput, which is the number of number

of data packets being successfully sent per amount of time.

IV. Big Data Overview

Big data is a term used to refer to large datasets that are too complex and voluminous for traditional data-processing software to manage. Big data usually refers to data sets that are so large or complex that traditional data-processing applications are inadequate to deal with them. These datasets are usually characterized by high velocity, variety, and volume. Big data analysis is often carried out by using advanced analytics techniques such as machine learning, artificial intelligence, and data mining.

Challenges

Data Storage: Storing large amounts of data in an efficient manner for analysis is a challenge.

Data Mining: Mining the vast amounts of data and extracting useful information from it is a challenge.

Data Cleaning: Cleaning and pre-processing the data is a challenge due to its unstructured nature.

Data Security: Ensuring the security of data and protecting it from unauthorized access is a challenge.

Data Visualization: Visualizing the data in a meaningful way for better understanding is a challenge.

Data Analysis: Making sense of the large amounts of data and extracting insights is a challenge.

Advantages

Improved Decision Making: Big data helps companies to make better decisions by providing a large amount of data that can be analyzed and used to draw insights.

Improved Customer Experience: With the help of big data, companies can collect customer data and use it to customize products and services to better meet the needs of their customers.

Cost Reduction: Big data can help companies reduce costs by optimizing operational processes, identifying new cost saving opportunities, and increasing efficiency.

Improved Efficiency: Big data can help companies to better understand their operations and improve efficiency.

Improved Risk Management: Companies can use big data to analyze large amounts of data and identify potential risks. This helps them to mitigate the risks and make better decisions.

V. NoSQL Databases Overview

NoSQL databases are non-relational databases that don't use the traditional table-based relational database structure. Instead, they use flexible schemas which allow them to store data in a more unstructured way. In this section, we will provide brief overview, present some of the challenges their usage brings, but also the advantages of implementing them in systems.

Seeing both advantages and disadvantages will help us understand their complexity and realise how important it is to choose the right implementation. Because no implementation is perfect, there are always tradeoffs, whichever characteristics we take into account.

Challenges

Lack of Standard Interfaces: One of the major drawbacks of NoSQL databases is the lack of standard interfaces. Each vendor provides its own interface and protocols, making it difficult to switch between vendors.

Limited ACID Compliance: NoSQL databases generally lack the ACID (Atomicity, Consistency, Isolation, Durability) compliance found in traditional relational databases.

Weak Security Model: Many NoSQL databases lack a robust security model, which can lead to data breaches and other security risks.

Lack of Structured Query Language (SQL): NoSQL databases do not use Structured Query Language (SQL), which makes it difficult to access data from the database.

Data Model Complexity: The data models used in NoSQL databases are often complex and difficult to understand. This makes it difficult to design and develop applications that interact with the database.

Poor Performance: NoSQL databases often have poor performance, particularly when dealing with complex queries.

Advantages

Non-relational: NoSQL databases are non-relational, meaning that they do not use the traditional table-based relational model used in SQL databases.

Dynamic schema: NoSQL databases have a dynamic schema for unstructured data, which means that the data structure can change over time.

Scalable: NoSQL databases are typically more scalable than traditional relational databases, allowing for more concurrent users and more data to be stored without performance degradation.

Cloud-based: NoSQL databases can be run in the cloud. This makes them easily accessible and reduces the cost of hosting and managing the databases.

Easy to Use: NoSQL databases are generally easier to set up and maintain than relational databases because they don't have complex features like foreign keys and joins.

Flexible: NoSQL databases are more flexible, allowing developers to store data in a variety of formats, including documents, key-value pairs, and graph databases.

Process Large Data: NoSQL databases are often better suited for handling large amounts of unstructured data.

Cost Effective: NoSQL databases are typically more cost-effective than relational databases, since they don't require expensive hardware and software.

VI. Data Models

In this section, we are going to discuss three data models of our choice, introduce their properties and specifications.

Document Data Model

NoSQL document data models are databases that store data as documents instead of tables. Documents are JSON-like objects that contain key-value pairs, and can store data in a variety of formats such as arrays, strings, numbers, booleans, and dates.

This type of data model allows for faster data retrieval and flexible data schemas, allowing developers to quickly adjust their data models in response to changes in their application's requirements. Document databases are popular for applications that require low latency and high scalability, such as mobile and web applications.

Key-Value Data Model

NoSQL Key-Value data model is a type of database that stores data in the form of key-value pairs. It is a simple data model which allows for fast access of data.

It stores data as a collection of key-value pairs where each key is unique to the data item and can be used to retrieve the associated value. The key-value pairs are stored in a distributed data store and are managed by a database engine. Key-value databases are highly scalable and can be used to store large amounts of data. Key-value databases are also highly available and can be used for high availability applications.

Wide Column Data Model

NoSQL wide column data model stores data in columns and rows, similar to a traditional relational database. However, NoSQL wide column data model allows for more flexibility than a relational database, as the columns can be added and removed on the fly.

They can have high performance for operations such as data read and write operations, as well as for data queries. The performance of such operations is typically dependent on the size of the data set being queried and the number of columns stored in the database.

Generally, NoSQL wide column data models have faster read and write operations than traditional relational databases, as well as better scalability for large data sets. Additionally, the use of column families in wide column databases can help reduce the number of joins needed to execute a query, resulting in faster query performance.

VII. Data Models Implementations

In this section, we are going to discuss the properties of the three data models we have chosen to benchmark. We will compare their suitability for different systems, which will provide us the results we can expect when executing the benchmark.

MongoDB

MongoDB is an open-source document-oriented NoSQL database used for high volume data storage. It is classified as a NoSQL database because it does not rely on a traditional table-based relational database structure. Instead, it uses JSON-like documents with dynamic schemas. MongoDB is used to store large amounts of data in a distributed and decentralized manner, allowing applications to scale quickly and easily.

Here are some of the most important MongoDB properties:

High Performance: MongoDB provides high performance data persistence. It includes in-memory speed and on-disk durability.

High Availability: It provides high availability with replica sets.

Horizontal Scalability: MongoDB provides horizontal scalability as it can easily scale up and down with no downtime.

Rich Query Language: MongoDB allows for a rich and expressive query language that allows users to filter and sort data in any way they want.

Indexing: Any field in a MongoDB document can be indexed with primary and secondary indices.

Aggregation: MongoDB has powerful aggregation capabilities that allow for sophisticated data analysis and reporting.

Redis

Redis is an open-source, in-memory data structure store that is used for caching, message queuing, and other data storage needs. It is optimized for speed, providing very high performance for both reads and writes. Redis is a key-value store, meaning each data object is stored as a key-value pair.

With regard to NoSQL databases, here are some of the Redis properties:

Low latency: Redis has extremely low latency due to its in-memory storage.

High throughput: Redis can handle thousands of requests per second due to its optimized data structures.

High availability: Redis can maintain high availability due to its replication and clustering capabilities.

Flexibility: Redis can be used for a variety of use cases due to its powerful data structures and scripting capabilities.

Apache Cassandra

Apache Cassandra is an open source distributed NoSQL database management system designed to handle high volumes of data across many commodity servers. It provides high availability with no single point of failure.

Cassandra offers robust support for clusters spanning multiple data centers, with asynchronous masterless replication allowing low latency operations for all clients. It provides linear scalability and high performance with a read and write throughput both increasing linearly when additional machines are added.

Fault Tolerance: Apache Cassandra is designed to provide high fault tolerance as it automatically replicates data across multiple nodes in the cluster.

Scalability: Apache Cassandra can easily scale up to hundreds of nodes in a cluster and can be used to store large amounts of data.

Easy to Use: Apache Cassandra is easy to use and understand. It provides a wide range of tools and libraries to access and manipulate data.

High Availability: Apache Cassandra provides high availability by replicating data on multiple nodes in a cluster.

Fast Write Performance: Apache Cassandra offers a high write performance due to its log-structured storage engine.

Flexible Data Model: Apache Cassandra offers a flexible data model that allows users to store and query data in different ways.

Implementation Comparison

It's important to highlight that all three implementations have their own strengths and weaknesses, depending on the specific application requirements.

In terms of performance, Redis is generally faster, due to its in-memory storage engine, therefore it's a better choice for applications that require high performance and is often used for applications that require fast read/write operations.

However, MongoDB is more scalable, as it supports sharding and replication. MongoDB is better

for applications that need to store unstructured data and require dynamic queries.

Apache Cassandra is best suited for applications that require high performance and scalability. Cassandra is also popular for applications that require low latency and high availability.

VIII. Results

In this section, we are going to discuss our database benchmarking procedure results which we accumulated using the Yahoo! Cloud Service Benchmark. We are going to examine the performance of each implementation against different kinds of workloads and their read/write latency and throughput.

For benchmarking all three types of NoSQL databases we used an EC2 instance on Amazon Web Services cloud of type **t2.large**, which has following hardware and software parameters:

- CPU: 2 vCPU
- RAM: 8 GiB
- OS: Ubuntu Server 22.04 LTS
- Main storage: 8 GB gp2 SSD

Workload A: Update Heavy

Starting with a first workload, the parameters of it are following:

- Read/Update ratio: 50/50
- Default data size: 1 KB
- Request distribution: zipfian

This workload is marked as "Update heavy" and we can see that during the *Load* phase, where data are inserted and prepared Cassandra is falling way behind MongoDB, that tries to level with Redis. Once we take a look into *Run* phase results, it is clear, that Redis is winning with highest and more than double the throughput compared to MongoDB, beating Cassandra ten times. As an application example, we can consider Session store recording recent actions

Workload B: Read Mostly

Following the first benchmark comes second workload, the parameters of it are following:

- Read/Update ratio: 95/5
- Default data size: 1 KB
- Request distribution: zipfian

We can clearly see, that the biggest change in comparison with Workload A is the ratio of Read/Update operations. During the *Load* phase, MongoDB this time almost beats Redis, which is consistently sitting around the mark of 1600 ops/second. During the *Run* phase, Redis gets back its superiority, but this time MongoDB shows better number, supposedly because of mostly performing Read operations. Cassandra is stable with its ten times lower throughput in both phases. As an application example, we can consider photo tagging; add a tag is an update, but most operations are to read tags

Workload C: Read Only

Third workload comes with almost similar parameters as the second one, specifically:

- Read/Update ratio: 100/0
- Default data size: 1 KB
- Request distribution: zipfian

As we can see, in the *Load* phase there is almost no change in numbers for all databases, however during the *Run* phase we can see that MongoDB this time performs even better than in the second benchmark. We can assume, that this further throughput increase is caused by performing only read operations and not performing a single Update during second phase, which suits MongoDB better. As an application example, we can consider user profile cache, where profiles are constructed elsewhere (e.g., Hadoop).

Workload D: Read Latest

Workload with number four comes with following parameter settings:

- Read/Update/Insert ratio: 95/0/5
- Default data size: 1 KB
- Request distribution: latest

We might notice, that in the distribution of operations we can firstly see Insert operation being performed, even though it comes only in 5 percents of total operations performed. During the *Run* phase, we can clearly see the decline of throughput of MongoDB, as it performs unfavorable Insert operations, it records comparable numbers as with the second Workload. Cassandra still performs with low but very stable throughput of approximately one tenth of Redis. As an application example for this

Workload, one can consider user status updates; people want to read the latest.

Workload E: Short Ranges

Fifth workload comes with following set of parameters:

- Scan/Insert ratio: 95/5
- Default data size: 1 KB
- Request distribution: zipfian

With this workload we can see that during the *Load* phase MongoDB performs for the first and only time during our benchmark better than Redis, which is very interesting. Big shift in numbers comes in the *Run phase*, where we can see that MongoDB performs three times better in the throughput numbers than Redis. MongoDB might have an advantage here thanks to the fact, that the insert order is hashed, not ordered and although the scans are ordered, it does not necessarily follow that the data is inserted in order. Cassandra's performance plummets with a throughput low enough even for its own past results. As an application example of this worload, we can consider threaded conversations, where each scan is for the posts in a given thread (assumed to be clustered by thread id).

Workload E: Read-modify-write workload

The sixth and last Workload comes with following parameters:

- Read/read-modify-write ratio: 50/50
- Default data size: 1 KB
- Request distribution: zipfian

As we can see, Redis takes back the lead in performance, mostly thanks to fast modify operations. MongoDB in this case performs very poorly and Cassandra comes back to its numbers from previous benchmarks. As an application example in this case, we can consider user database, where user records are read and modified by the user or to record user activity.

IX. Limitations

In this section, we would like to discuss limitations that were present during the comparison. The first issue that we found was that we were not using a real cluster, but rather a simulation of a cluster by connecting Docker containers that run on a single machine. This could have impacted performance since the individual nodes communicate in a specific way when they are Docker containers.

Other limitation we see in the fact that there were only little replicas in order to simplify the process. In real world there would definitely be more replicas, which could make a difference in performance.

We are also concerned about the benchmarks being performed in a invariable environment (one specific type of hardware, one operating system, etc.). The limitation is that there was not enough diversity of environment in which the benchmarks were done.

The last limitation were the workloads themselves. Even though the workloads provided with YCSB are really powerful, they don't cover all possible scenarios. Furthermore, the benchmarking tool has not been updated for years now but the databases have evolved significantly.

X. Conclusion

As we could see during different Workloads in previous section, Redis wins in most of the cases, probably thanks to its technology of storing data in main memory, which, despite the fact of ongoing SSD dominance, still performs times faster than the main disk storage.

MongoDB shows very solid numbers, mostly thanks to its data handling approach, since it stores a large volume of unstructured data and follows a document-based storage. Thanks to that, it can be relatively faster than traditional relational database, as for example MySQL.

Cassandra with its lowest throughput numbers of the three database systems is maybe falling a little bit behind performance-wise, however for certain usecases it certainly can outweigh its disadvantages, for example for handling write-heavy workloads (transaction logging, time series data, tracking of inventory, Internet of Things status and events, tracking the weather and much more) where data is inserted and rarely updated.

References

[1] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM symposium on Cloud computing (SoCC '10). Association for Computing Machinery, New York, NY, USA, 143–154. https://doi.org/10.1145/1807128.1807152

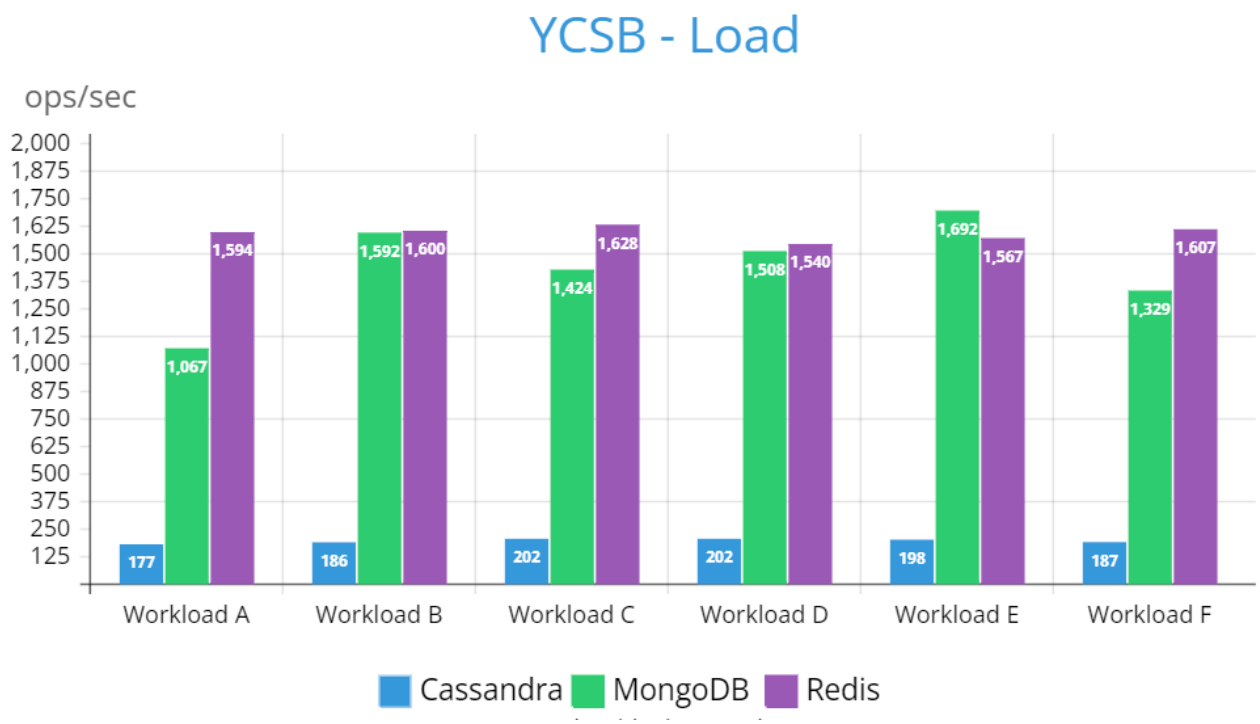

Fig. 1. YCSB - Benchmark ”Load” average throughput

[2] Yusuf Abubakar, Thankgod Sani Adeyi and Ibrahim Gambo Auta. Article: Performance Evaluation of NoSQL Systems using YCSB in a Resource Austere Environment. International Journal of Applied Information Systems 7(8):23-27, September 2014.

[3] Matallah Houcine, Belalem Ghalem, Bouamrane, K.. (2017). Experimental comparative study of NoSQL databases: HBase versus MongoDB by YCSB. Computer Systems Science and Engineering. 32. 307-317.

[4] Khazaei Hamzeh, Fokaefs, Marios, Zareian Saeed, Beigi Nasim, Ramprasad Brian, Shtern Mark, Gaikwad Purwa and Litoiu Marin. (2015). How do I choose the right NoSQL solution? A comprehensive theoretical and experimental survey. Journal of Big Data and Information Analytics (BDIA). 2. 10.3934/bdia.2016004.

[5] N. B. Seghier and O. Kazar, ”Performance Benchmarking and Comparison of NoSQL Databases: Redis vs MongoDB vs Cassandra Using YCSB Tool,” 2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI), 2021, pp. 1-6, doi: 10.1109/ICRAMI52622.2021.9585956.

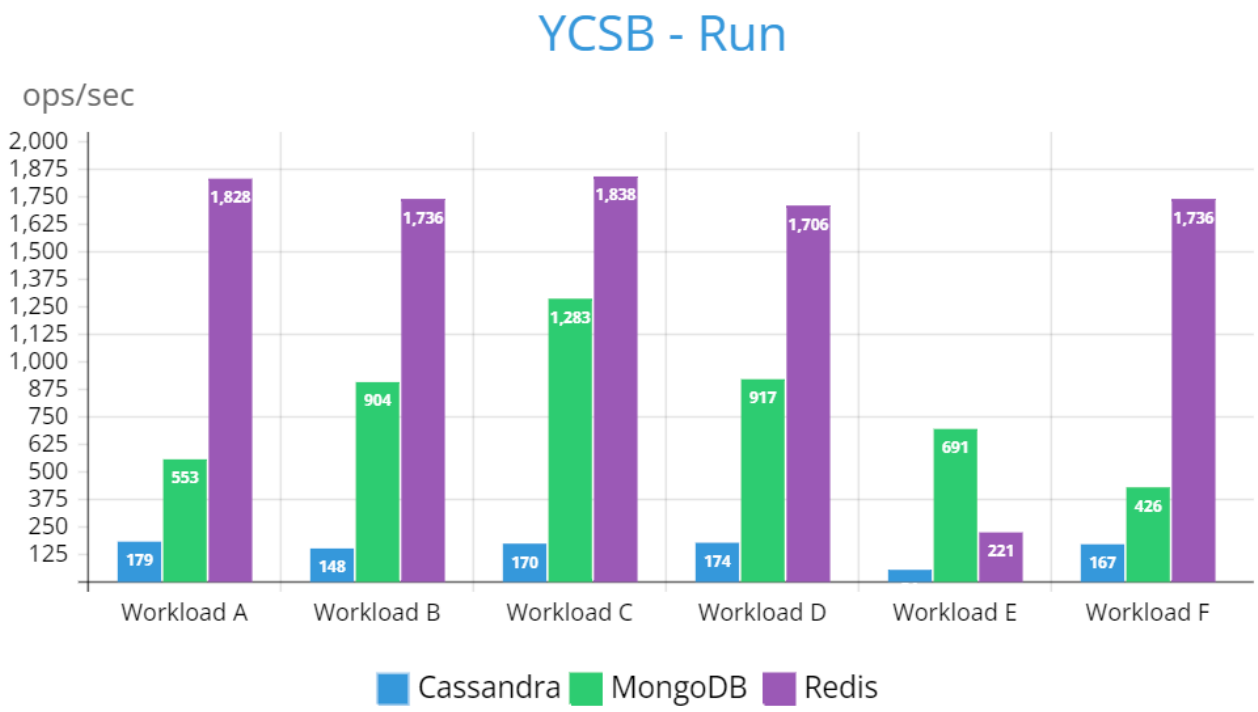

Fig. 2. YCSB - Benchmark ”Run” phase average throughput