ANKARA YILDIRIM BEYAZIT UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

ANALYSIS AND EVALUATION OF BIG DATA CLUSTERS IN DIFFERENT CLOUD COMPUTING ENVIRONMENTS

Ph.D. Thesis by

AIMEN MUKHTAR ALTAHIR RMIS

Department of Electrical and Computer Engineering

June, 2020

ANKARA

ANALYSIS AND EVALUATION OF BIG DATA CLUSTERS IN DIFFERENT CLOUD COMPUTING ENVIRONMENTS

A Thesis Submitted to

The Graduate School of Natural and Applied Sciences of

Ankara Yıldırım Beyazıt University

In Partial Fulfilment of the Requirements for the Degree of Doctor of Philosophy in Electrical and Electronics Engineering, Department of Electrical and Computer Engineering

by

AIMEN MUKHTAR ALTAHIR RMIS

June, 2020

ANKARA

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"ANALYSIS AND EVALUATION OF BIG DATA CLUSTERS IN DIFFERENT CLOUD COMPUTING ENVIRONMENTS"** completed by **AIMEN MUKHTAR ALTAHIR RMIS** under the supervision of **ASST. PROF. AHMET ERCAN TOPCU** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Ph.D.

Asst. Prof. Ahmet Ercan TOPCU

Supervisor

Asst. Prof. Mustafa YENIAD

Jury Member

Asst. Prof. Ali Osman ÇIBIKDIKEN

Jury Member

Asst. Prof. Hilal KAYA

Jury Member

Prof. Dr. Hasan Şakir BILGE

Jury Member

I hereby declare that, in this thesis which has been prepared in accordance with the Thesis Writing Manual of Graduate School of Natural and Applied Sciences,

- All data, information and documents are obtained in the framework of academic and ethical rules,
- All information, documents and assessments are presented in accordance with scientific ethics and morals,
- All the materials that have been utilized are fully cited and referenced,
- No change has been made on the utilized materials,
- All the works presented are original,

and in any contrary case of above statements, I accept to renounce all my legal rights.

Date: June, 2020 **Signature** **:……..**

Name & Surname: AIMEN MUKHTAR ALTAHIR RMIS

ACKNOWLEDGMENT

My gratitude is deeply paid to my advisor, Asst. Prof. Ahmet Ercan TOPCU, of the Graduate School of Natural and Applied Sciences, of Ankara Yıldırım Beyazıt University, for his continuous supervision, sharing his experience, encouragement, and guidance throughout my study.

Also, I would like to thank the members of my thesis committee, for their support and valuable suggestions that made great contributions to this work.

Finally, the greatest thanks go to my family members for their infinite support. This thesis is dedicated to them. and my gratitude is extended to all my friends and colleagues who provided me with every help they can.

June, 2020 **AIMEN MUKHTAR ALTAHIR RMİS**

ANALYSIS AND EVALUATION OF BIG DATA CLUSTERS IN DIFFERENT CLOUD COMPUTING ENVIRONMENTS

ABSTRACT

Cloud technology has become the supreme way to deliver enterprise applications and the preferred solution for companies launching innovations or extending their infrastructure. NoSQL databases are often used to provide availability and flexibility for big data processing. However, in distributed and parallel processing environments, there has not yet been a comprehensive evaluation of NoSQL databases. NoSQL was developed to better meet the data storage needs of a large number of records, but there are different kinds of NoSQL technologies, and most have not been thoroughly compared. Understanding the performance of NoSQL databases is essential for the selection of the correct database for a specific application. Several databases have emerged that can handle the cloud domain to take advantage of the flexibility of the cloud environment. The aim of this dissertation is to develop a model to evaluate and analyze NoSQL databases using a cloud environment where these systems are deployed (i.e., in a cloud or a local data center cluster). This model is then evaluated with the most popular NoSQL databases in cloud computing (MongoDB and Riak KV) and internal cluster environments to analyze the performance of those two systems. A number of aspects of performance, including monitoring throughput and latency, are compared. The model architecture has the ability to use other databases as well as other clouds. An evaluation of the model system in three cloud computing infrastructures is also presented. The model architecture can use other databases as well as other clouds. Furthermore, a decision tree is constructed based on many experiments to determine the performance criteria for the databases on the selected cloud computing platforms. The main findings of this work are as follows: (i) Performance gain is achieved by integrating MongoDB with Google Cloud, especially in terms of reading and writing a massive amount of data in a cluster environment; (ii) The infrastructure of Google Cloud is more manageable and efficient in dealing with the distributed environment and big data, and it achieved better results than the other clouds; (iii) The results also showed the limitations of Riak KV in both the Google and OpenStack clouds, especially in terms of update latency.

Keywords: NoSQL; Cloud Computing; MongoDB; Riak KV; big data; cluster; YCSB; Basho-Bench.

BÜYÜK VERİ KÜMELERİNİN FARKLI BULUT BILIŞİM ORTAMLARINDA ANALİZLERİ VE DEĞERLENDİRİLMESİ

ÖZ

Bulut teknolojisi, kurumsal uygulamalar sunmanın en iyi yolu ve yenilikler başlatan veya altyapılarını genişleten şirketler için tercih edilen çözüm haline gelmiştir. NoSQL veritabanı genellikle büyük veri işleme için kullanılabilirlik ve esneklik sağlamak amacıyla kullanılmaktadır. Ancak, dağıtılmış ve paralel işleme ortamlarında, NoSQL veritabanının henüz kapsamlı bir değerlendirmesi yapılmamıştır. Bunun yanısıra NoSQL çok sayıda verilerin depolama ihtiyaçlarını daha iyi karşılamak için geliştirilmiştir. Ancak farklı NoSQL teknolojileri bulunmakta ve bunların birçoğu ayrıntılı olarak karşılaştırılmamıştır. Belirli uygulama alanlarında doğru veritabanının seçimi için NoSQL veritabanlarının performansını belirlemek çok önemli olmaktadır. Bulut ortamının esnekliğinden yararlanmak amacıyla bulut bilişim alanını kullanarak işleyebilen çeşitli veritabanları ortaya çıkmıştır. Bu tez çalışmasında sistemlerin dağıtıldığı bir bulut ortamını (yani bir bulutta veya yerel bir veri merkezi kümesinde) kullanarak NoSQL veritabanlarını değerlendirerek analiz etmek için bir model geliştirilmiştir. Sonrasında, bu sistemin performansını analiz etmek için bulut bilişim ve iç kümeler ortamlarında en popüler NoSQL veritabanlarıyla (MongoDB ve Riak KV) bu modeli kullanarak değerlendirmeler yapılmıştır. Performansın ve gecikmenin izlenmesi de dahil olmak üzere performans değerleri farklı parametrelerle değerledirilerek karşılaştırma yapılmıştır. Bu geliştirilen model mimarisi, diğer bulutların yanı sıra diğer veritabanlarını da kullanma yeteneğine sahiptir. Ayrıca üç farklı bulut bilişim altyapısında model sisteminin değerlendirilmesi yapılmıştır. Model mimarisi, diğer bulutların yanı sıra diğer veritabanlarını da kullanabilmektedir. Ayrıca, bulut bilişim platformlarındaki (CCP) bu veritabanlarının performans kriterlerini belirlemek için birçok deneye dayanarak karşılaştırma yapan karar ağacı oluşturulmuştur. Bu çalışmanın ana bulguları şu şekildedir: (i) MongoDB'yi özellikle bir küme ortamında büyük miktarda veri okumak ve yazmak açısından Google Cloud ile entegre ederek elde edilen performans kazançları; (ii) Google Cloud altyapısı, dağıtılmış ortam ve büyük verilerin kullanılmasında daha iyi yönetilebilir ve verimli olduğu ortaya çıkmaktadır ve diğer bulut altyapılarından daha iyi sonuçlar vermiştir; (iii) Sonuçlar ayrıca Riak KV'nin hem Google Cloud hem de OpenStack bulut alytapılarında, kullanımına yönelik olarak özellikle güncellem gecikmesi açısından Riak KV'nin performans açısından sınırlarını göstermiştir.

Anahtar Kelimeler: NoSQL; Bulut bilişim; MongoDB; Riak KV; Büyük veri; küme; YCSB; Basho-Bench.

CONTENTS

NOMENCLATURE

Acronyms

DBMSD	Data Base Management System
SQL	Structured Query Language
NoSQL	Not Only SQL
RDBMS	Relational Data Base Management System
KV	Key Value
CC	Cloud Computing
IoT	Internet of Things
CP	Cloud Platform
CCP	Cloud Computing Platform
API	Application Programming Interface
ACID	Atomicity, Consistency, Isolation, Durability
CS	Cloud Storage
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
YCSB	Yahoo Cloud Serving Benchmark
NIC	Network Interface
VM	Virtual Machine
Vnodes	Virtual Nodes
JSON	JavaScript Object Notation
AWS	Amazon Web Services
SSH	Secure Shell

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Databases have recently appeared as repositories with structured and organized data, in which all data are combined into a set of registers arranged into a regular structure to facilitate the extraction of information. To access data it is common to use a system, usually known as a database management system (DBMS). A DBMS can be defined as a set of mechanisms for storing, editing, and extracting data; in the past few years, the concept of the DBMS has become synonymous with databases. The size and the complexity of the database are defined by the number of registers used. A simple database can be represented as a file containing data, while a more complex database can store millions of records with a large number of gigabytes globally [1]. Databases are increasingly becoming important enterprise tools. Storage types, functions, and interactions with databases have improved over the past few years with the development of information and communication technologies, and databases have become resources that millions of people use every day in countless applications. All of these benefits and usages must be built and organized in an optimal way for quick and easy extraction. Whenever the amount of data increases, the database grows more substantial in size. With the exponential growth of database size, access to data has to be made as efficient as possible. That leads to the well-known problem of efficiency in information extraction. Today, the amounts of big data that can be managed by the systems known as NoSQL, corresponding to "Not Only SQL," like the Riak system, outstrip what can be achieved by the most extensive relational database management systems (RDBMSs). NoSQL databases are essentially created from the ground up to require less management like data distribution or automatic repair. Simpler data models also lead to lower administrative burdens [2]. NoSQL databases mostly use clusters of cheap commodity servers to handle the exploding data and transaction volumes, while relational databases tend to rely on costly private servers and storage systems. When using NoSQL, the cost per transaction or gigabytes per second can be many times less than the cost for RDBMSs, permitting us to save money and process more data at a much lower price [3].

NoSQL key-value (KV) stores, as well as document databases, let the application save practically any structure it needs in data elements. Strictly determined “Bigtable”-based NoSQL databases (HBase, Cassandra) characteristically allow new columns to be created without too much confusion. The large number of NoSQL offerings consequently leads to the problem of differentiating between these offerings and their suitability in different circumstances [4].

Cloud computing (CC) is a valuable technology for delivering large-scale and complex computing. It eliminates the requirement of keeping costly computing hardware, software, and dedicated space. A huge increase in the scale of data or big data produced by CC has been observed. Addressing huge data is a challenging and time-consuming task that needs major computational infrastructure to guarantee successful data analysis and processing. CC is one of the most significant shifts in modern information communication technology, and services for enterprise applications have become reliable architectures for implementing complex and large-scale computing. The benefits of CC include virtualized resources, parallel processing, security, and data service integration with scalable data storage. CC can not only minimize the cost and restrictions for automation and computerization by individuals and enterprises; it can also provide reduced infrastructure maintenance costs, efficient management, and user access [5, 6].

1.1 Motivation and Research Problem

With the growth of CC technology, traditional network models cannot meet the needs of cloud services. The generation of massive amounts of data continuously at very high speeds in different domains, such as the growth of the Internet of Things (IoT), multimedia, and social media, has produced an amazing flow of unstructured and semi-structured data and these data are complex and heterogeneous. Effective processing and analysis remains a high priority. This challenge includes the techniques used in the software and hardware to process the data efficiently. Different areas require processing and analysis, such as finance, engineering, business, science, healthcare, and society. With the growth and creation of data rapidly and increasingly at record rates, this is referred to as “big data.” The emergence of many NoSQL databases and

various CC applications has become a generally recognized trend. The phenomenon of CC big data is eliciting interest from academia, industry, and governments.

CC has many problems and challenges; the relevant research problems can be listed as follows:

- Different NoSQL technologies exist with many various products developed by companies. In this situation, engineers might be confused while trying to decide which type of database to be used to deal with big data and its problems within their systems, and determining which architectures and solutions best fit big data is still an open problem. There has not been much research to date on cloud platform (CP) solutions for big data that provide a practical, experimentally driven characterization of the efficiency and suitability of distributed databases.
- Currently, there are various NoSQL databases in existence and they are prominent for handling big data. However, it should be understood that every database uses different approaches and has been designed to meet specific needs. Except for non-relational databases, these databases do not share any commonalities and may not have the best performance and consistency results in a given situation. Therefore, it is necessary to examine the behavior of different databases in detail under different workload conditions.
- There are many possible configurations for big data clusters. The question is how to effectively and objectively evaluate all of these configurations to ultimately choose the best setting based on the requirements of the end user. This presents the challenge of finding out which factors affect the performance of a cluster configuration and how much they affect the performance.
- Not all NoSQL solutions provide a modernized architecture suitable for specific applications, and this is particularly the case for applications that require high scalability, continuous availability, and data distribution. Datacenter support and, more commonly, multi-data center support should be a use case that NoSQL environments follow.

1.2 Research objectives and contributions

The main objective of this dissertation is to propose a framework for a big data cluster based on a CP. This framework will be used to analyze and evaluate NoSQL databases such as Riak KV and MongoDB. The framework can be used with different workload conditions and different datasets to get the best performance results. Work is presented for using mixed CPs and NoSQL databases to evaluate the performance of the combination of different databases and CPs using this framework.

The essential contributions of this research are:

1. To generate a fictitious workload and a data access pattern on the cluster that matches the workloads of real-world applications and monitor its performance in a CP.
2. To observe the performance of NoSQL databases (Riak KV, MongoDB) with large data volumes and various workloads (read, update, mix of reads and updates).
3. To monitor the performance of NoSQL databases (throughput, latency) when data are being read, inserted, and scanned and during update operations.
4. To create a decision tree for developers and database operators to choose the best CP for applications.

1.3 Dissertation organization

This dissertation is organized into six chapters, as follows:
Chapter 1: As an introduction, this chapter presents an overview of the main motivations, the research problem statement, and research objectives and contributions.

Chapter 2: Presenting the related works, this chapter focuses on a literature review of NoSQL databases, big data, CC, and other related work.

Chapter 3: Addressing the methodology, this chapter describes in detail the steps followed to make the recommended framework.

Chapter 4: This chapter on the experimental analysis includes a full description of the databases that have been used and clouds and experiments applied in different scenarios.

Chapter 5: Presenting the results and a discussion, this chapter evaluates and analyzes the results presented in Chapter 4 and gives a decision tree for a big data cluster based on a CP.

Chapter 6: The final chapter summarizes the presented work and concludes with the findings and contributions of the dissertation and possible future works.

CHAPTER 2

LITERATURE REVIEW

This chapter briefly defines the background of this work, including an overview of challenges, while explaining big data, NoSQL databases, CC types, models, and characteristics. Several related works are reviewed and discussed.

2.1 Big Data

We will begin by exploring the term "big data," which is closely related to NoSQL database systems. Big data can be described as the ability to manage large amounts of data at the right time and at the right speed. The term describes a huge amount of unstructured, semi-structured, and structured data that can be mined for information, and such data cannot be managed using RDBMSs [7, 8].

The idea of big data has become a significant engine for growth and innovation that depends on emerging technologies such as IoT, cloud computing, and analytics. Big data is thus essential for enhancing output increases in the world since it affects software-intensive industries, education, administration, and health. Every day, new data are created from a variety of sources, including social networks, photos, videos, and more. Due to the rapid growth of data, it has become complicated to process data through the available DBMSs. One of the solutions proposed to overcome the rapid growth of data is to apply better hardware; however, this approach is not enough because hardware enhancements have reached a point where the amount of data grows more than computer resources [9, 10]. Big data can be found in three forms:

- Structured data: Data that can be saved, obtained, and processed in a fixed format are called "structured" data. Over time, people in computer science have achieved greater success in developing technologies that process such data. There are two types of sources providing structured data: data are generated by human interference, such as input data and game data, and data are produced by devices, such as weblog data, financial data, and sensor data [6, 11].

- Unstructured data: Before the current ubiquity of online and mobile applications, databases processed direct, structured data. The data forms were relatively simple and described a set of relationships between various data types in the database. In contrast, “unstructured” data are data that are not fully suited to the traditional column-and-row structure of a relational database. In today’s big data world, most of the data created are unstructured, by some estimates accounting for more than 95% of all data generated [12].
- Semi-structured data: These data are a combination of structured and unstructured data. It is not easy to deal with this level of data complexity. Big data and extensive records lead to long-running queries; hence, we need new methods and techniques to overcome this challenge and manage large amounts of data [13].

The nature of big data is often described using four significant characters: volume, variety, veracity, and velocity, sometimes referred to as the 4 V’s of big data. Each of these 4 V’s has its own impact on data analysis. Figure 2.1 shows the 4 V’s of big data.

2.1.1 Volume

Volume is the amount of all types of data that are generated and continuously expanded from different sources. Most datasets are too large to be stored and analyzed using traditional RDBMS technology. As a result, deficiencies and weaknesses emerged in traditional databases. Therefore, distributed systems are an important new type of technology. For example, 43 trillion gigabytes of data were created by 2020 according to a study by the McKinsey Global Institute, and it is estimated that 2.3 trillion gigabytes of data are generated every day [14,15].

2.1.2 Variety

Variety indicates the various types of data collected through social networks, smartphones, or sensors. Such data types include audio, image, video, text, and data logs, in either unstructured or structured format. Previously, all data were in the formal data category, stored neatly in tables and RDBMSs. Currently, however, most of the world’s data generated by Internet service companies and many other organizations are unstructured data [15].

2.1.3 Velocity

Velocity describes the speed at which new data are produced, saved, analyzed, and visualized. In the big data era, data are generated in near real-time or real-time. Due to the absorption of supplementary datasets, previously archived data, or legacy collection sets, streamed data from multiple sources are introduced. The content of the data is continuously changing. For example, per minute, email users send 204 million messages, Facebook users share 2.46 million items, YouTube users upload 100 hours of video, and Google receives more than 4 million search queries [16,17].

2.1.4 Veracity

Veracity refers to the uncertainty of the data. Various amounts of data from different sources change at high velocity at different speeds. Organizations must ensure the accuracy of the data, or in other words, the veracity, fidelity, and authenticity of the data, because incorrect data can cause severe problems for organizations and customers [18].

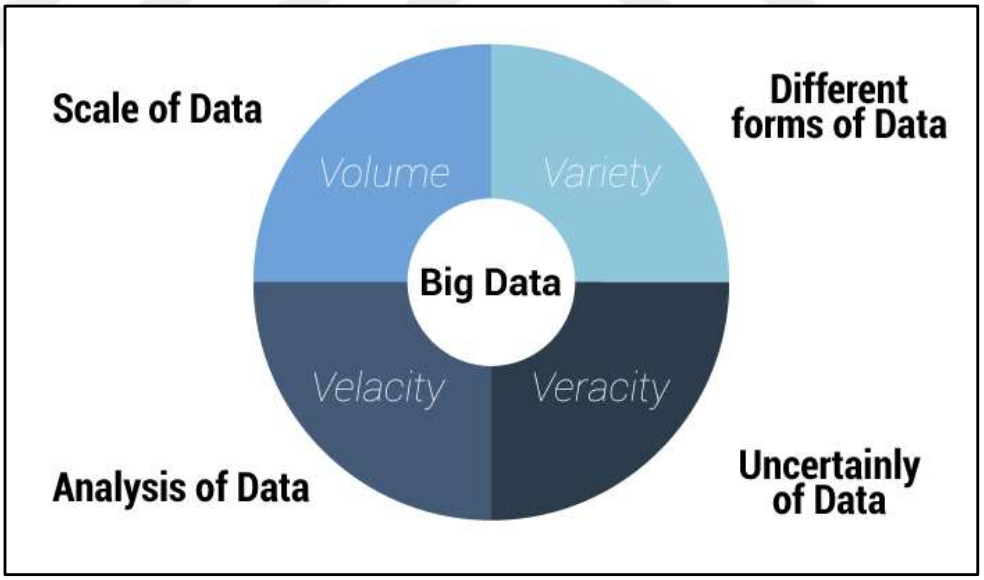


Figure 2.1 Four V's of big data

2.2 NoSQL Databases

High-level databases can be divided into two categories of RDBMSs and distributed databases, which provide alternatives to management and architecture systems based on the type of data that needs to be saved and manipulated.

A relational database is a set of formalized tables from which data can be reorganized or accessed in many various ways without having to rearrange the database tables. The Application Programming Interface (API) and standard RDBMSs use Structured Query Language (SQL). SQL statements are used both to interactively query information in RDBMSs and to collect report data. Each table (sometimes called a relationship) in a relational database includes one or more data categories in a column [19, 20].

NoSQL ("Not Only SQL") is a term that describes the entire database class. It does not have the characteristics of a traditional relational database and usually does not use standard SQL. NoSQL databases are considered to be next-generation databases, and they support massive data storage with horizontally scalable, open-source, distributed databases and massively parallel data processing [21, 22].

NoSQL databases can be classified into four categories, as follows:

- Key-value (KV): In general, NoSQL databases allow the use of various types of relational data tools. These are becoming common in new business plans and big data analysis, in which detailed data should be stored practically and efficiently [23]. Within this context, KV-store databases are the most straightforward NoSQL databases. They can help developers in the absence of a predefined schema. Different kinds of objects, data types, and data containers are used to accommodate this [20, 24]. High query speed with a simple structure, where the KV is the data model, supports benefits such as high concurrency and mass storage. Data modification and query operations are well supported through primary keys such as Riak KV [25] and Redis [26]. The following is an example of KV code.

Example KV databases

KEY	VALUE
"100"	Ali/25/Libya/IT
"101"	Ahmed/30/Syria/IT
"102"	Rahaf/32/Turkey/CS

- Column-oriented: A table in a column-oriented database can be used for the data model; however, this stores tables of extensible records. It includes columns and rows, which may be shared by being divided over nodes. In general, the benefit of this data model is a more appropriate application for aggregation and data warehouses. HBase [27] and Cassandra [28] are examples of this kind of data store.

Examples of column-oriented databases

CustomerID	**Column Family: Identity**
001	First name: Ahmed Last name: Ali
002	First name: Sohail Last name: Aimen Suffix: Jr.
003	First name: Ahmed Last name: Rame Title: Dr.

CustomerID	**Column Family: Contact Info**
001	Phone number: 99539789 Email: xxxxxxx@gmail.com
002	Email: sohail@gmail.com
003	Phone number: 5555555555

- Document data stores: Also known as document-oriented databases, these programs are used to retrieve, store, and manage information. The data are semi-structured. The document database can usually use the secondary index to facilitate the value of the upper application, however. The KV and document database structures are very similar, but they differ in how they process data. The name is derived from the way it stores data, as it stores documents in XML or JSON format [20, 29]. Couch and MongoDB [30] are examples of document data stores. The following is an example code for document data stores.

Example code for document databases

```
" phone":{
        " type": " apple Phone ",
        " color": " Black ",
        " brand": " ACME ",
        " price": " 300 ",
        " quantity": " 50 "
},
```

- Graph databases: A graph database comprises nodes that are connected by edges. Data can be stored in edges and nodes. One advantage of a graph database is that it can traverse relationships very quickly. Like the three other types of NoSQL databases mentioned above, graph databases have some problems with horizontal scaling. Therefore, every node can connect to any other node. Traversing nodes on various physical machines can have a negative effect on performance. Another difference from the three previously listed types of databases is that most graph databases support ACID (atomicity, consistency, isolation, and durability) transactions. Graph databases are often used to deal with complex issues such as social networks or path-finding problems [29], such as Neo4j [31].

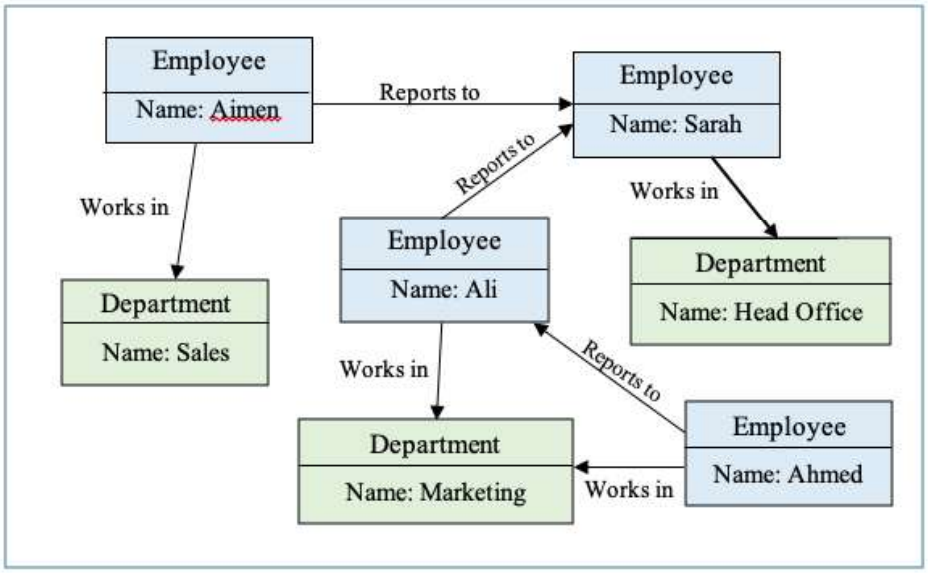


Figure 2.2 Example graph databases

2.3 Cloud Storage (CS) and Cloud Computing (CC)

CS is a model of data storage in which the digital data are stored in logical pools, and the hosting companies normally own and manage the physical environment that has the physical storage, including multiple servers and locations. These CS providers are responsible for maintaining the availability and accessibility of the data and protecting and operating the physical environment.

CS facilities can be retrieved through a co-located cloud computing service, a web service API, or applications that utilize the API, such as cloud storage gateways, cloud desktop storage, or web-based content management systems [32].

CC is a new computing model that brings together all models of businesses, disciplines, and technologies to provide information technology resources on request. It is used to work on and complete specified projects. CC is linked with CS as the data must be moved to the CS before CC systems can be used. However, once the data are transferred to the cloud, they can be processed as useful material and sent to other users.

2.3.1 Cloud computing features

CC has many characteristics that make it essential to the big data industry, such as:

- Resources pooling: This means that cloud providers can leverage multi-tenant models to generate the computing resources to serve multiple customers. There are different virtual resources and physical allocations and redistributions, depending on the needs of the customer [36].
- Availability: The functionality of the cloud can be adjusted depending on the purpose and can be increased significantly. It analyzes the storage usage and allows the user to buy extra Cloud storage if needed for a very small amount [36, 37].
- Lowering operating costs: Cloud environment resources can be allocated and deallocated on request, which can save considerable operating costs because resources can be released when the service request is low [38].

- Pay-as-you-go: In cloud computing, users only pay for the services or space they already use. There are no hidden or extra fees to pay. The service is economical, and most of the time, free space is allocated [38, 39].
- Easy to access: Cloud service is offered to users as a web-based service. Therefore, they can access the service by any device supported by an Internet connection [36, 38].
- Economical: This is a one-time investment because the company (host) must purchase storage, and a small portion can be offered to many companies, saving the monthly or yearly costs of the host. The amount spent is used for essential maintenance and some fees, which represents lower costs [37, 38].
- Scalability and elasticity: Infrastructure providers have many resources and considerable infrastructure. As a result, they can quickly scale their services to handle growing service requests based on customer needs. This is scalability. Elasticity, on the other hand, is the ability to scale resources up and down when needed and it allows for dynamic integration and extraction of physical resources to the infrastructure [40].

2.3.2 Cloud Service Model

There are three popular CC models. According to the requirements, the user can choose any of the three models, as seen in Figure 2.3 [33, 35].

- Infrastructure as a Service (IaaS): This type of CC provides users with access to computing resources such as servers, networking, and storage. Organizations use their applications and platforms via service providers.
- Platform as a Service (PaaS): This is a CC product that provides users with a cloud environment in which they can develop, deliver, and manage applications. In extensions to storage and different computing resources, users can use a set of pre-built tools to develop, test, and customize their applications.
- Software as a Service (SaaS): This is a CC product that gives users access to produced cloud-based software. Users do not install applications on their local devices. Instead, the application is located on a remote network that is available

through the web or API. Using the application, users can analyze and store data and help on projects [34, 36].

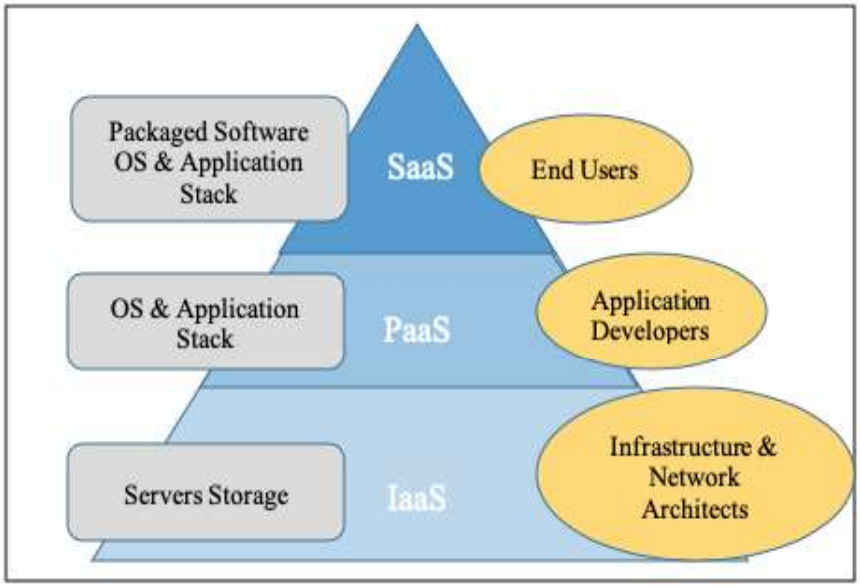


Figure 2.3 Cloud service models

2.3.3 Cloud deployment model

There are various kinds of clouds, each with their own advantages and disadvantages, as shown in Figure 2.4.

- Public cloud: A public cloud is a virtual resource pool developed from hardware managed and owned by a third-party company that automatically configures and distributes these resources across multiple clients through a self-service interface. This is an easy way to extend workloads that experience unusual request fluctuations [41].
- Private cloud: Also known as an internal data center, a private cloud is dedicated to a single organization. Private clouds can be built and managed by external providers or organizations. Because this technology is operated and owned by the company, this kind of cloud is more expensive than public clouds, but it is also more secure.
- Hybrid clouds: A hybrid cloud is an integration of private and public cloud models that works to address the characteristics of each approach. In a hybrid cloud, some

of the service infrastructures operate in a public cloud while the rest run in a private cloud. Hybrid clouds enable companies to keep critical secret information and data within their firewalls while leveraging public clouds for non-confidential data [42, 43].

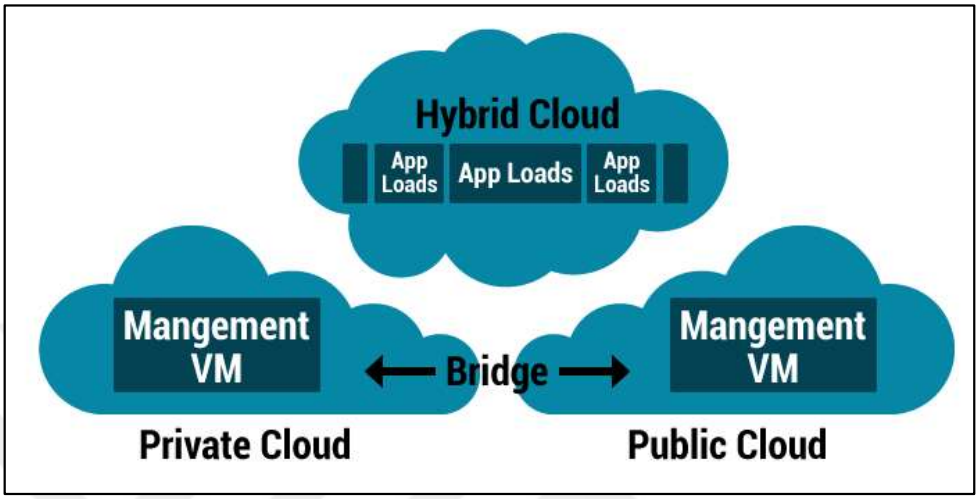


Figure 2.4 Types of clouds

2.4 Relationship between CC and big data

CC and big data are integrated together. Big data offers users the efficiency to use service computing to process distributed queries across several datasets and suitably return output result sets. CC provides the first engine through the use of Hadoop, a class of distributed data-processing platforms [44, 45]. The use of CC with big data is illustrated in Figure 2.5. Big data sources from the cloud and the web are saved in a distributed fault-tolerant database that is processed by large-dataset programming models with parallel distributed algorithms in the cluster. As shown in the figure, the primary purpose of data visualization is to view the analysis results presented by different graphics for decision-making. Big data leverages cloud-based distributed storage technology rather than local storage connected to electronic devices or computers. Big data assessments are driven by fast-growing cloud-based applications developed using virtualization technology. Therefore, CC not only facilitates the calculation and processing of big data but also serves as a service model [45].

Amazon Web Services (AWS), Google BigQuery, and Azure HDInsight are among the most common platforms for big data analytics and leverage both CC and related technologies [46].

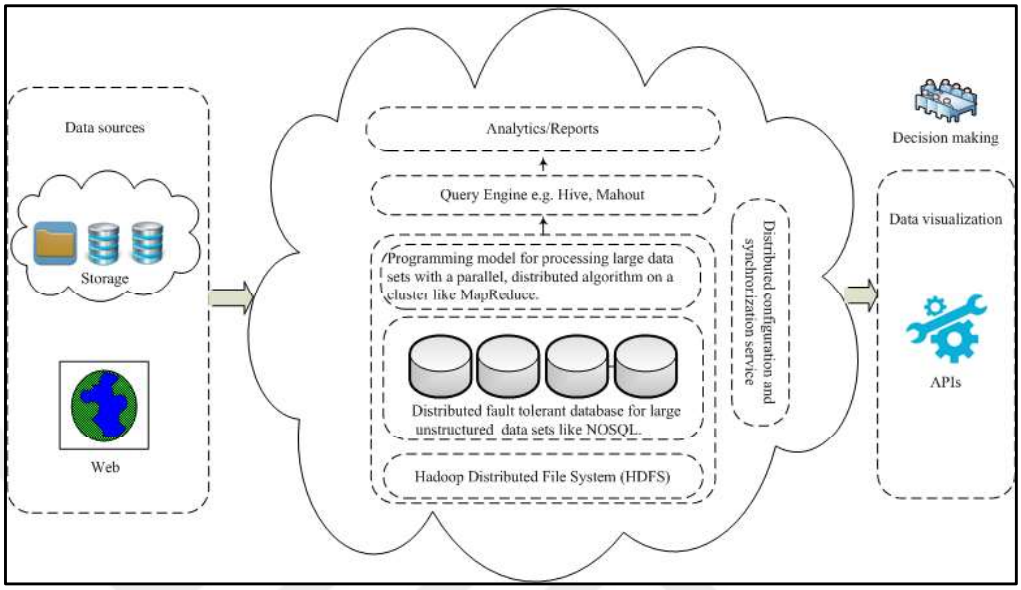


Figure 2.5 Cloud computing usage in big data

2.5 Benchmarking

Performance measurement tools are essential for both users and designers of DBMSs, whether they are proposed for online analysis processing (OLAP) or online transaction processing (OLTP). Performance evaluations are helpful for designers to define the elements of the architecture and, more commonly, to verify assumptions about the actual behavior of the system. Therefore, performance evaluation is a necessary part of a well-designed and scalable system development process, which is most important in today's cloud computing environment. Users can also utilize performance assessments to compare the efficiency of various technologies before choosing a software solution or adjusting the system. Performance evaluation through experiments on real systems is often associated with benchmarking. It involves performing a set of tests on a given system to evaluate the performance in a given setting. Generally, a database benchmark requires two main factors: the data model (extensions and conceptual architecture) and the workload model (set of write and read operations) that will be applied to the dataset relative to the predefined protocol. Most

benchmarks also include a set of composite or straightforward performance metrics, such as throughput, response time, disk number for inputs/outputs, memory usage, etc. [90].

Benchmarking is a way to find the best performance, whether for a particular company, a competitor, or a completely different industry. This knowledge can then be applied to identify the gaps in organizational rules to gain a competitive advantage [47].

Benchmarking is commonly used and it is a wise practice to establish benchmarks, determine the best methods, identify opportunities for improvement, and build a competitive environment within an organization. Integrating benchmarks into an organization will generate valuable data that can promote discussion and stimulate new ideas and methods. In the best case, it can be used as a tool that helps companies prioritize and evaluate chances for improvement [48].

Benchmarking can allow us to:

- Compare the performances of different databases.
- Specify performance gaps to identify areas for improvement.
- Produce a standardized set of metrics and processes.
- Set performance expectations.
- Monitor NoSQL database performance and manage change.

Benchmarking is not a straightforward way to solve problems. Instead, it gradually addresses the problem area. Without repeatable, clear benchmarks, companies or individuals may be blinded by arrogance. Both maintainers and contributors rely on benchmarks to guarantee that their performance is not negatively affected. Usually, having no baseline is like having no logging or metrics. The ability to benchmark workloads gives information about growth and has a positive impact on all users, not just the makers of the original benchmarking program. When considering distributed systems, it is necessary to choose the actual topology. Determining the right device is also essential. In the deploying of distributed systems, unique configurations are frequently recommend such as the fastest memory and storage nodes. Databases have an incredible variety of cases and workloads. Logically, there is no universal or all-

inclusive benchmark. Instead, we can use various tools like the Yahoo Cloud Serving Benchmark (YCSB) or Basho-bench [49].

2.6 Related works

This section introduces some related works from the literature on this topic. As a critical property of cloud computing, many works have analyzed and evaluated big data in CC and distributed systems through NoSQL databases.

For example, George et al. [50] compared the database and workload type performances in cloud environments and investigated MongoDB, Cassandra, and HBase in a set of usage scenarios based on various kinds of workloads. They also selected some measurements and outcomes in relation to every system's strengths to handle its own traffic. To this end, the YCSB benchmark client was applied to provide a general workload to evaluate the performance of various NoSQL databases. The operations were performed in BonFIRE cloud computing. The results may be summarized as follows:

- MongoDB is especially useful in the case when most of the operations performed are reads. However, the system has a particular weakness in scanning and retrieval workloads.
- Cassandra is the more stable solution, although the change in workload maintained a satisfactory and stable performance in all cases.
- HBase is severely affected by the choice of temporarily storing data on the heap, especially when combined with the limited memory resources in the available nodes.

In the work of Li and Manoharan [51], the KV-store implementations in SQL and NoSQL databases were compared. Although NoSQL databases are usually optimized for KV-stores, SQL databases are not. However, these authors found that not every NoSQL database achieves better results than the tested SQL databases. Even with NoSQL databases, the performance can vary greatly depending on the kind of operation, such as reading and writing. It was also observed that there was little

correlation between performance and the data model used by each database. RavenDB and CouchDB performed poorly in read, write, and delete operations in NoSQL databases. Casandra's read operations were slower, but Cassandra was quite good for delete and write operations. Couchbase and MongoDB were the two fastest for read, delete, and write operations. However, Couchbase does not support getting all keys (or values). If the application does not need to traverse all keys and values, then Couchbase will be the right choice. Otherwise, MongoDB can be selected, which is second only to Couchbase in read, delete, and write operations.

Abramova et al. [52] analyzed and evaluated two of the most popular NoSQL databases: Cassandra and MongoDB. In their experiments, they tested execution times based on workload type and database size. They experimented with six different kinds of workloads. As the amount of data increased, MongoDB started to have decreased performance, sometimes with poor outcomes. On the other hand, Cassandra got faster while working with an increase in data. After running different workloads to analyze read/update performances, it was possible to conclude that when update operations are involved, Cassandra is faster than MongoDB, providing lower execution times independently of the database sizes used. The overall analysis revealed that MongoDB fell short with increasing records being used, while Cassandra still had much to offer. In conclusion, Cassandra showed the best results for almost all scenarios.

Ou and Chen [53] presented a series of decision procedures for choosing the IaaS in CC from the perspective of technology startup companies. They targeted startup companies as potential users and their paper presented a decision tree to help developers choose the most appropriate CP technology for their needs. They chose two emerging platforms, Docker and OpenStack, to be evaluated as competitors and they adopted the YCSB to evaluate the performance of the KV-store. Software engineering perspective metrics were used to assess the availability of the two target platforms. The authors provided a decision tree for developers to choose the CP based on the experiments to benchmark the KV-store review and performance according to the usability. Based on the analysis and evaluation, they produced decision trees for seven issues to assist developers in making the best choices to meet their needs. In [54], MongoDB and HBase were compared in terms of read, update, and insert operations

tested on two different workloads. The authors used the YCSB, which is a framework designed by Yahoo, to test database performances. The outcomes were abstracted and simply analyzed to explain that the overall throughput of MongoDB is significantly much better than that of HBase as the overall throughput for HBase is half the value of that of MongoDB in terms of overall throughput. Thus, much better performance can be obtained with MongoDB than HBase. Furthermore, with the same results for different workloads, MongoDB can be considered better. Niyizamwiyitira and Lundberg [55] evaluated the performance of SQL and NoSQL DBMSs including MongoDB, Cassandra, CouchDB, PostgreSQL, and RethinkDB. These authors used a four-node cluster to run a database system with an obvious load generator. The experiments were performed using three differently sized datasets and they evaluated write throughput and latency and read throughput and latency for four queries (i.e., distance query, K-nearest neighbor query, range query, and area query). The results of the study showed that for write operations, Cassandra was a maximum throughput when multiple nodes are used, while PostgreSQL was the lowest latency and highest throughput for a one node. For read operations, MongoDB was the lowest latency for all experiments, Cassandra was the highest throughput for read. They used the real workloads of Cassandra, MongoDB, CouchDB, PostgreSQL, and RethinkDB instead of simulated workloads.

Ahmed et al. [56] studied the deployment of Spark clusters as cloud services on OpenStack-based clouds. The Hi-Bench benchmark kit was used for comparing the performances of the Spark cluster as a service and the conventional Spark cluster. They also conducted an in-depth analysis of the performance of the Spark cluster. The comparison was made using the complete big data benchmark suite known as Hi-Bench. The results clearly described how Spark as a cloud service provides good outcomes in terms of duration, effort, and throughput. In other research, Yang et al. [57] designed a significant monitoring solution based on the existing Hadoop structure combined with OpenStack clouds. A number of different metrics were gathered from the Hadoop metrics subsystem and all data were stored in the HBase database to make it ready for processing or applications with the MapReduce paradigm. These authors presented a general view of the situation of the infrastructural resources and the network services running on OpenStack. To assess the feasibility of the cloud platform

monitoring scheme, the OpenStack platform was deployed in the laboratory. The CP consisted of five nodes, including a control node, a network node, and three compute nodes with many VMs. The results of the study showed that Hadoop was the best choice for offline processing, i.e., when the whole dataset is huge. The results also showed the feasibility of monitoring the OpenStack platform and offering it as a cloud service with cloud-centric introspection compared to the current single-node processing approaches. Lu and Zhou [58] conducted a study of big data on a cloud agile provision framework with a focus on discussing the mode of automatic orchestration and competent in achieving the associated management of OpenStack private cloud. The VM and a big data platform could attain agile delivery, fast resource reclaiming, and high delivery quality. The repeatable delivery process and the capability of being integrated with monitoring, inspection, and configuration management databases were positive findings of the work.

In [59], the authors presented a control system for NoSQL databases of truly elastic behavior. It is important to note, however, that there was a modular cloud-enabled framework for monitoring and adaptively resizing the NoSQL clusters. The system includes a decision-making module that allows for optimal cluster resize operations to maximize any quantifiable award function provided together with life-long adaptation to workload or infrastructural changes. The public can start HBase clusters of different sizes and use different workloads through multiple YCSB clients. Users will be able to watch in real time as the system makes automatic VM additions and removals and can also see how cluster performance metrics change relative to the optimization parameters of their choice. In another study, Kumar et al. [60] presented the performance analysis of the OpenStack cloud with commodity computers in big data environments. To determine performance measures, virtual systems created in the OpenStack cloud and the Hadoop distributed framework were configured for all virtual instances and commodity computers to form their respective clusters. For the experiments, image to pdf conversion and a word count program were executed in the same configuration of OpenStack cloud and commodity computers. The authors found that for personal computers the Hadoop cluster was as powerful and fault-tolerant as the cloud Hadoop cluster, but not as scalable as the cloud cluster. This study concluded that the data storage and analysis in the Hadoop cluster in the cloud are more flexible

and more easily scalable than in the real system cluster, and the cluster in commodity computers is faster than the cloud clusters. Abramova et al. [61] tested the performance of Cassandra based on several factors, including the number of nodes, workload characteristics, number of threads, and data size. They analyzed whether it provided the desired acceleration and scalability attributes. Scaling nodes and the number of datasets do not guarantee performance. However, Cassandra handles concurrent request threads well and extends well with concurrent threads. A summary of the results of that study showed that when the number of nodes in a cluster increases from 1 or 3 to 6, even for relatively large datasets, this trend cannot guarantee an improvement in performance. Tsuyuzaki and Onizuka [62] discussed the characteristics of NoSQL databases and their benchmark systems. They reported results from running a benchmark on a MongoDB database. They tested it for elasticity and availability and revealed that data size had a significant impact on database performance when the system was extended or when machines were taken offline.

Klein et al. [12] introduced a method and provided the results for choosing among three NoSQL database systems for a large and distributed healthcare organization. The performance assessment methods and results were determined for the following databases: MongoDB, Cassandra, and Riak. The testing was based on the YCSB benchmark for evaluating NoSQL databases. The authors concluded that the Cassandra database provides the best throughput performance with the highest latency.

CHAPTER 3

METHODOLOGY

This chapter presents the techniques necessary to achieve the goals and overcome the problems and challenges presented in the first chapter. First, a general structure is presented that will be used to evaluate and analyze the performance. The characteristics of the used NoSQL databases, Riak KV and MongoDB, are explained in detail. Three different clouds have been chosen (OpenStack, DigitalOcean, and Google Cloud) and they are also presented in detail here. To generate different loads of data, benchmark tools are used (Basho-bench and YCSB).

3.1 System Architecture

This section explains the design requirements for the functional architecture for performance evaluation and testing. The experiments conducted in this study included three main components of databases, benchmarks, and CC.

Faced with the challenges of traditional RDBMSs in handling big data and meeting cloud needs, many specialized solutions have emerged in recent years to address these issues. NoSQL data stores emerge as alternatives to data processing that can handle large amounts of data and provide the scalability needed. To conduct extensive and accurate experiments to determine the best performance evaluation of the databases used in this research in a distributed database environment, the experiments were conducted in different environments in an internal cluster as well as in more than one cloud while using a benchmark for generating a common set of workloads for evaluating the performance of different databases. There are some criteria and mandatory parameters that must be used in configuring each node that is part of the cluster and each node that connects to the cluster as a client:

- **Network Interface (NIC):** The NIC allows the virtual machine (VM) to communicate with the virtual network.
- **External/Internal IP:** These have the same purpose, but the difference lies in the scope. The entire Internet uses IP external addresses to locate

computer systems and devices. To find computers and devices connected to it, the IP internal address is used inside a private network.

- **Operating System (OS):** The OS used is determined based on the node. In this research, Linux Ubuntu 14.04.5 is used.
- **RAM/Disk:** The size of the RAM and the size of the disk are determined in the node.
- **Security:** Network security groups and routes can be defined in each node.
- **Instance/Project Name:** This is displayed in the cloud console. The name must be unique within each cloud project.

The system architecture used is shown in Figure 3.1.

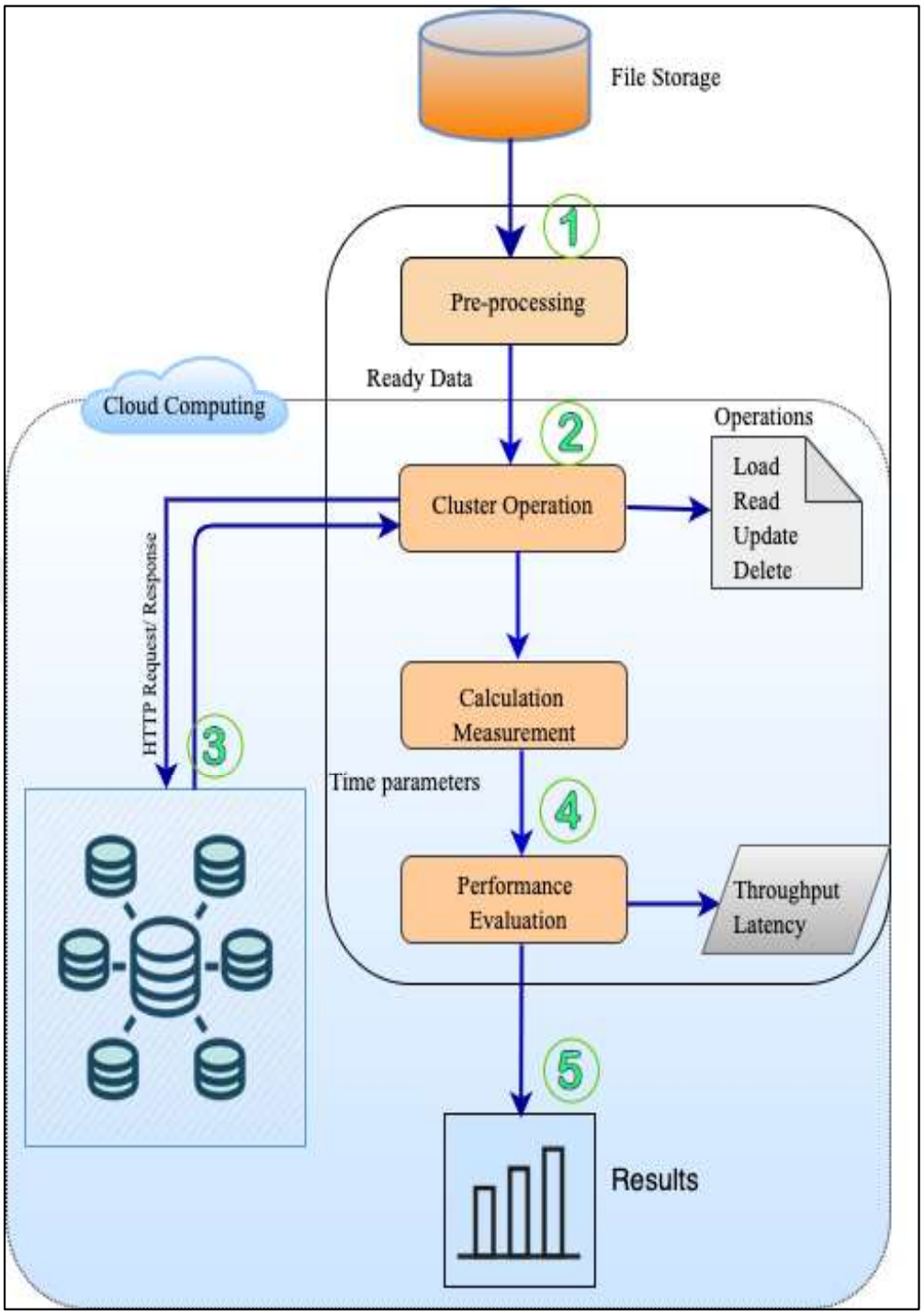


Figure 3.1 The model system architecture

The above figure shows a general architecture of the model system divided into many parts:

1. Pre-processing

In the first step in building the system model, we prepare the file storage, whether it is a commodity or VM, and next we install auxiliary software (OS, benchmark software, etc.) and then NoSQL databases, as shown in Figure 3.2.

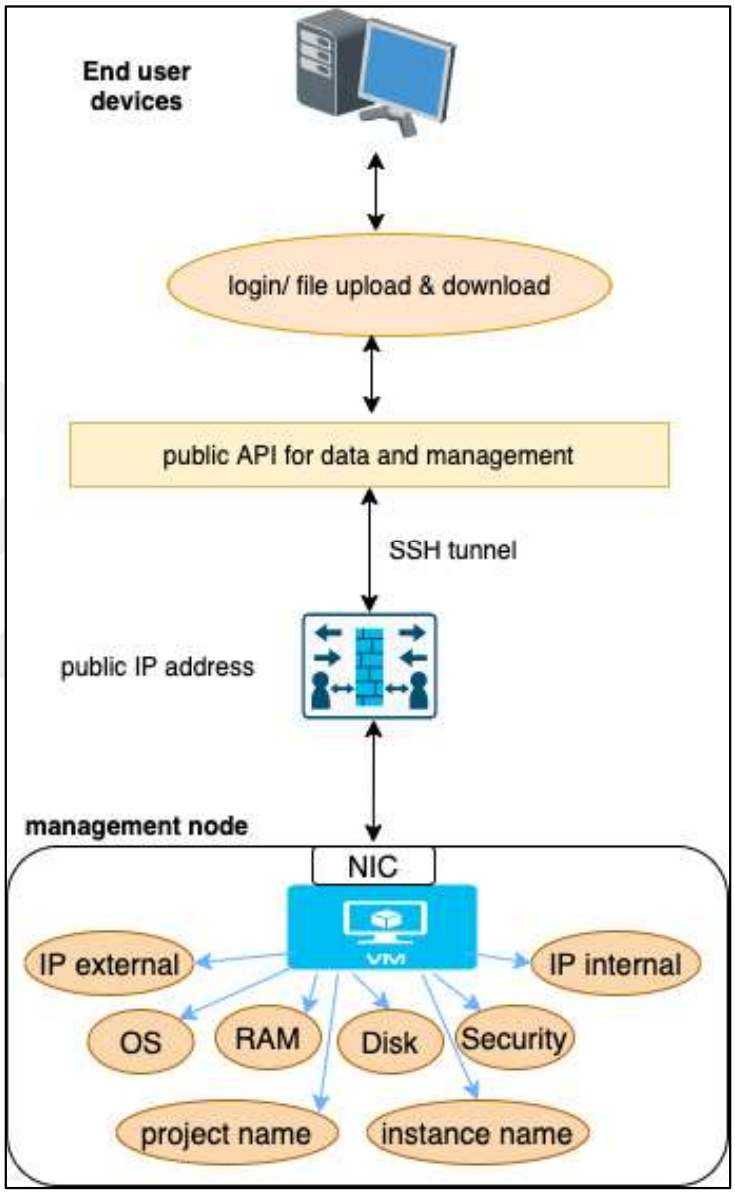


Figure 3.2 Pre-processing operation

2. File Operation

We prepare a benchmark file that consists of various operations, which will later be run on the various NoSQL databases to be tested. The number of operations is equal

to the record count for each database. We also choose the number of threads to run the desired workload. We do this for varying proportions of data, from small to big data. With each of these datasets we run the number of chosen operations (read, update, delete, and other) as shown in Figure 3.3.

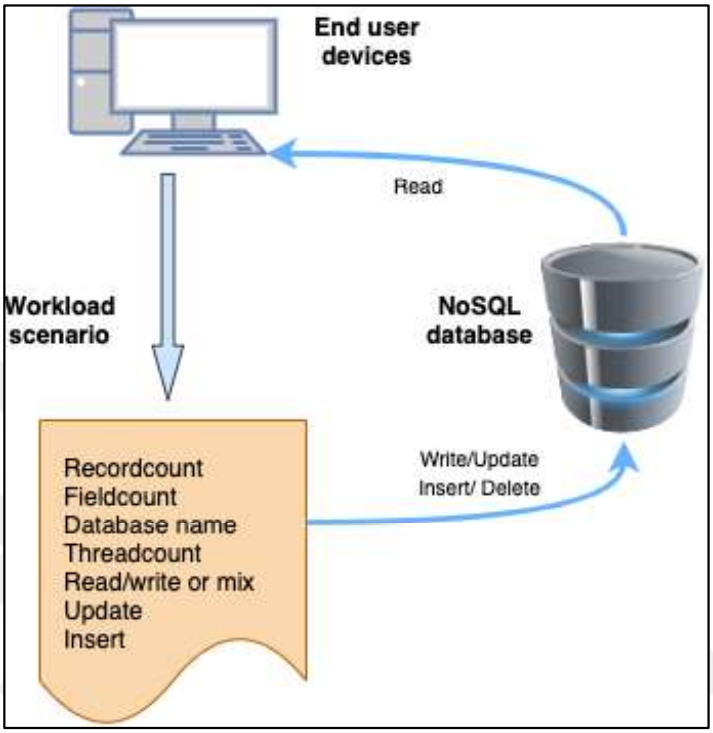


Figure 3.3 File operation

All file operation can specify the following properties:

- Database name to use: Alternatively, this may be specified on the command line.
- Thread count: Number of client threads; alternatively, this may be defined on the command line.
- Record count: The number of records in the dataset at the start of the workload.
- Field count: The number of fields in a record.
- Field length: The size of any field.
- Min field length: The minimum size of each field.

- Read proportion: Determines the percentage of data reads. The following describes a read operation:

```
"private void Read(I-Configuration config)
        {
            I-ObjectContainer oc =
Db4oFactory.OpenFile(config,
            DatabaseFile);
            I-ObjectSet objectSet = oc.Query(typeof(Item));
            while (objectSet.HasNext ())
            {
                Item item = (Item) objectSet.Next();
            }
            oc.Close ();
        }"
```

- Update proportion: Determines the percentage of data updates. The following describes an update operation:

```
"private void Update (IConfiguration config)
        {
            IObjectContainer oc = Db4oFactory.OpenFile
            (config, DatabaseFile);
            IObjectSet objectSet = oc.Query(typeof(Item));
            while (objectSet.HasNext())
            {
                Item item = (Item)objectSet.Next ();
                item.Change ();
                oc.Store (item);
            }
            oc.Close ();
        }"
```

- Insert proportion: Determines the percentage of data inserted. The following describes an insert operation:

```
"private void Insert(int itemCount, IConfiguration config)
        {
            IObjectContainer oc = Db4oFactory.OpenFile
           (config, DatabaseFile);
            for (int i = 0; i < itemCount; i++)
            {
                oc.Store (Item.NewItem (i));
                if (i % 100000 == 0)
                {
                    oc.Commit ();
                }
            }
            oc.Commit ();
            oc.Close ();
        }"
```

- Delete proportion: Determines the percentage of data deleted. The following describes a delete operation:

```
private void Delete(IConfiguration config)
        {
            IObjectContainer oc = Db4oFactory.OpenFile
            (config,  DatabaseFile);
            IObjectSet objectSet  = oc.Query  (typeof(Item));
            while  (objectSet.HasNext())
            {
                oc.Delete(objectSet.Next  ());
                oc.Commit  ();
            }
            oc.Close  ();
        }
```

3. Cluster Architecture

We prepare the datasets and run them on the cluster, as explained in the previous step. We monitor the progress of the operations to avoid any problems that may occur at the beginning of the operations; see Figure 3.4. We rely on two types of network distribution:

- Master-slave: A multi-node NoSQL database cluster with master-slave replication by automatic failover configured between them. This structure typically requires an odd number of members to guarantee that the right (primary) master database is selected. This selected database will handle all incoming write operations and store data about them to be accessed and replicated by each (secondary) slave replica member for applying them to their datasets. In this way, all servers will represent the same content and guarantee its availability [63, 64].
- Master-less: Having the right delivery model depends on the application requirements. If necessary, availability may be an issue, and a master-less network is the most suitable solution. If a batch job that runs outside of business hours can be used to manage big data, it is better to use a simpler master-slave model. These systems share the responsibility of the master among every node in the cluster. In this case, testing is more comfortable because it is possible to remove each node in the cluster, and the other

nodes will remain to function. The problem of master-less networks is that there is growing communication and complexity overhead, which is necessary for all nodes to be kept up to date about the cluster situation [65].

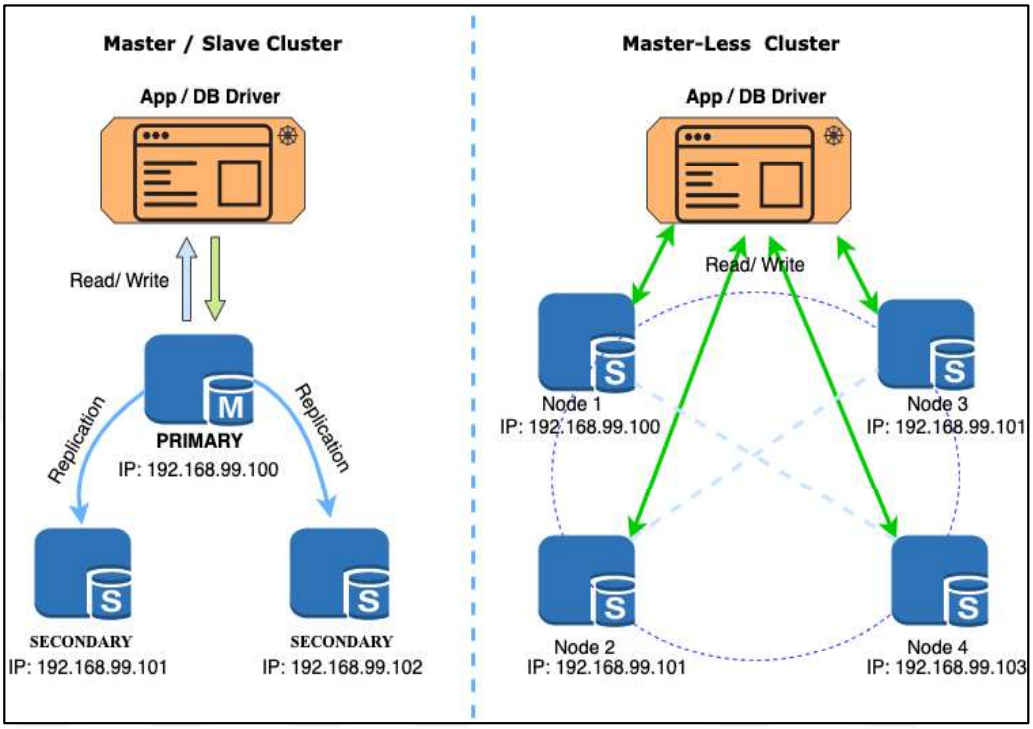


Figure 3.4 Cluster architecture

4. Calculation Measurement

We calculate the measurements to evaluate the performance of the NoSQL databases based on two metrics of performance:

- Throughput: Throughput is the rate of production or the rate at which something can be processed. In benchmarking contexts, it is the number of operations executed in unit time. It is the overall operations performed in the run phase.
- Latency: Latency testing can vary from application to application. In some implementations, measuring latency requires unique and complex equipment or knowledge of special computer commands and programs. In other cases, latency can be measured with a benchmark like YCSB or

Basho-bench. It can be defined as time to complete single operations, captured in quintiles per operation.

5. Results

The same experiments were run multiple times to verify that the obtained results were robust. Those results are presented here using graphs, tables, and decision trees. We will also discuss how the results were analyzed, as shown in Figure 3.5.

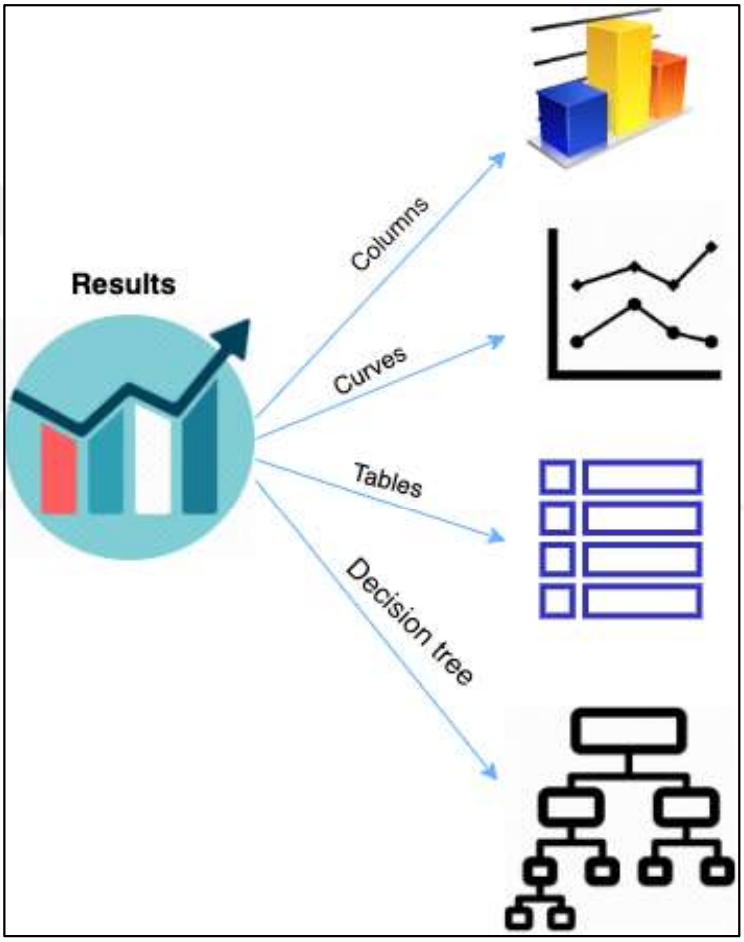


Figure 3.5 Representation of results

3.2 Technologies and Tools

The following section describes the main techniques and tools that are critical to the design and implementation of the proposed research; Figure 3.6 shows the architecture of the main technologies and tools.

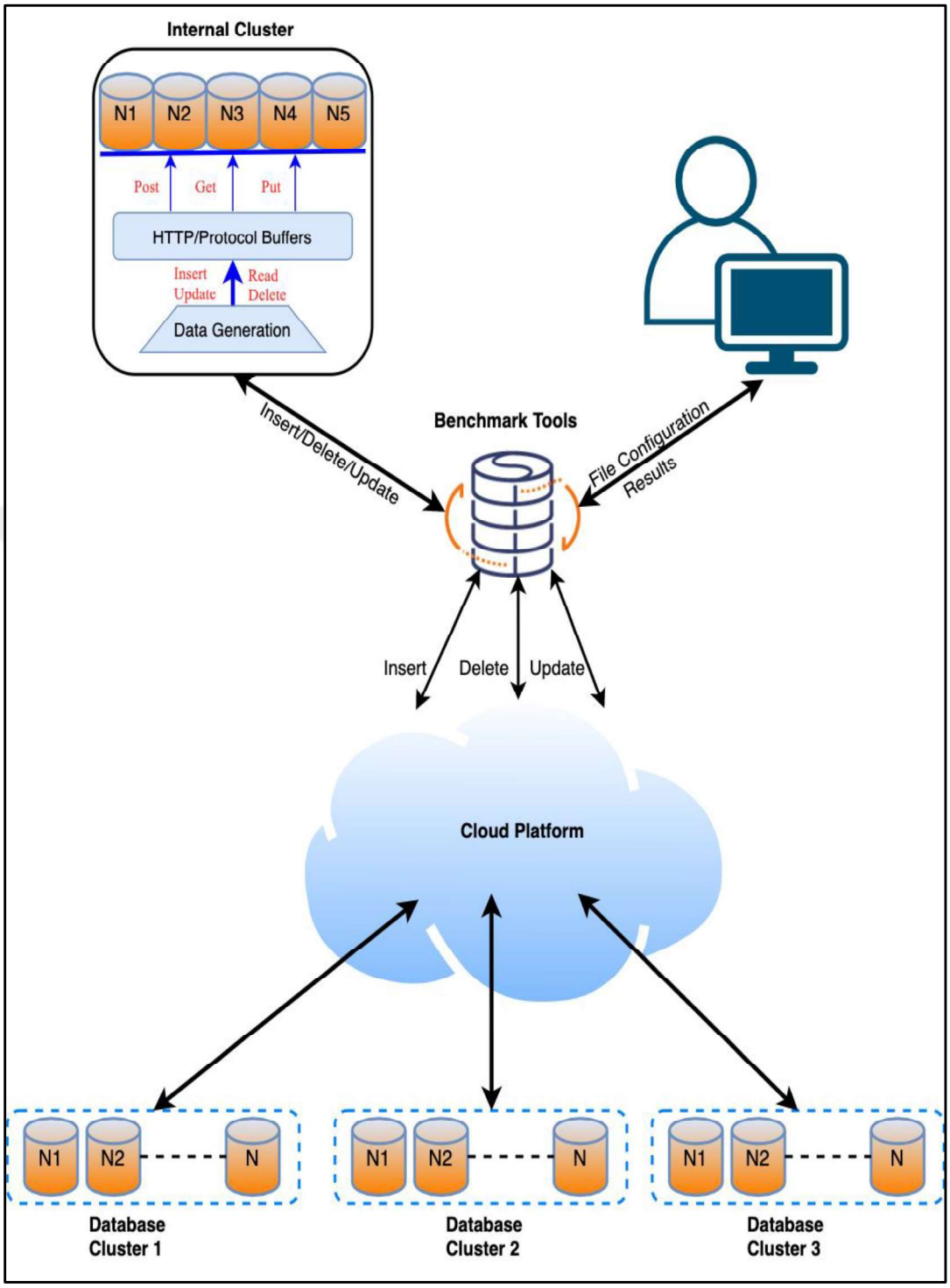


Figure 3.6 Main architecture diagram

3.2.1 NoSQL databases

The two NoSQL databases evaluated in this research have different feature sets. Each database provides a core set of characteristics that can be used as a basis for performance evaluation.

3.2.1.1 Riak KV

A KV database or KV-store is a simple database used for managing, retrieving, and storing associative arrays, a data structure more commonly known today as a hash table or dictionary. Dictionaries include a group of records or objects. They contain many different fields, each of which contains data. These records are retrieved and stored using a key that uniquely identifies records for quickly finding data in the database. In general, a KV database does not have a query language. They provide a way to update, retrieve, and store data using simple put, delete, and get commands [66].

Riak is a KV database. It is an open-source enterprise version of Riak Enterprise DS. It is a KV database developed by Basho in 2007 and written in the Erlang and C programming languages. The enterprise version adds multi-data center monitoring, replication, and additional support [67].

Riak is a distributed NoSQL database that is extremely scalable, available, and straightforward to work with. It automatically assigns the data in a cluster to ensure quick performance and fault tolerance. Riak Enterprise offers multi-cluster replication that guarantees low latency and strong business continuity. Riak KV is an appropriated distributed NoSQL KV database that ensures read and write functions even in cases of hardware failure or network partitions by supporting both local and multi-cluster replication. Riak KV is designed to work and deal with a combination of challenges facing big data applications, including following session or client data, storing data from connected devices, and replicating data around the world. It is designed with KV to provide a powerful, simple data model to store large amounts of unstructured data [66, 68].

Riak KV achieves fast performance and robust business continuity by automating data distribution across the cluster, where capacity is easily added without a large operational burden with a master-less architecture that guarantees high availability and

scales that are nearly linear using commodity hardware [68]. Nodes in Riak form a cluster. This cluster is isolated into partitions and virtual nodes (Vnodes) to form a ring to obtain all the benefits of Riak. The ring is a 160-bit integer space separated into similarly sized partitions, as shown in Figure 3.7.

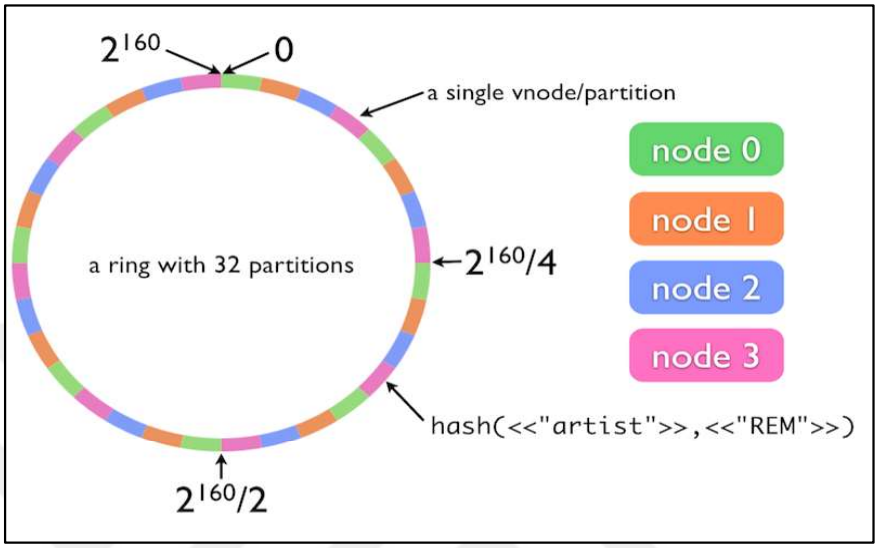


Figure 3.7 Architecture of the Riak cluster

Each node (also called a physical node) in the ring runs a certain number of virtual nodes (Vnodes). Each Vnode occupies one partition in the ring. It defines the partition size of the ring when configuring Riak or when the cluster is initialized [69].

- **How does Riak KV store data?**

Riak KV stores the data in buckets as a combination of keys and values. A key is simply a binary value that uniquely identifies a value. Values are the data associated with the key and saved. Values can be strings, numbers, binary data, or almost any data. The bucket is used to determine a virtual namespace used to store Riak KV "objects." In Riak KV, "object" is a nickname for a set of buckets, keys, and values. Every bucket/key entry can point to a different entry to form an entry link, and these entries can be accessed by link traversal via the Riak HTTP interface [70].

- **Riak drivers and client libraries**

Riak now supports eight client libraries that can be applied to the interface with Riak. Users can choose any language with the supported client libraries. The choice of

libraries depends on the programmer and application comfort. The supported languages are:

1. **Erlang:** Since Erlang is the development language of Riak, it is tightly mixed with the HTTP and protocol buffer interfaces of Riak.
2. **JavaScript:** JavaScript is an official language for querying Riak that uses JQuery in order to interface to Riak.
3. **Java:** Supports Java language for interfacing. The client can combine Java-based use by working with Java Client for communicating with Riak.
4. **PHP:** Package interface with the PHP language using the PHP client.
5. **Python:** Can interface with the Python language using the Python client.
6. **Ruby:** Ruby is likewise a formally supported language for Riak interfacing.
7. **C#:** A C# Riak client library for use with C++ compilers.
8. **Node.j**s: A client that makes it simple to communicate with Riak.

- **Riak cluster installation and configuration**

Before starting, it is necessary to install Riak KV on all VMs or nodes. By default, Riak KV is not available in the Ubuntu-14.04 default repository. Therefore, it is necessary to add a repository for this, as seen in Appendix A.

3.2.1.2 MongoDB

MongoDB is an example of a document-oriented database. It is difficult to save massive amounts of unstructured data using a traditional RDBMS. The primary storage components in a document database such as MongoDB are sets, rather than tables as in the case of RDBMSs. These collections in MongoDB consist of different or similar JSON/BSON-based documents or sub-documents. The documents that have some similarities in structure are ordered into sets. This can be done when needed, without any pre-definition. Instances within instances or documents with in documents, even lists or arrays of documents are possible with MongoDB.

MongoDB is a document database. It can store different types of data such as array, number, string, or subdocument with unique various storage engines in a single deployment. This is helpful for moving data between storage engine technologies. It is done using local replication. Figure 3.8 shows MongoDB's adaptive storage architecture. MongoDB 3.2 has four production storage engines, as shown in Figure 3.8, all of which can be used in a single replica set. The WiredTiger engine in MongoDB has concurrency monitoring and native compression features, and it has the best storage and performance efficiency. With the default, the WiredTiger storage engine can achieve concurrency control and native compression with optimal storage and performance efficiency. MongoDB allows two mixes of in-memory motors for ultra-low latency operations with a disk-based motor for altogether existence. It allows the assembly of large-scale, easy-to-access, robust frameworks and it authorizes various sensors and applications to store their data in a mode-adaptable manner. In the alter table command of RDBMSs, there are no database blockages; this command is used to change the mode. However, in uncommon states, such as the write-intensive scenarios in the master-slave model of MongoDB, there may be a blockage at the document level or a bottleneck in the system if sharding is not used. This is a kind of database partitioning that divides very massive databases into smaller, faster, and more easily managed sections called shards. MongoDB empowers horizontal scalability because table joins are not essential as they are in conventional RDBMS.

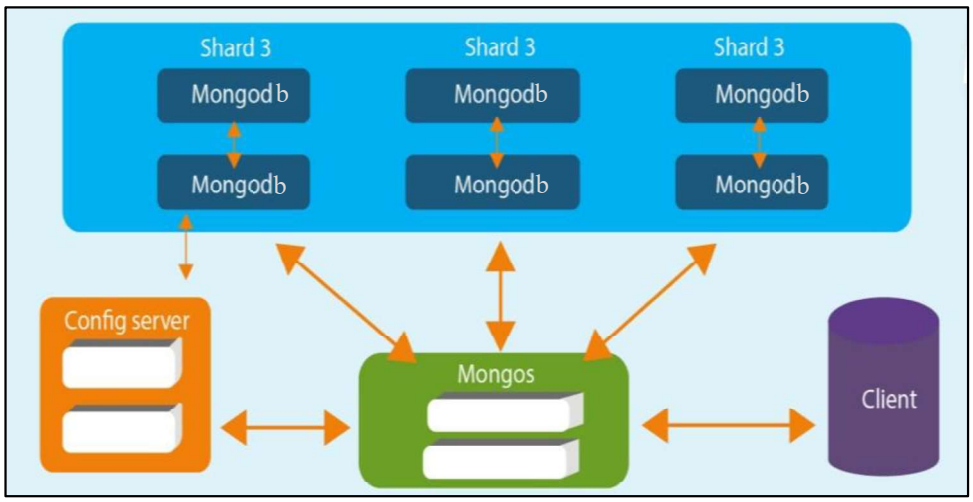


Figure 3.8 Architecture of MongoDB

MongoDB has automatic sharing properties in which replica server nodes are added to the system. It is a very fast database that not only provides indexing for primary attributes but also provides auxiliary attributes. This feature is available even in subdocuments and using aggregation frameworks, Hadoop systems, and MapReduce to compare various collections [71, 72].

- **How does MongoDB store data?**

In MongoDB, the data are stored as a document. These documents are saved in MongoDB in JavaScript object notation (JSON) format. JSON documents support set fields, so associated data and data lists can be saved with documents rather than external tables. It is considered a table similar to RDBMSs. It can store many documents that do not share the same structure. This is because MongoDB is a database without architecture. It has a dynamic schema data structure, which means that whenever data are used for storage purposes, it only builds the schema, and so we can say that the data structure will be stored [73, 74].

- **MongoDB drivers and client libraries**

Although some drivers use the C extension for better performance, the drivers and client libraries are usually written in their respective languages. MongoDB supports many programming languages, including C, C++, C#, Go, Erlang, Java, JavaScript, Node.js, Perl, PHP, Python, and Ruby [75].

- **MongoDB cluster installation and configuration**

The server to be installed should be configured according to the appropriate MongoDB installation file, and any other dependencies can be installed. See Appendix B.

3.2.1.3 Comparison of MongoDB and Riak KV

To better understand the differences between Riak KV and MongoDB, some characteristics of those NoSQL databases including replication, development language, data storage, storage type, and usage must be evaluated [67, 71]. All of these features are shown in Table 3.1.

Table 3.1 Riak KV and MongoDB features

	Riak KV	**MongoDB**
Description	Distributed, fault-tolerant key-value store	One of the most popular document stores
Development language	Erlang	C++
Storage type	Key-value store	Document store
Protocol	TCP/IP	TCP/IP
Initial release	2009	2009
Website	basho.com/products/riak-kv	www.mongodb.com
Operating systems	Linux OS X	Linux / Mac OS / Windows / Solaris
APIs and other access methods	HTTP API Native Erlang Interface	Proprietary protocol using JSON
Replication methods	Multi-node cluster	Master-slave replication
Server-side scripts	JavaScript and Erlang	JavaScript
Data storage	Disc	Disc

3.2.2 Cloud Computing Platform

Three different CC platforms have been used, and their characteristics, compositions, and structures are described in this section in detail.

3.2.1.1 DigitalOcean

DigitalOcean is a CC vendor that provides an IaaS platform for software developers. DigitalOcean is very popular with open-source companies and developers with AWS and GCP. To deploy DigitalOcean's IaaS environment, developers launch a private VM instance, which DigitalOcean calls a "droplet." Developers choose the droplet's size, which geographical region and data center it will run in, and which Linux operating system it will use: CoreOS, CentOS, Debian, FreeBSD, Ubuntu, or Fedora. Secure Shell (SSH) is likewise supported for reliable communication. In addition to choosing a Linux distribution, developers can also generate droplets from an existing VM image that comes with a pre-installed application (DigitalOcean calls this a "one-

click application"). DigitalOcean allows nine droplet sizes. The minimum size starts at 20 GB of solid-state drive (SSD) storage and 512 MB of RAM with one CPU. The largest droplet size is 640 GB of SSD storage and 64 GB of RAM with 20 CPUs. Developers can choose to resize the droplet after it is created. Developers use DigitalOcean to monitor and manage their droplets through control panels and open-source APIs. The dashboard allows developers to rebuild and scale droplets based on changes in workload and redirect network traffic and backups between droplets. A feature called "team account" can be used to establish resource sharing among different DigitalOcean users [76].

3.2.1.2 OpenStack

OpenStack is cloud software that can control a large number of computing, network, and storage resources. It also enables users to provide on-demand resources. Created in 2010, developed by Rackspace NASA and Hosting, OpenStack is designed to provide an open-source cloud solution for building public or private clouds. The goal of OpenStack is to allow any company to make and deliver CC services running on standard hardware [77, 78]. The use of OpenStack for open-source solutions is centered around the following core principles:

- Open source: All code will be released under the Apache 2.0 license, making it free for the community to use.
- Open design: The development community holds a design summit every six months to gather and write specifications and requirements for future releases.
- Open development: A publicly available source code repository is kept throughout the development process.
- Open community: Vibrant development and healthy user agreements are produced by transparency and open processes.

The general architecture for OpenStack, like any cloud platform, involves infrastructure with standard hardware; it can include any physical devices such as a disk, server, or network device. To provide cloud services, OpenStack develops a virtualization layer that provides end users with an abstract view of the physical infrastructure. These virtualization layers are built from the various components

illustrated in Figure 3.9. OpenStack architecture consists of three main components: Storage (Swift), Compute (Nova), and Network (Quantum).

In addition to these three pillars, OpenStack has developed many other services, each designed to work together to provide a complete IaaS solution. The integration of these services is facilitated by a common application programming interface provided by each service [77, 78].

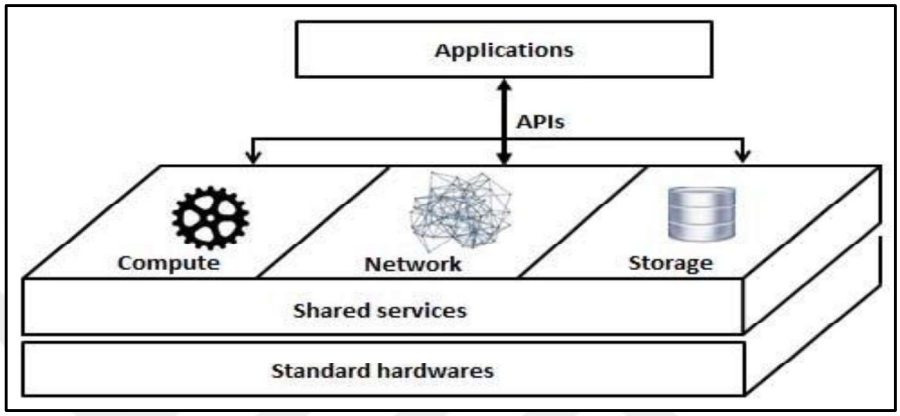


Figure 3.9 OpenStack general architecture [78]

3.2.1.3 Google Cloud Platform (GCP)

GCP, offered by Google, is a set of CC services that run on the same infrastructure that Google uses internally for its end-user products, such as YouTube and Google Search [80]. As a set of management tools, it gives a set of modular cloud services, including machine learning data analytics and computing data storage. GCP provides IaaS, PaaS, and server less computing environments [79, 80]. GCP is a CC service portfolio built around the original Google App Engine structure for hosting web applications from Google data centers. Since the beginning of the Google App Engine in 2008, it has grown into one of the significant CC platforms on the market, although GCP still lags behind AWS and Microsoft Azure in terms of market share. GCP is primarily a public cloud provider, although Google's use of Anthos has dramatically increased the focus on multi-cloud and hybrid workloads, allowing users to manage the Google Kubernetes Engine and GCP as well as workloads on Azure and AWS. Although cloud cost wars have faded over the past few years, Google has followed its own pricing

model and often claims to offer the cheapest of the top three providers. Nevertheless, Google does differentiate itself in terms of service [81].

Google Compute Engine (GCE), the main part of GCP, allows users to build and run VMs on their Google infrastructure. GCE allows performance, value, and scaling that makes it possible to start big compute clusters on Google's infrastructure quickly. Without up-front investment, users can run thousands of virtual CPUs on systems designed to deliver fast and consistent performance [82].

GCE provides a powerful computing infrastructure through which we can select and configure the platform components to use. With GCE, the user is responsible for configuring, managing, and monitoring systems or applications. Google guarantees that resources are reliable and available, but it is up to the user to configure and manage them. The benefit is that users have sufficient control over the system and infinite elasticity [83].

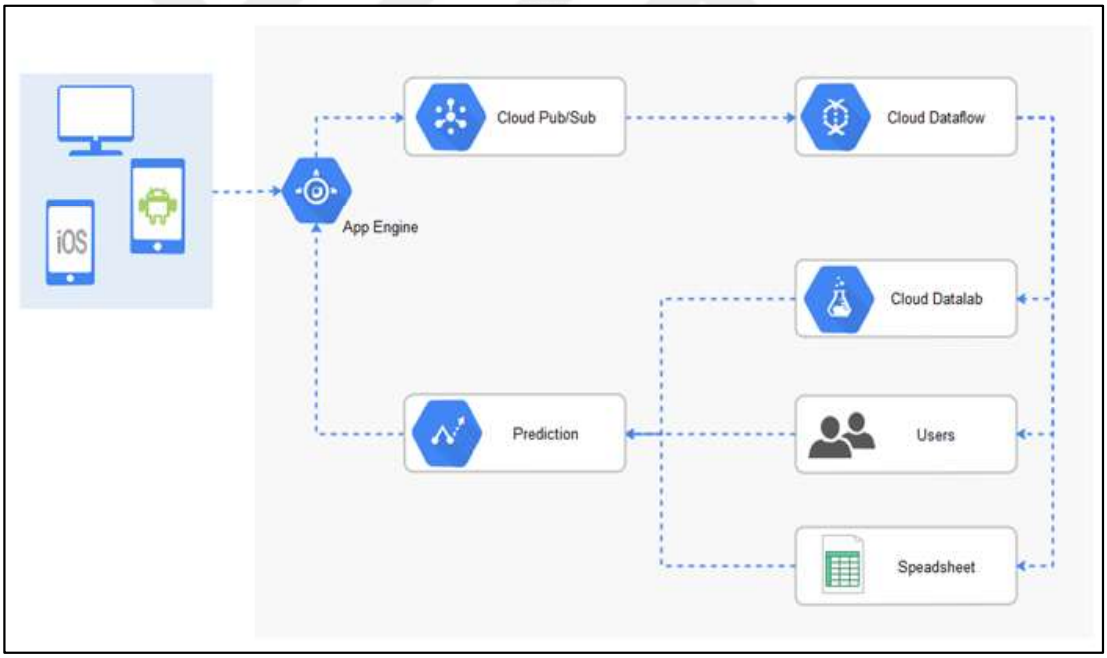


Figure 3.10 Google Cloud general architecture [83]

3.2.3 Benchmarking NoSQL systems

NoSQL is a more modern database category developed to meet the constraints of RDBMSs in facing big data needs. As a category, NoSQL describes a variety of

technologies that consider three aspects of big data: the exponentially increasing amount of data, the speed at which those data need to be processed, and the big changes in data being created by today's applications [84].

The NoSQL database performance behavior under various conditions is critical, as is the environment in which the database will run, and the best way to evaluate platforms. Using benchmarks (like parameters, expected data, and production configurations) and concurrent user workloads provides both businesses and IT great insight into platforms. In the following subsections, the benchmark tools that will be used in this research are introduced [85].

3.2.3.1 Basho-bench benchmark

Basho-bench is a benchmarking tool for accurate and repeatable performance and stress testing, and it generates performance graphs. Originally developed for testing Riak, it provides a pluggable driver interface and has been extended to benchmarking tools for a variety of projects. Basho-bench focuses on two indicators of performance: throughput and latency [86].

Every node can be a Riak node or a traffic generator. A traffic generator runs a copy of Basho-bench that generates and sends commands to Riak nodes. A Riak node holds an independent and complete copy of the Riak package, which is specified by an IP as a port number and address. Figure 3.11 shows how to organize Riak nodes and how traffic generators are organized within a cluster. There is one traffic generator for each three Riak nodes [87].

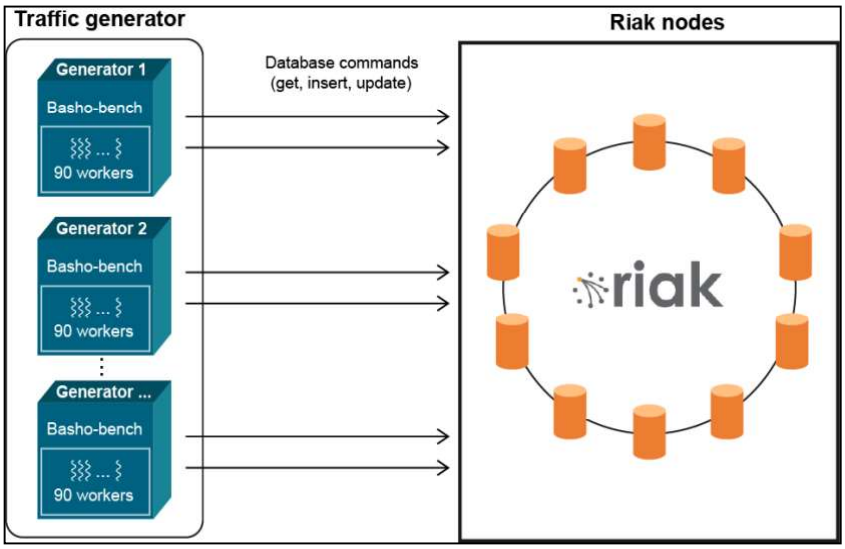


Figure 3.11 Riak nodes and traffic generators in Basho Bench

3.2.3.2 Yahoo Cloud Serving Benchmark (YCSB)

YCSB was designed and open-sourced by a group of Yahoo developers to establish benchmarking clients for various NoSQL data stores. The YCSB tool applies a vector-based method as one way of developing benchmarks to suitably react to an application's particular performance.

The YCSB tools are created for databases used in the cloud. These systems usually do not have a SQL interface. They only support a subset of relational operations, and their use cases are usually very different from traditional RDBMS applications and are not suitable for existing tools. The existing YCSB architecture used to benchmark such systems is shown in Figure 3.12.

The YCSB kernel package was created to evaluate various characteristics of system performance, depending on a set of workloads to evaluate a system's appropriateness for various workload characteristics at different points in the performance space [88, 89].

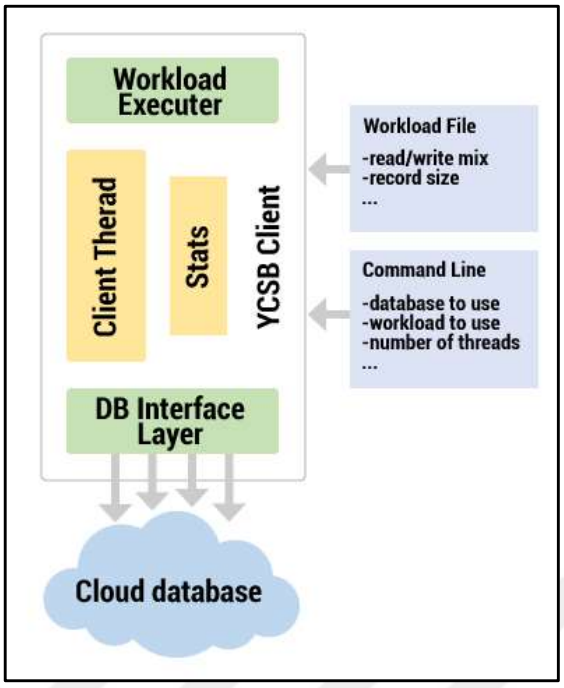


Figure 3.12 The architecture of the YCSB

The YCSB reporting framework reports the total throughput of operations per second. The measurement is calculated by dividing the total number of operations performed (writes plus reads) by the workload run time. Execution time is the time between the start of the first operation in the workload and the completion of the last process, excluding the initial setup and final cleanup time. This execution time is likewise reported individually as the overall run-time. Every workload execution produces a separate output file, including metadata and metrics. Metadata are recorded by the custom YCSB:

- Number of client threads.
- Database cluster configuration (IP addresses and number of server nodes).
- Workload configuration parameters for this execution.
- Command-line parameters used to recall this execution.
- Basic quality metrics (the number of operations, number of operations performed and completed successfully).
- The time of day of the start and end of this execution.

CHAPTER 4

EXPERIMENTAL ANALYSIS

This chapter presents the experiments and results of the research. Five different tests were conducted to evaluate the performance of the databases in a distributed environment. Figure 4.1 shows the architecture, and the two utilized clusters of the NoSQL database are an internal cluster and cloud cluster. The performance analysis and experimental results are demonstrated in this chapter. The experiments are divided into two sections:

Section 1: Internal cluster using five computers of the cluster in the local lab.

Section 2: Cloud cluster by using a VM in the CP. The performance analysis and experimental results are then demonstrated.

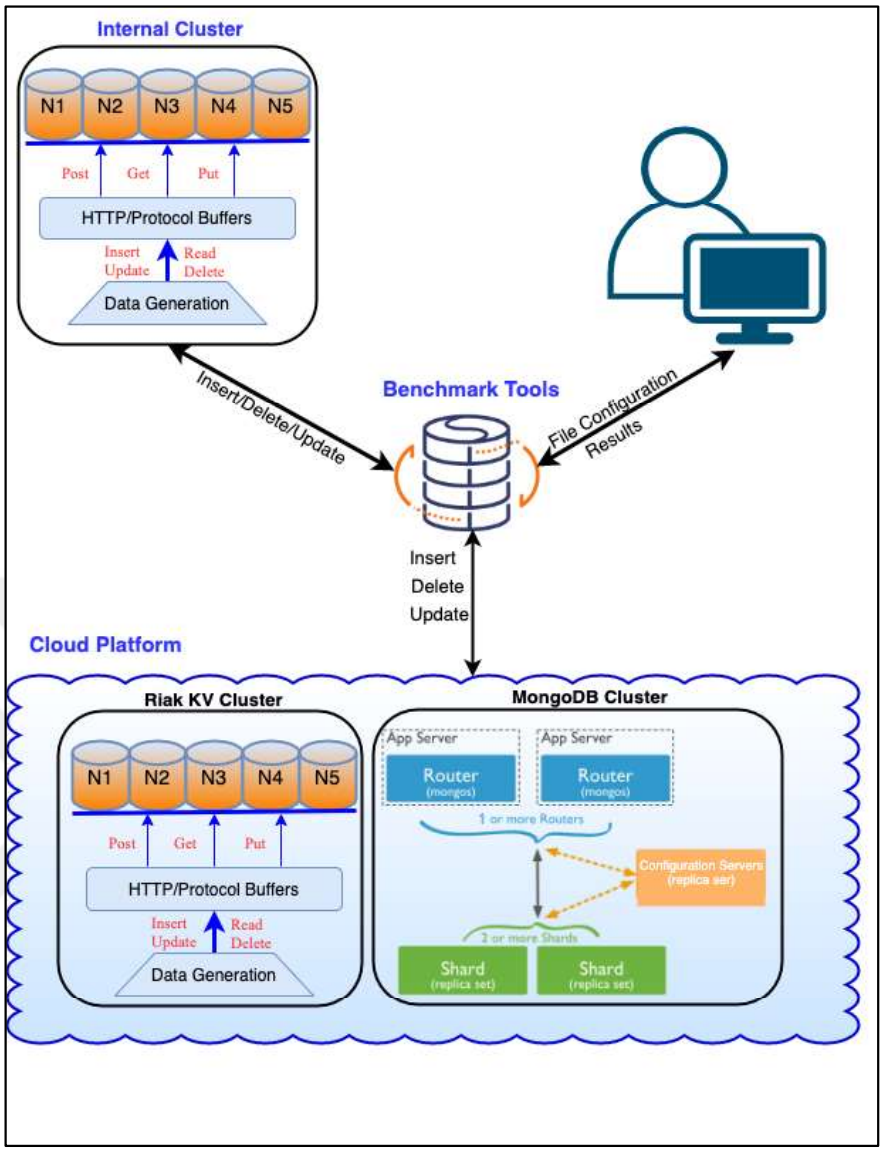


Figure 4.1 Performance analysis architecture

4.1 Section 1: Internal cluster

4.1.1 Experimental setup and dataset

The experiments were performed in an environment using five nodes of the cluster with 16 GB RAM, Intel-Xeon-CPU E3 1241 v3-@ 3.50 GHz × eight-processor speed, and one TB of ephemeral storage in each unit. Ubuntu 14.0.4 LTS (64-bit) was installed for each group. The test was done with a different number of keys (10K,

100K, 1,000K, 10,000K, and 200,000K) and a fixed size of 10,000 KB for every key. Table 4.1 illustrates the datasets used.

Table 4.1 The datasets used in the experiments of section 1

Record count	Record size (for each record)	Total size of the dataset
10K	1 KB	10 MB
100K	1 KB	100 MB
1,000K	1 KB	1 GB
10,000K	1 KB	10 GB
20,000K	1 KB	200 GB

4.1.2 Experiment 1: Evaluating the Riak KV cluster by YCSB

4.1.2.1 *Performance configuration and structure for experiment 1*

YCSB is a test tool to perform reads, updates, and writes based on workload and measure performance. The YCSB is commonly used for the evaluation and performance of NoSQL databases. The benchmark contains two parts: a data producer and a set of performance tests that evaluate, update, and read operations. Each test scenario is called a workload and is defined by a set of operations such as updating and reading some used records, and a total number of transactions. The benchmark tools give a set of predetermined workloads that can be executed as follows:

1. **Workload A:** Updates are heavy. It consists of a 50/50 proportion of reads and updates.
2. **Workload B:** Mostly reads. It consists of a 95/5 proportion of reads and updates.
3. **Workload C:** Reads only. The workload is 100% reads.

To evaluate the loading time, different numbers of records (10K, 100K, 1,000K, 10,000K, and 200,000K) and varying numbers of threads (1, 4, 8, and 12) were generated. Each experiment comprised 10,000 operations. Figure 4.2 illustrates the experimental structure, with details of the primary components.

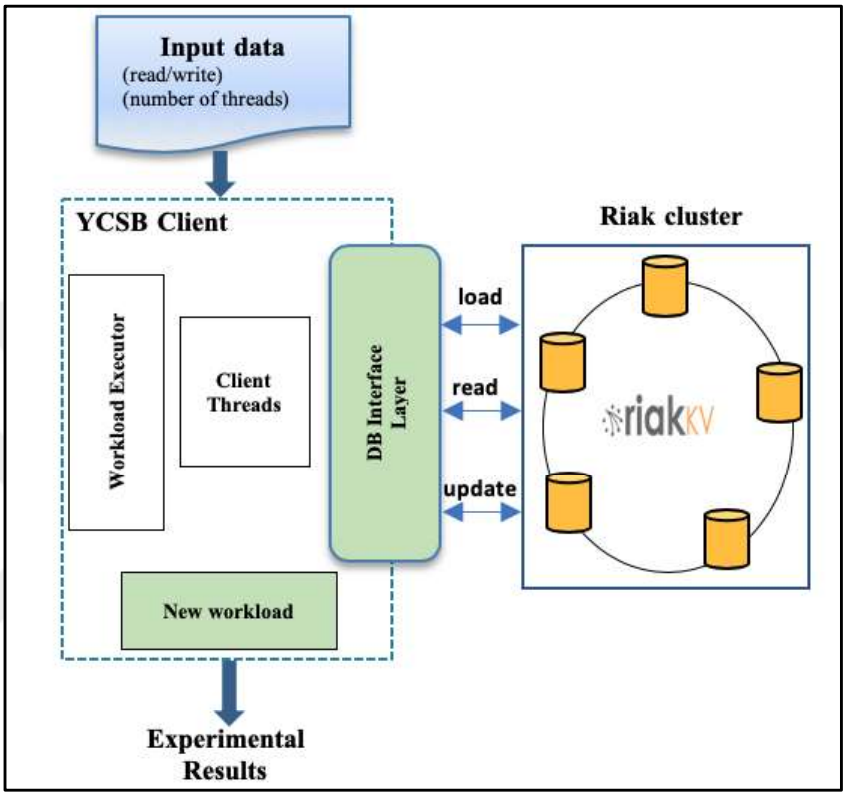


Figure 4.2 Experiment 1 in internal cluster

4.1.2.2 Experimental test and results

The default data size in the basic core differs from the same basic application type. With this benchmark, there is a record table with 10 fields for each record. Each record is identified by a primary key, such as the string "user412356." Each field is named field_0, field_1, and so on. The value of each field is a random ASCII string, with 1 KB records (10 fields, 100 bytes each, plus key).

1. **Workload (A):** Update-heavy workload. This workload has a mix of 50/50 reads and updates.

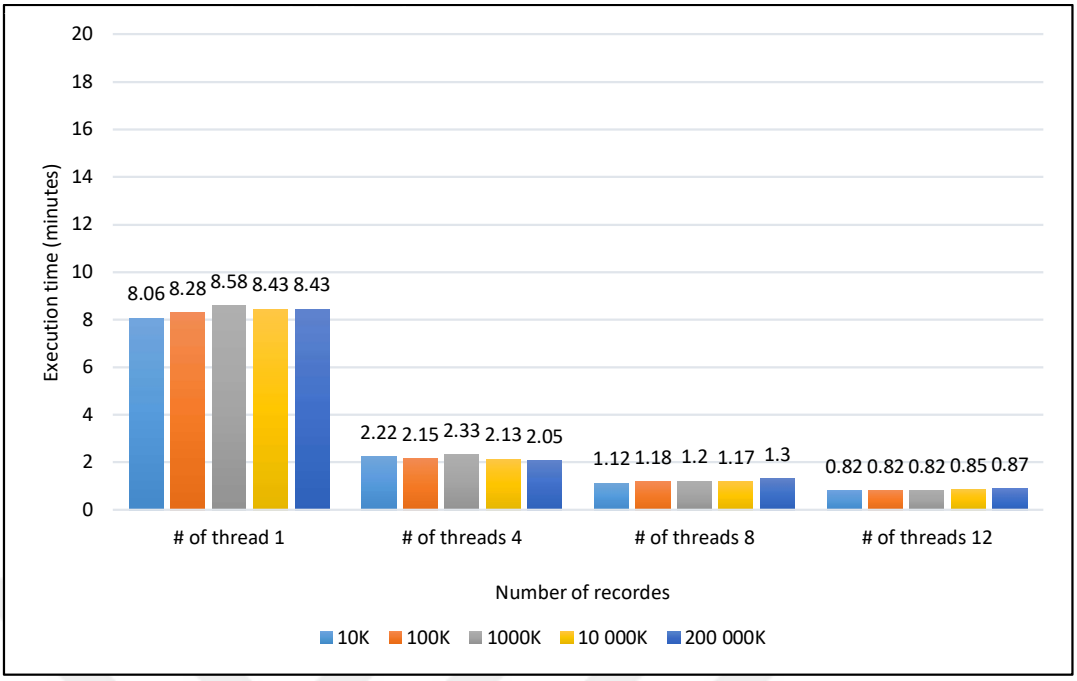


Figure 4.3 Execution time for workload A: 50/50 reads/updates

The results of the execution of the 50% read-50% update workload is shown in Figure 4.3. From the results, we observe that Riak KV generally has better throughput performance when we increase the number of threads. However, the execution time using a small number of records (for example, 10K) was very close to the execution time using a large number of records (200,000K), as shown in Figure 4.3. We notice from the graph for 1 thread that when the number of records in the cluster increased from 10K and 100K to 1,000K, the execution time was 8.06 minutes, 8.28 minutes, and 8.58 minutes, respectively. However, when the number of records is 10,000K and 200,000K, the execution time becomes 8.43 minutes.

Figure 4.4 illustrates a comparison of execution times, and it is seen that with 1 thread, the execution time increased by about 7 times compared to 8 and 12 threads. The case is similar for 4 threads, where the execution time decreases by almost 6 times.

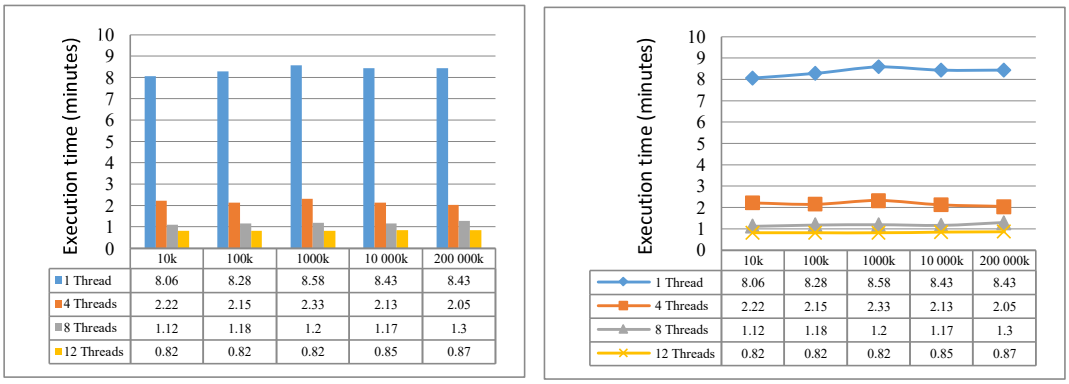


Figure 4.4 Comparison of execution times for workload A

The results for 12 threads differed from the other cases (1, 4, and 8). When increasing the number of records to 10K, 100K, 1,000K, 10,000K, and 200,000K, the execution time increased by 0.82, 0.82, 0.82, 0.85, and 0.87 minutes, respectively.

2. **Workload B:** 95% reads and 5% updates.

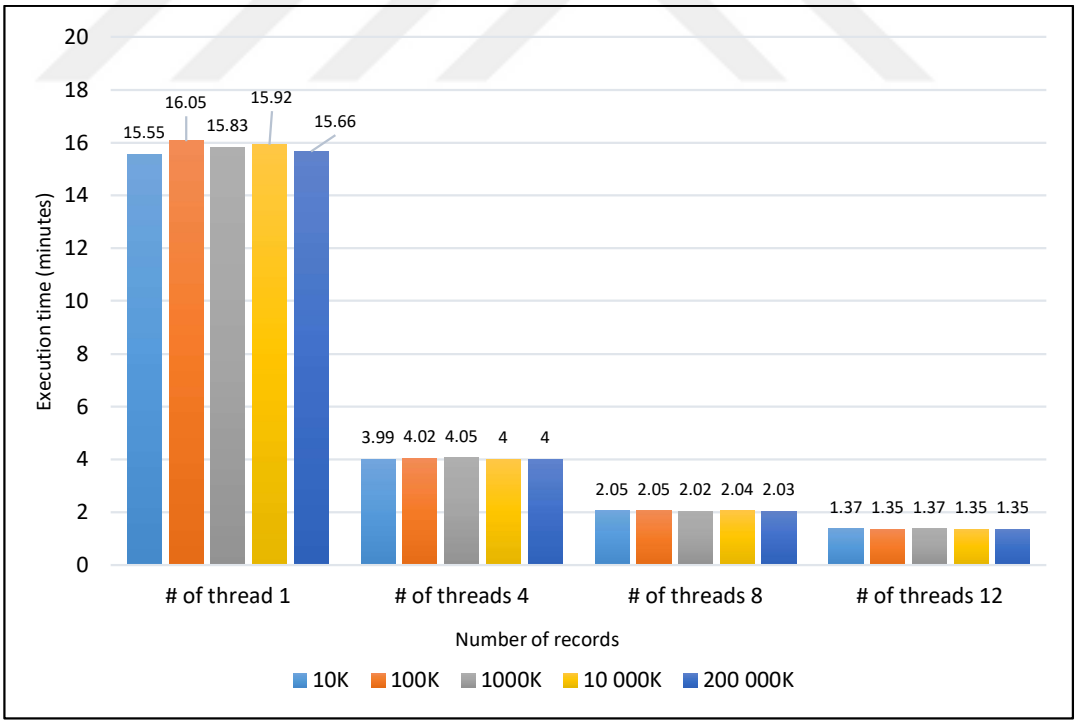


Figure 4.5. Execution time for workload B: 95/5 reads/updates

The performance behavior exhibited in workload A differed from the experiment conducted for workload B (95% reads, 5% updates). Moreover, the execution time of workload B was higher than the execution time of workload A with all numbers of threads. Furthermore, the execution time decreased with increasing numbers of threads; for example, for 10,000K records with 1, 4, 8, and 12 threads, the difference in execution time was very large (15.92, 4.00, 2.04, and 1.35 minutes, respectively). As can be seen in the figure above, the number of records has no significant effect on the performance of Riak KV. For example, in Figure 4.5, the execution times using 10K records and 200,000K are 15.55 minutes and 15.66 minutes, respectively, which is a very small difference.

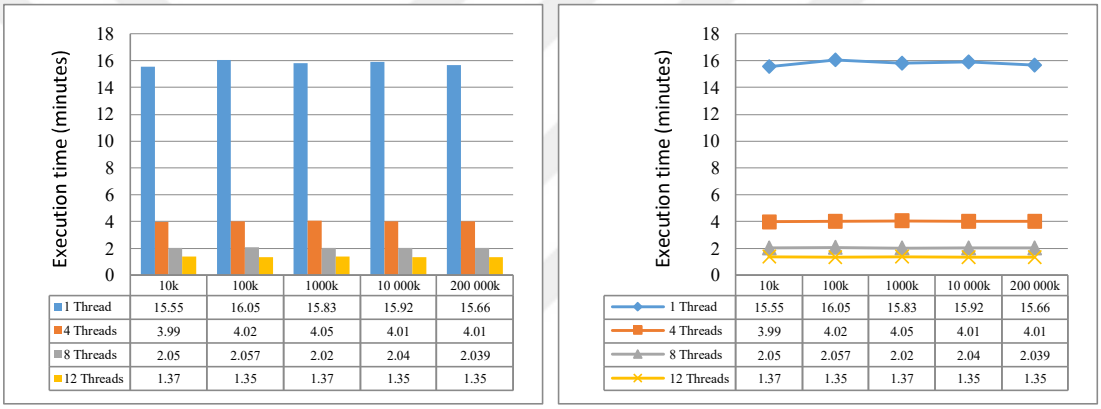


Figure 4.6 Comparison of execution times for workload B

Similar to the previous results obtained with 8 and 12 threads, the execution time using a smaller number of records (10K, 100K) was very close to the execution time using a large number of records (200,000K), as shown in Figure 4.6. On the other hand, the results shown for 1 thread reveal that the execution time increased about 7 times compared to 4, 8, and 12 threads. This is similar to the behavior noted for the execution of workload A.

3. **Workload C:** Read-only, with workload read ratio of 100%.

The results of the execution of workload C with 100% reads are shown in Figure 4.7.

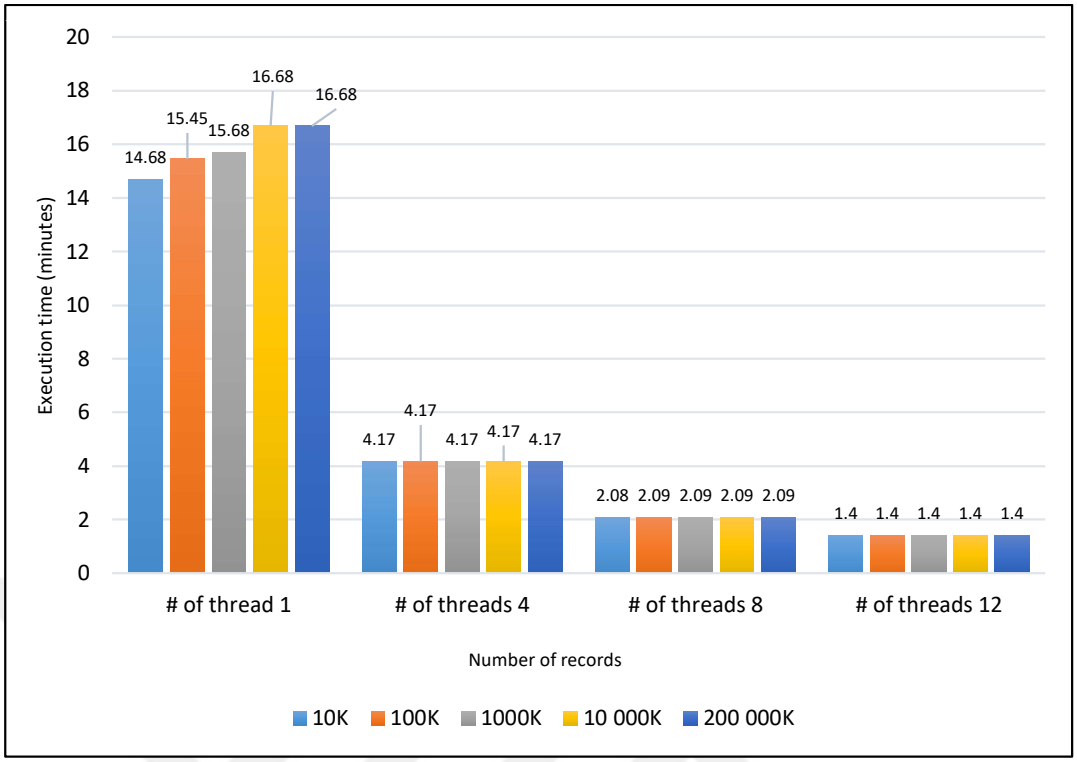


Figure 4.7 Execution times for workload C: 100% reads

In tests of workload C, the increase of reading records increased the execution time only slightly, which confirms again that the number of records has no significant effect. For example, Figure 4.7 shows the execution time when using 100K records as being 2.09 minutes, and when the number of records reaches 200,000K, the execution time is still the same (2.09 minutes). Overall, the performance is worse than the execution time of workload A.

This workload is 100% reads and the Riak KV is not enhanced for performing reads compared to updates. As the number of reads performed by retrieving a disk increases, the performance is degraded.

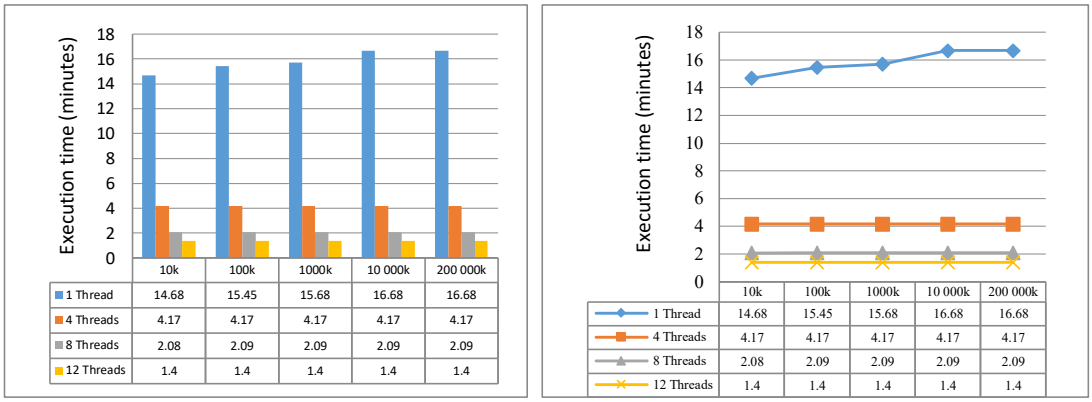


Figure 4.8 Comparisons of execution times for workload C

Figure 4.8 shows the execution time for 100K and 200,000K records as being 4.17 minutes and 1.40 minutes, respectively. The end result shows that as threads increase, execution time is decreasing regardless of workload type.

As the number of threads increases, this advantage still exists. The system takes advantage of this opportunity to effectively handle more threads.

4.1.3 Experiment 2: Evaluating Riak KV Cluster with Basho-bench

4.1.3.1 Performance configuration and structure for experiment 2

Basho-bench is a test tool to perform reads, updates, and writes based on workload and measure performance. The permissible operations that a driver might run are in the format of [{put,5},{get,3},{delete,1}], meaning that put will be called five times in every 9 operations, get will be called three times, and delete will be called once, on average. The benchmark package gives a predetermined set that can be executed as follows:

1. Workload A: Consists of a 1/1 proportion of reads and updates.
2. Workload B: Mostly reads. It consists of a 9/1 proportion of reads/updates.
3. Workload C: Reads only. The workload is 100% reads.

To evaluate the loading time, we generate a different number of keys (10K, 100K, 1 million, 10 million, and 200 million) and varying numbers of threads (4, 8, and 12). The benchmark requires a configuration file that contains the parameters needed to

start the benchmark. For example, for the next file configuration for part of workload A, the number of records is 10K and the number of threads is 8.

```
{mode, max}.
{duration,10}. % minutes
{concurrent, 8}. %% number of threads
{code_paths, [”/n0/aimen/basho bench/deps/stats”]}.
{key generator , {int to bin , {uniform int , 10000}}}. %%number of  key 10 K
{value generator , {fixed bin , 1000}}.  %% size of key 1KB
{riakc pb ips ,
[{192,168,10,1},{192,168,10,2},[{192,168,10,3},[{192,168,10,4},[{192,168,10,5}]}.%% %
Connect to a cluster of 5 machines
{http_raw_port, 8098}.
{http_raw_path, "/basho"}.
{operations , [{get , 4}, {update, 4}]}. %% read:50%, update:50%
```

Figure 4.9 illustrates the experimental structure with details of the primary components.

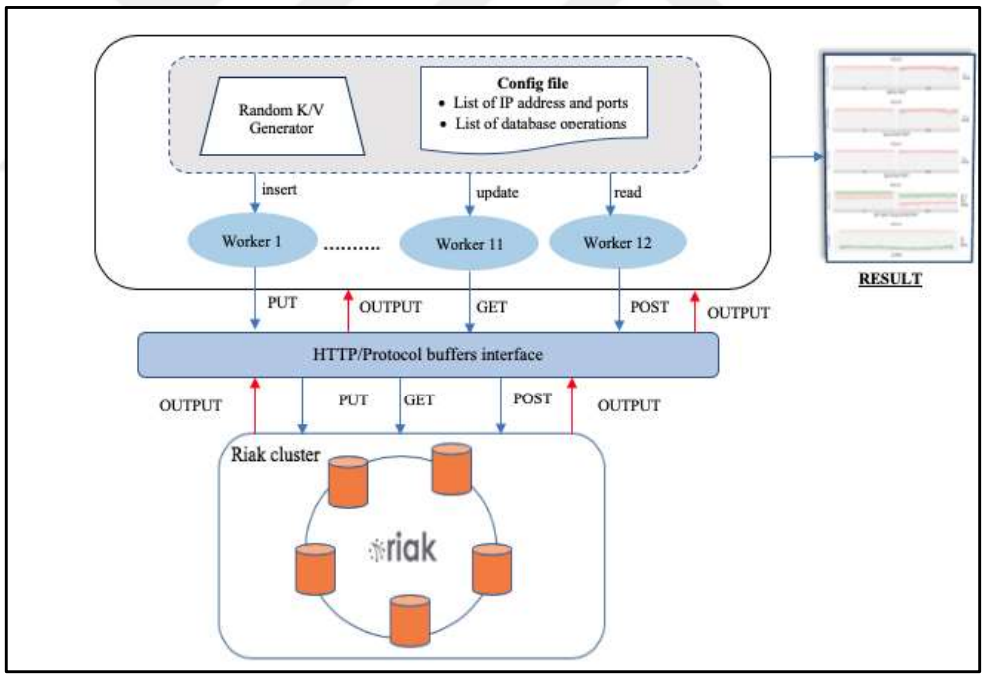


Figure 4.9 Experiment 2 in internal cluster

4.1.3.2 Experimental test and results

Each of the following subsections addresses one of the 3 experiments, describing the different experimental scenarios with reads and updates, and also their results.

1. **Workload A:** This workload has a 1/1 read-update mix. Figure 4.10 shows the results.
    - *Throughput results*

We notice from Figure 4.10 that with 8 threads, when the number of keys in the cluster is increased from 100K to 1 million, the throughput performance is similar (190 operations). However, when the number of keys is 10K, the performance is higher (250 operations). For the overall case of 12 threads, the performance is very high for all records compared to other numbers of threads.

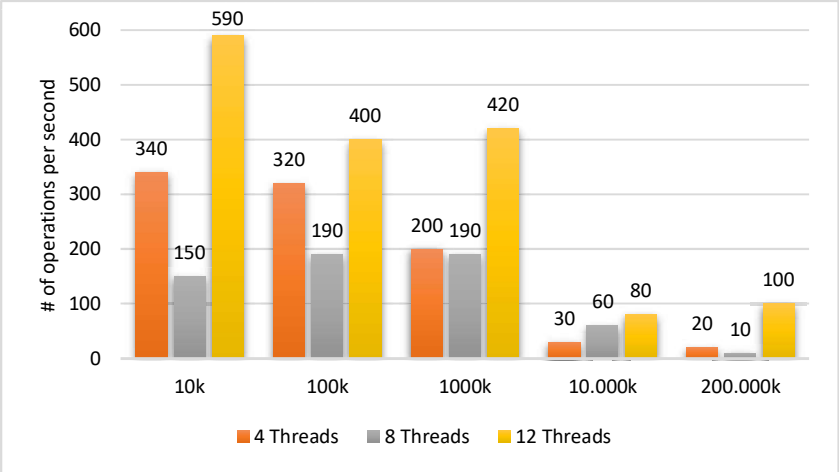


Figure 4.10 Throughput performance for workload A

- *Latency results*

Latency is the delay from the input into the system until the desired result; in each case, the term is understood slightly differently, and the latency problem varies from one system to another. Latency greatly impacts how usable and enjoyable mechanical and electronic equipment and communications are.

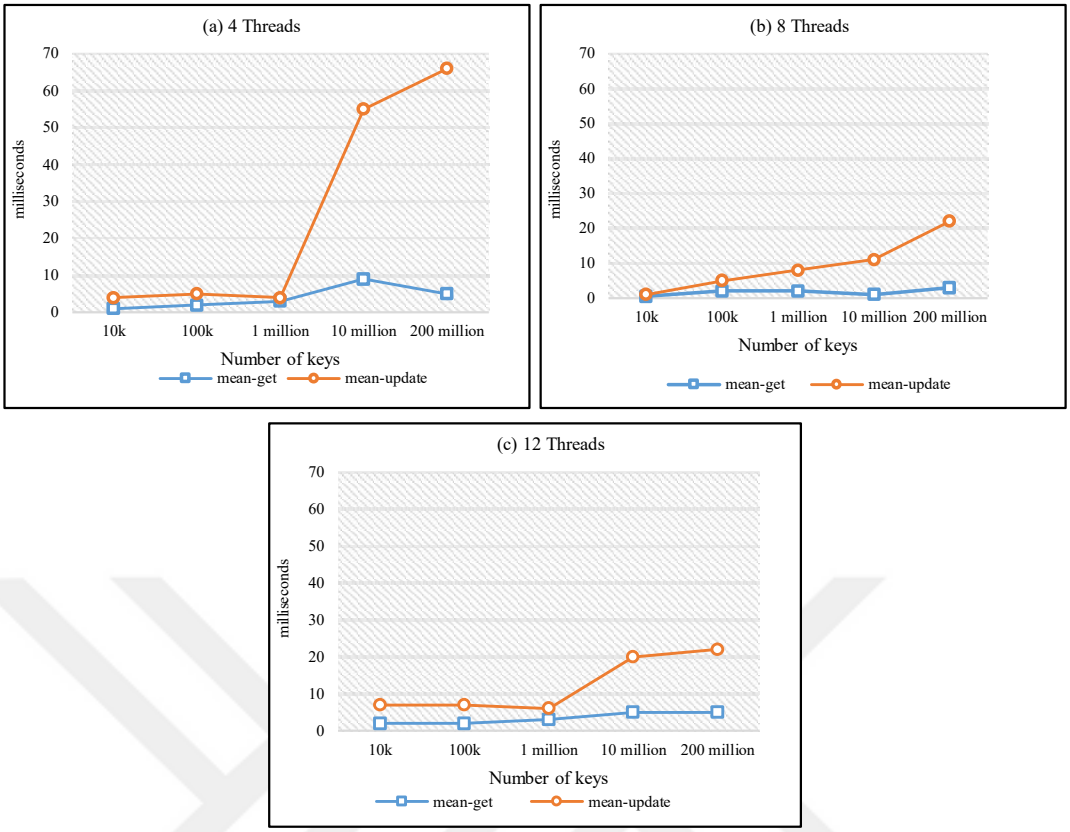


Figure 4.11 Latency for workload A

From Figures 4.11a-4.11c, it can be observed that the three cases have high latency in the process of data updates. This is expected because the reading process does not usually have great latency like the rest of the operations. For the highest value of 4 threads, the latency rate reached 66 milliseconds and was almost equal to the values for 8 and 12 threads.

2. **Workload B:** In the second experiment, updates are heavy. Workload B consists of a 9/1 proportion of reads and updates. Figure 4.12 shows the results.
    - *Throughput results*

The experimental results are illustrated in Figure 4.12.

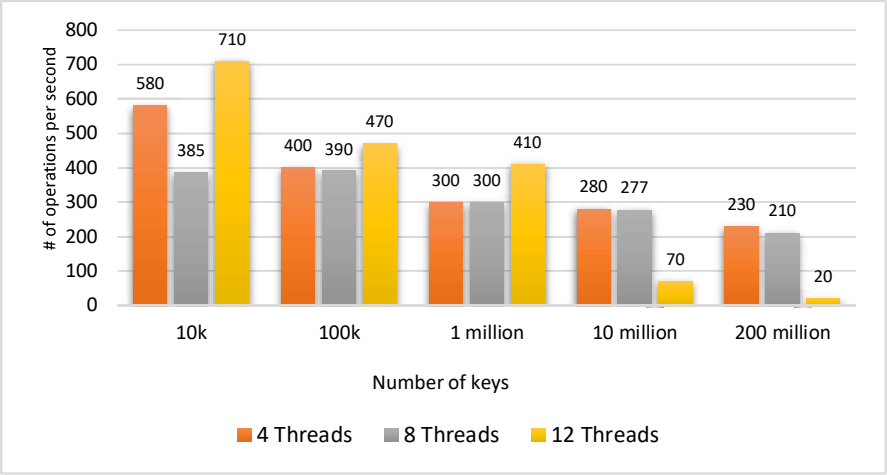


Figure 4.12 Throughput performance for workload B

The performance behavior exhibited in experiment A differs from that in experiment B (9 read operations, 1 update operation). Moreover, the throughput performance of experiment B is higher than the throughput performance of experiment A for all threads. Furthermore, the performance decreases when we increase the number of threads. For example, for a number of keys from 10K to 200 million with 4 and 8 threads, the difference in the number of operations was not expected.

As can be seen from the figure, the number of keys has a significant effect on the performance of Riak KV. For example, the number of operations using 10K and 200 million keys with 12 threads is 710 operations/second and 20 operations/second, respectively, which is very large.

- *Latency results*

From Figure 4.13, it can be seen that the results here are different from those of workload C. The latency is high with 4 threads, where the number of records reaches 10 million in about 44 milliseconds, and it is 70 milliseconds with 200 million keys.

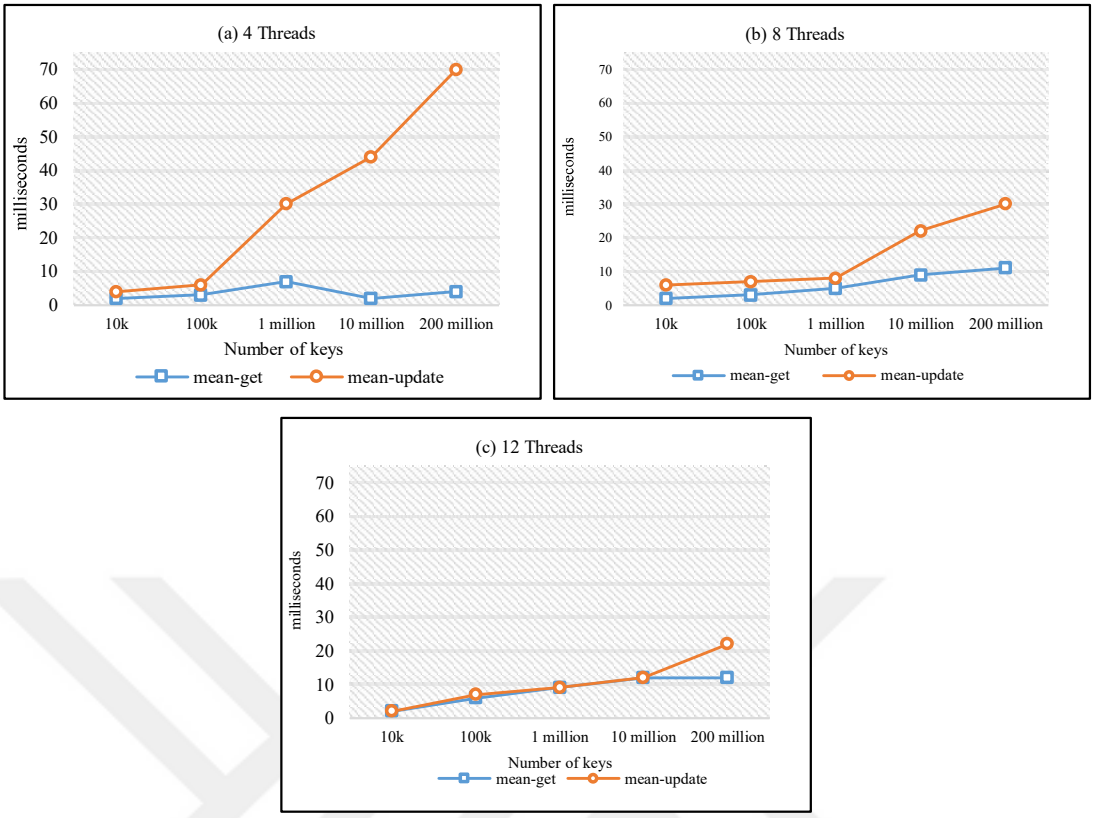


Figure 4.13 Latency for workload B

3. **Workload C:** Workload C tests read-only operation. For this experiment, the read ratio is 100% and the results are shown in Figure 4.14.

- *Throughput results*

The throughput performance of workload C with 100% reads is shown in Figure 4.14. The increase of read keys decreases the throughput, which confirms again that the number of keys has a significant effect. For example, the figure shows the number of operations using 100K keys as being 625 operations/second, and when the number of keys reaches 200 million, the number of operations decreases to 475 operations/second. Figure 4.14 shows the number of operations for 200 million keys as 320 operations/second, thus making it less efficient for 8 threads, with lower throughput performance compared to other numbers of threads. In general, for the read-only experiments, the performance was high and stable for all numbers of keys as compared to workloads A and B.

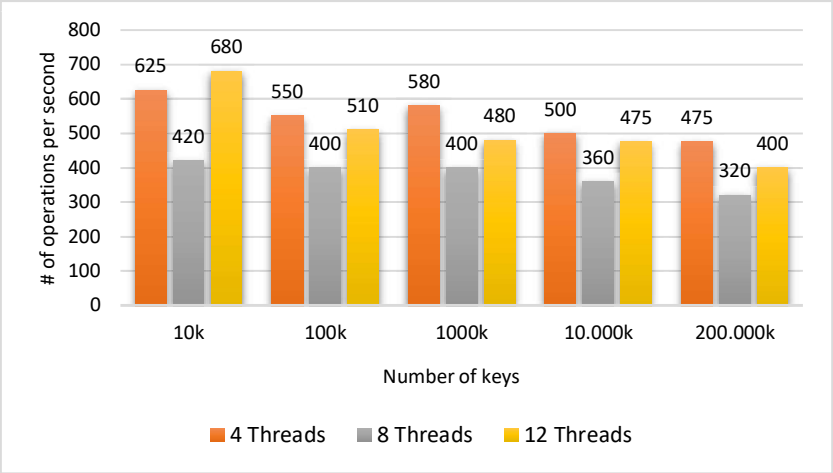


Figure 4.14 Throughput performance for workload C

- *Latency results*

Figure 4.15 shows the latency for workload C, with read operations only. Note that when using 4 threads, the latency is high, where the highest value was 12 milliseconds with 200 million keys. The results show that the latency was almost equal for all numbers of keys, from 10K to 200 million, while the threads were performing read-only operations.

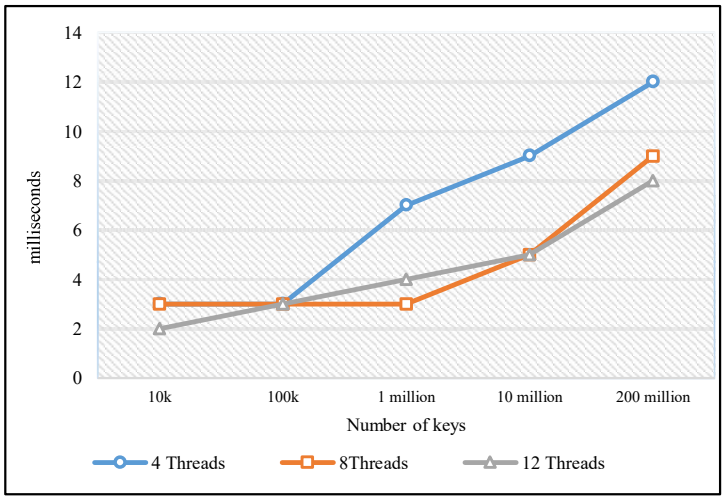


Figure 4.15 Latency for workload C

4.2 Section 2: Cloud Cluster Environment

4.2.1 Experimental setup and dataset

Three different cloud computing environments are used in this section. For all experimental analysis, YCSB is used as one of the most commonly used benchmarks to test NoSQL databases. YCSB has a client that consists of two parts: a workload generator and a set of scenarios. Those scenarios, known as workloads, are combinations of read, write, and update operations performed on randomly chosen records. The predefined workloads are as follows:

1. **Workload A:** Update-heavy workload. This workload has a mix of 50/50 reads and updates. This workload thus entails reads and updates equally. A typical real-life representative of this is an online shopping cart: people review their carts and change their choices.
2. **Workload B:** A workload with mostly reads. It has a 95/5 reads/update mix. This workload represents real-life applications like reading on a social website mixed with occasional commenting.
3. **Workload C:** Read-only. This workload is 100% reads.
4. **Workload D:** Read the latest. This workload consists of 95/5 reads/inserts. In this workload, new records are inserted, and these are treated as the most popular ones.
5. **Workload E:** Short ranges. It has a ratio of 95/5 scans/inserts. In this workload, short ranges of records are queried instead of individual records. This workload retrieves records satisfying certain conditions. A typical application scenario is that people view news retrieved based on recommended trends or topics on social media websites.
6. **Workload F:** In this workload, the read/read-modify-write ratio is 50/50; the client will read a record and modify it. An example of typical real-life usage is that people often review and change their profiles on websites.

The default data size in the basic core differs from the same primary application type. With this benchmark, 100K, 5,000K, and 20,000K records were generated. There is a record table with ten fields for each record. Each record is identified by a primary key, such as a string like “user412356.” Every field is named field_0, field_1, and so on.

The value of each field is a random ASCII string, with 1 KB records (10 fields, 100 bytes each, plus key), and a varying number of up to 12 threads. Each experiment comprised 10,000 operations. Table 4.2 shows the dataset used.

Table 4.2 The datasets used in the experiments on the cloud cluster (section 2)

Record count	Record size (for each record)	Total size of the dataset
10K	1 KB	10 MB
5,000K	1 KB	5 GB
20,000K	1 KB	20 GB

All the tests were executed in a CC environment with five VMs. Each operation against the data store is randomly chosen to be one of the following:

- Insert: Insert is used for a new record.
- Update: Update a record by changing the value of one field.
- Read: Read a record, and one randomly selected field or all fields.
- Scan: Scan records in order, starting at a randomly chosen record key. The number of records to scan is randomly chosen.

Especially for scanning, the distribution of the scanning length is selected as part of the workload. Therefore, the scan method takes the first key and the number of records to be scanned. Of course, a real application may instead specify a scan interval (i.e., from February 1st to February 15th). The number of record parameters allows us to control the size of these intervals without having to define and specify meaningful endpoints for the scan.

4.2.2 Experiment 1: Evaluating and testing in DigitalOcean

4.2.2.1 Performance configuration and structure for experiment 1

All the tests were executed in the DigitalOcean CC environment with five VM "droplets" and OS Ubuntu 14.04.5 x64 with 4 GB RAM available, size two vCPUs/80 GB. The tested versions of the NoSQL databases are MongoDB version 4.0 and Riak KV version 2.2.3. YCSB-0.15.0 is also used. Figure 4.16 illustrates the experimental structure with details of the primary components.

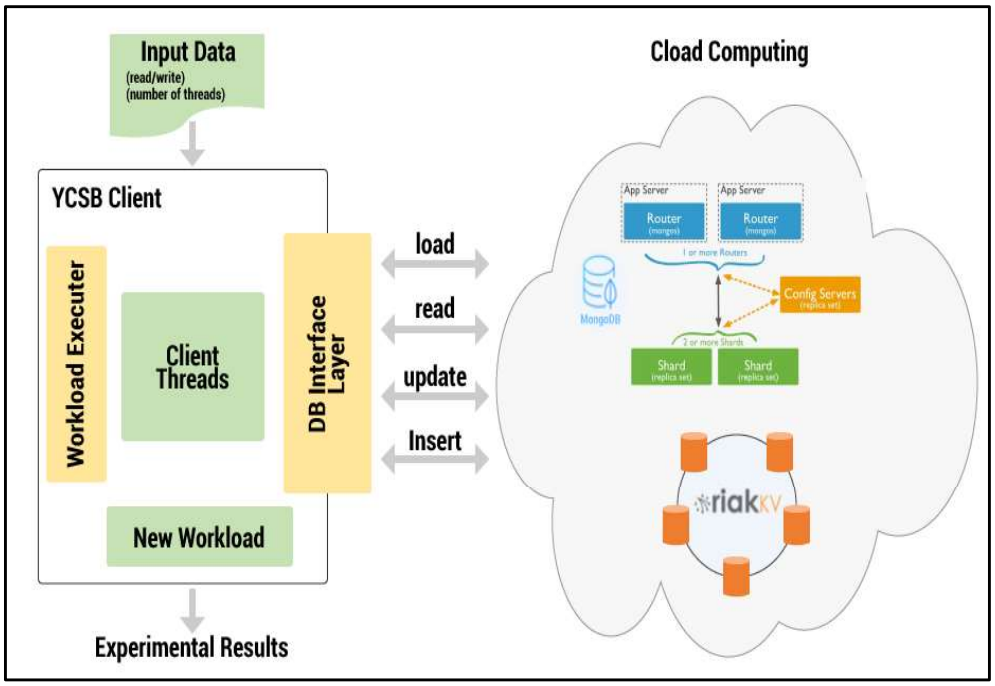


Figure 4.16 Experiment 1's structure in cloud cluster

4.2.2.2 Experimental test and results

The following subsection presents and analyzes the results of loading 100K, 5,000K, and 20,000K records generated with YCSB.

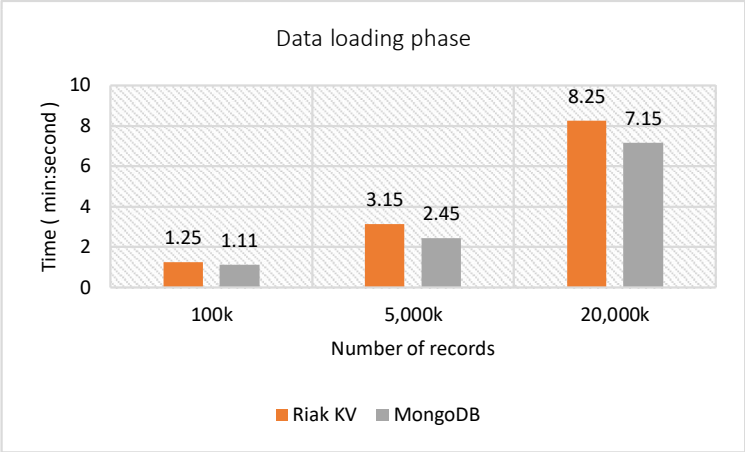


Figure 4.17 Experiment 1's data loading test in cloud cluster

Loading time (less is better): As shown in Figure 4.17, as the size of data increases, the runtime of loading data for Riak KV and MongoDB increases, but MongoDB has the lowest runtime. In the first test (100K records), MongoDB and Riak KV have almost identical loading times. As the number of records increases, the loading time increases until it reaches 8:25 for Riak KV and 7:15 for MongoDB with 200,000K records.

1. **Workload A:** 50/50 reads and updates.

As illustrated in Figure 4.18, good performance results are obtained by the document store databases. Compared to Riak KV, MongoDB had better throughput time regardless of database size. The performance of MongoDB was faster than that of Riak KV using a mix of 50/50 reads and updates with 100K, 5,000K, and 20,000K records. Another important point that can be observed is that throughput times are very close when the number of records increases from 5,000K up to 20,000K for Riak KV. From Figure 4.18, it can be noted that the three cases have high latency in the process of data updates. This latency is expected because the reading process usually does not have high latency like that of the update operations, where the highest value reached was a latency rate of 11.08 milliseconds with Riak KV and the latency of Riak KV was slightly higher than that of MongoDB for workload A.

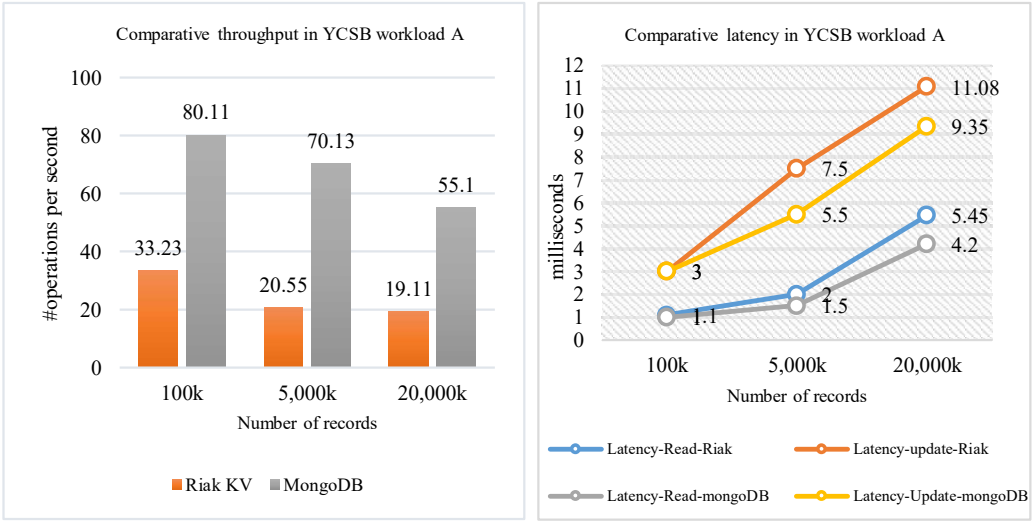


Figure 4.18 Experiment 1, workload A in cloud cluster

2. **Workload B:** 95/5 reads and updates.

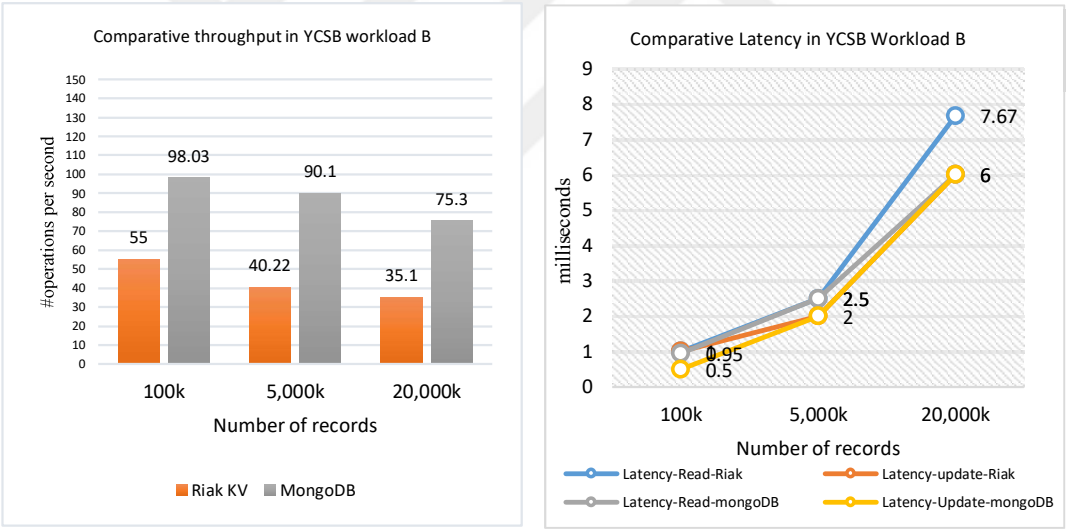


Figure 4.19 Experiment 1, workload B in cloud cluster

The throughput performance of workload B differed for Riak KV and MongoDB, as shown in Figure 4.19. The throughput time for MongoDB kept decreasing as the data volume became larger, and the highest throughput times for MongoDB were 98.03, 90.1, and 75.3 operations/second. When increasing the number of records to 10K, 5,000K, and 20,000K, respectively, and for small amounts of data (100K records), Riak KV had better results. Almost all latency results were close in time, except for

Riak KV when the number of records was 20,000K; the latency time was then relatively high at 7.67 milliseconds.

3. **Workload C:** 100% reads only.

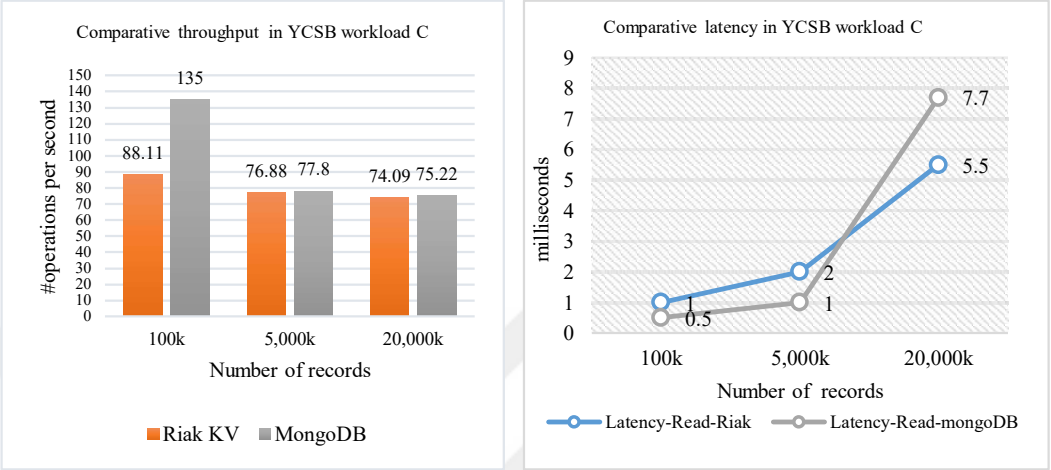


Figure 4.20 Experiment 1, workload C on cloud cluster

Workload C comprises 100% reads. As in the previous experiments, with large numbers of reading operations, MongoDB becomes more productive with a large volume of data, as illustrated in Figure 4.20. MongoDB showed similar behavior to that of the previous workload when the data size was 10 MB (10K), but there was a difference with the increase in the number of records (5,000K, 20,000K) as the throughput performance was closer to that of Riak KV: with 5,000K records, this value was 76.88 operations/second for Riak KV and 77.8 operations/second for MongoDB. With 20,000K records, these values were respectively 74.09 and 75.22 operations/second. The latency results show that MongoDB had the highest latency time of 7.7 milliseconds with 20,000K records.

2. **Workload D:** 95/5 reads and inserts.

Figure 4.21 shows the results obtained while executing workload D, a workload of 95% reads and 5% inserts. To test this workload, 10,000 operations were performed over databases with 100K, 5,000K, and 20,000K records stored.

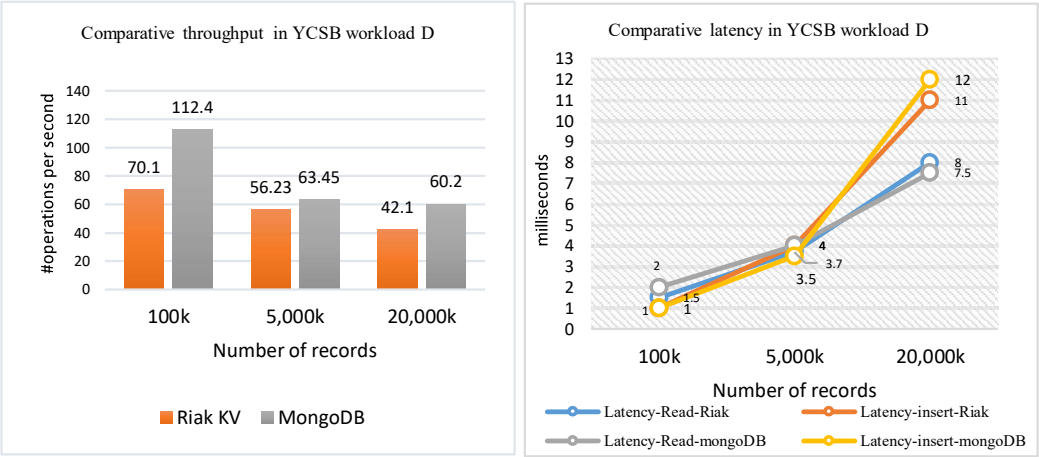


Figure 4.21 Experiment 1, workload D in cloud cluster

As illustrated in Figure 4.21, the results are indicative of the superiority of MongoDB over Riak KV for all dataset sizes. For each throughput time, Riak KV showed better results. With increased data size, both MongoDB and Riak KV started having lower throughput times; with 5,000K and 20,000K records, these values were 56.23 and 63.45 operations/second. Still, Riak KV was not even close to MongoDB at 20,000K records, with 42.1 operations/second for Riak KV and 60.2 operations/second for MongoDB. It had the highest performance when the number of records was 100K, reaching 112.4 operations/second.

3. **Workload E:** 95/5 scans and inserts.

With this workload, MongoDB performed better than Riak KV and throughput was generally low, as observed from Figure 4.22, due to the scanning process. The latency values were very close, with insignificant difference.

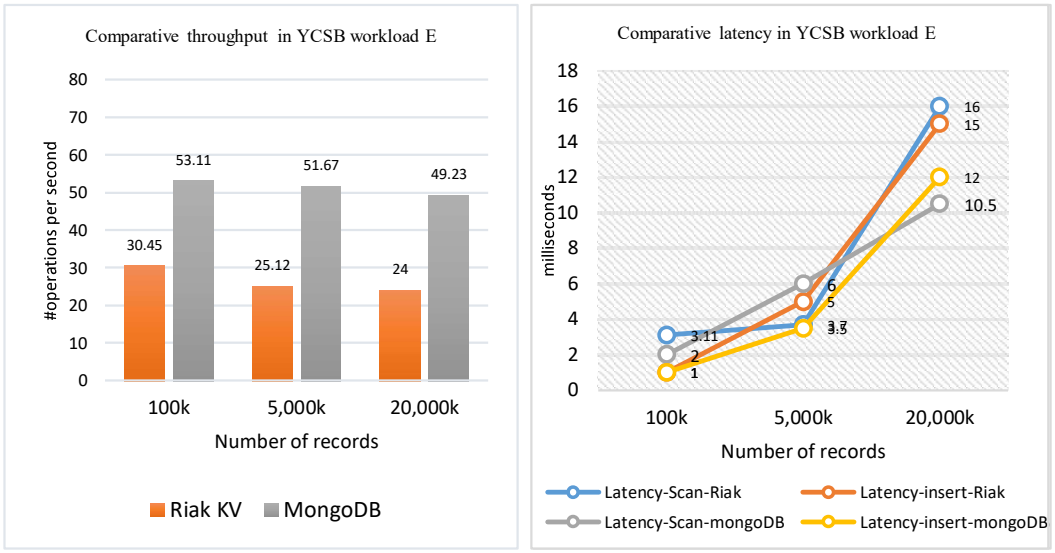


Figure 4.22 Experiment 1, workload E in cloud cluster

4. **Workload F:** 50/50 reads and read-modify-writes.

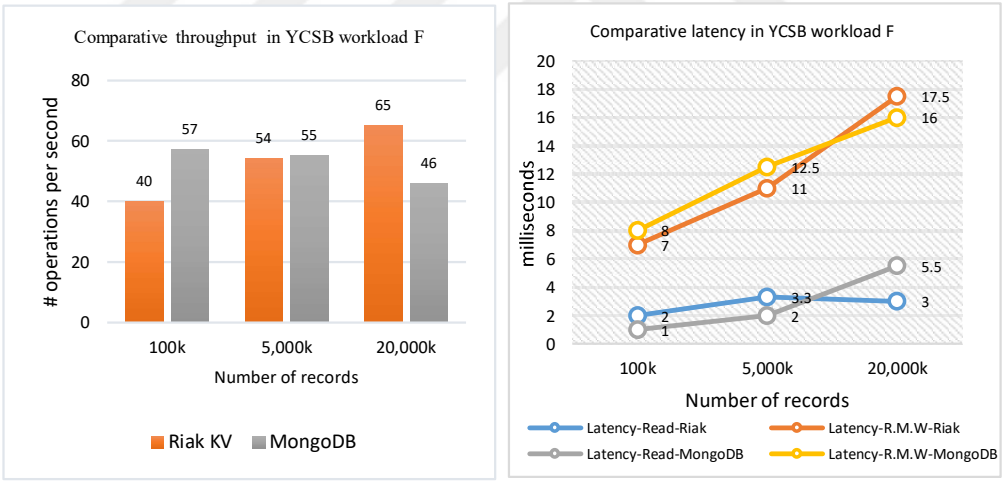


Figure 4.23 Experiment 1, workload F in cloud cluster

With this workload, the client will read a record, modify it, and write back the changes. MongoDB showed unexpectedly negative results, as illustrated in Figure 4.23. The highest throughput was obtained for Riak KV with the increase in data volume, increasing from 54 operations per second to 65 operations per second. Latency with the read-modify operation was very high, reaching 17.5 milliseconds with Riak KV and 16 milliseconds with MongoDB.

4.2.3 Experiment 2: Evaluating and testing in OpenStack

4.2.3.1 Performance configuration and structure for experiment 2

All the tests were executed in the OpenStack environment with 5 VM OS Ubuntu 14.04.5 x64 with 4 GB RAM available, size 4 vCPUs/40 GB. The tested versions of the NoSQL databases are MongoDB version 4.0, Riak KV version 2.2.3, and YCSB 0.15.0. Figure 4.24 illustrates the experimental structure with details of the primary components.

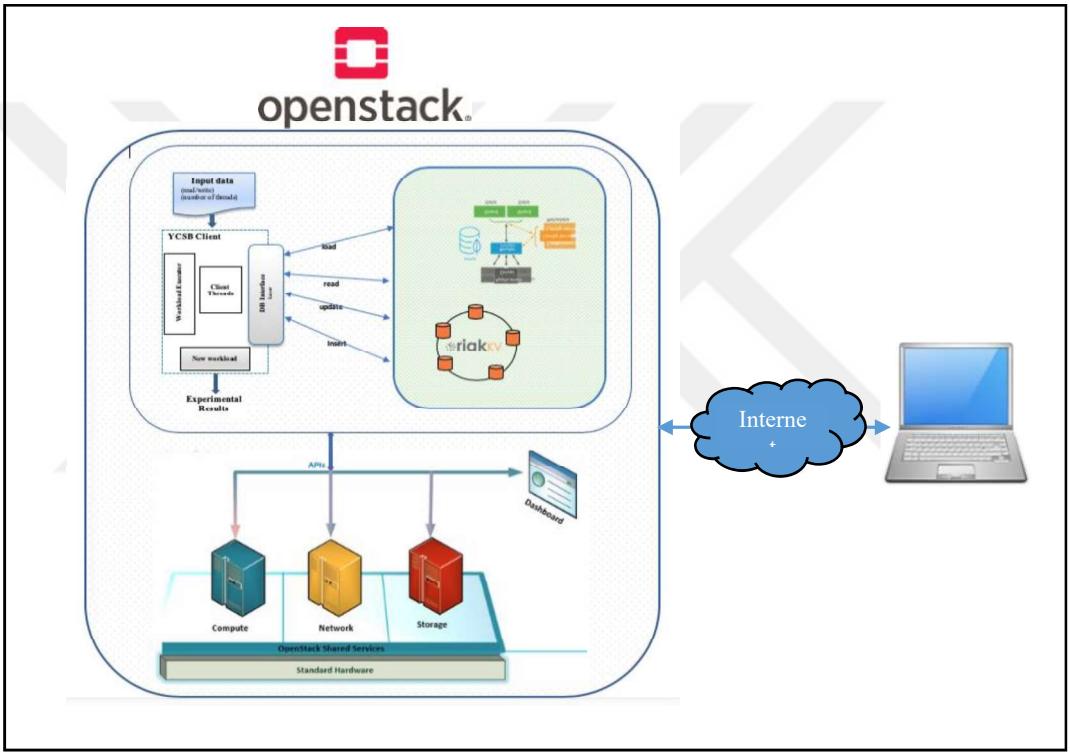


Figure 4.24 Experiment 2's structure in cloud cluster

4.2.3.2 Experimental test and results

The following section presents and analyzes the results of the YCSB-generated load for 100K, 5,000K, and 20,000K records.

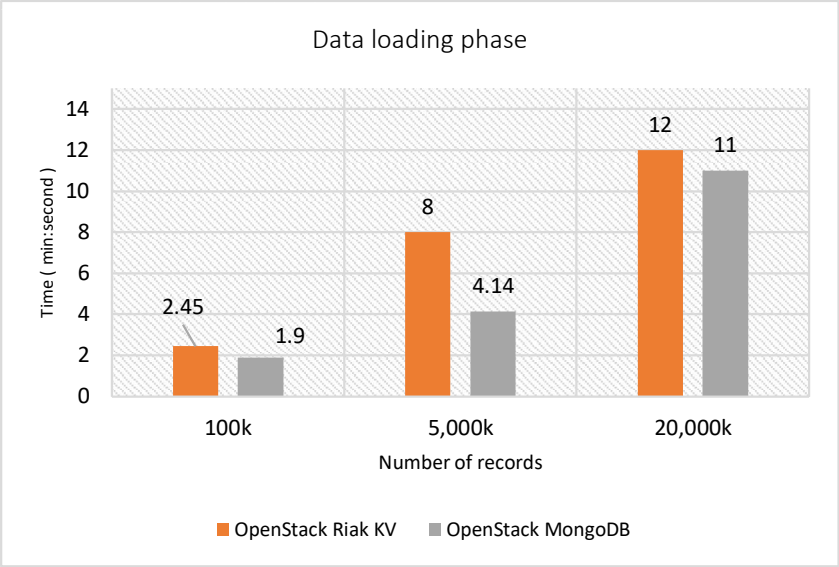


Figure 4.25 Experiment 2's data loading test in cloud cluster

Figure 4.25 shows the results of data loading evaluated by comparing the execution time of two tested databases while loading 100K, 5,000K, and 20,000K records. We can note that there is no significant difference between MongoDB and Riak KV. MongoDB has a better insertion time, regardless of the number of records, compared to Riak KV. We also notice that loading time increases with the increase in the number of records until reaching 12 milliseconds for OpenStack Riak KV and 11 milliseconds for OpenStack MongoDB when inserting 200,000K records.

1. **Workload A:** 50/50 reads and updates.

In experiment 2, we examine workload A with 50% reads and 50% updates. Charted throughput and latency curves are generated for each cloud serving system. As Figure 4.26 shows, MongoDB achieved the best throughput and the lowest latency for reads. Riak KV has high latency for reads, and it reached the lowest throughput at 18 operations/second with 20,000K records. The update operations show that Riak KV provides the highest latency for updates, where the highest value was 15.5 milliseconds. MongoDB achieved lower latency for updates and the latency of Riak KV was slightly higher than that of MongoDB for workload A.

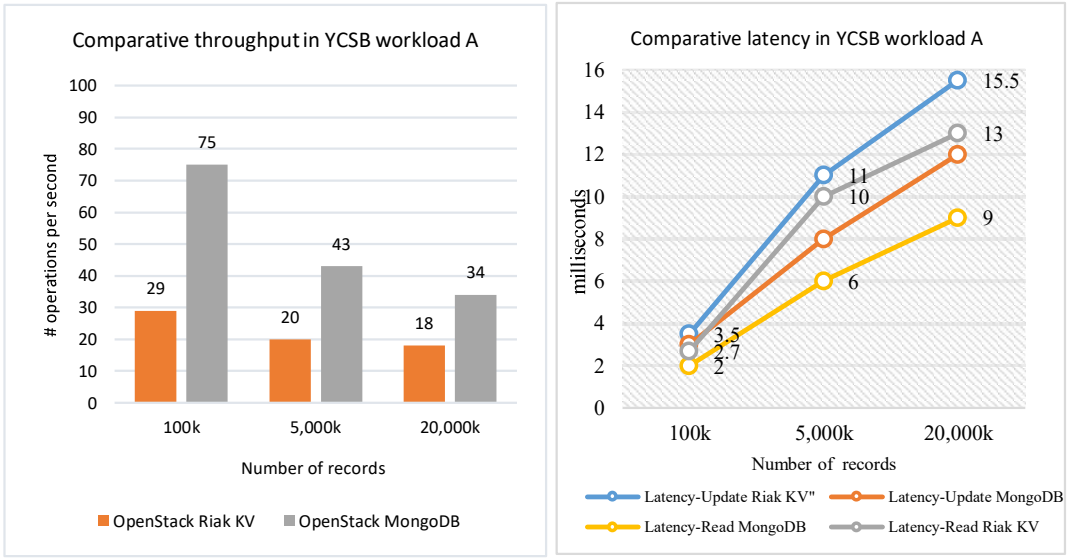


Figure 4.26 Experiment 2, workload A in cloud cluster

2. **Workload B:** 95/5 reads and updates. Workload B was run with 95% reads and 5% updates. Figure 4.27 shows the results.

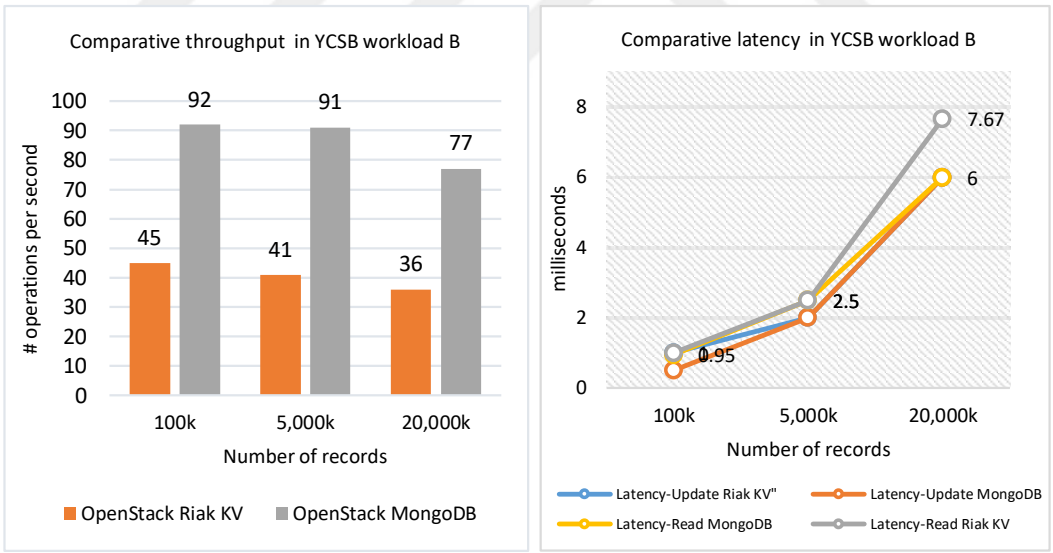


Figure 4.27 Experiment 2, workload B in cloud cluster

MongoDB in OpenStack reaches 92 operations/second with the lowest latency for reads. For the latency results, almost all read results were close in time. For update

operations, MongoDB still reached the best throughput, and the latency decreased as the offered throughput increased.

3. **Workload C:** 100% reads only.

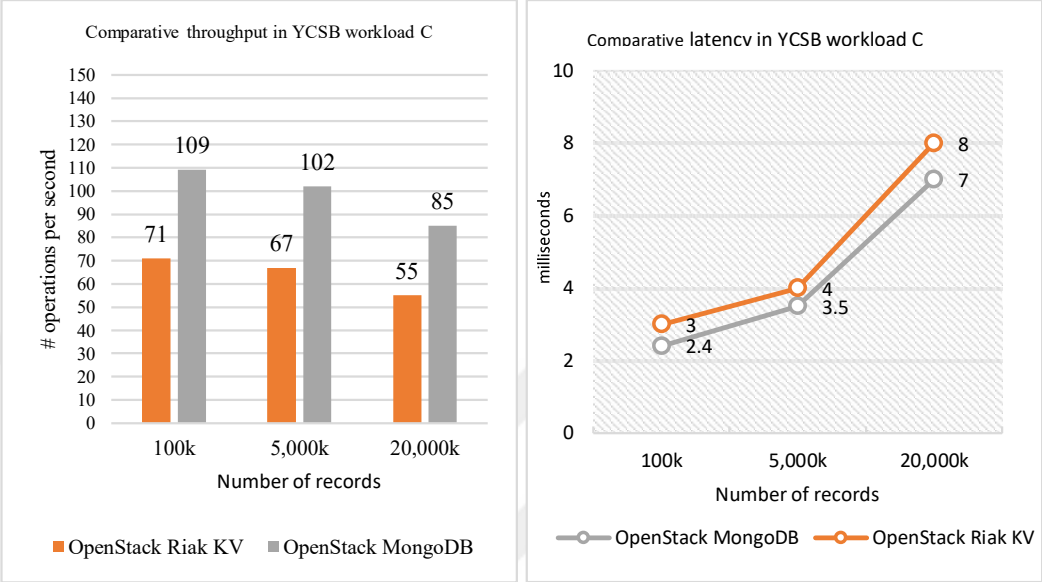


Figure 4.28 Experiment 2, workload C in cloud cluster

From Figure 4.28, we can conclude that for reading operations only, MongoDB is more efficient than Riak KV. MongoDB showed similar behavior to that of the previous workload when the data size was 100 MB (100K), but there was a difference with the increase in the number of records (5,000K, 20,000K). The throughput performance was different in OpenStack with 5,000K records at 67 operations/second for Riak KV compared to 102 operations/second for MongoDB, and also with 20,000K records with throughput performance of 55 versus 85 operations/second, respectively. The latency results show that Riak KV in OpenStack has the highest latency time of 8 milliseconds with 20,000K records.

4. **Workload D:** 95/5 reads and inserts.

Figure 4.29 shows the results obtained while executing workload D, a read-insert workload. A total of 10,000 operations were performed over databases with 100K, 5,000K, and 20,000K records stored.

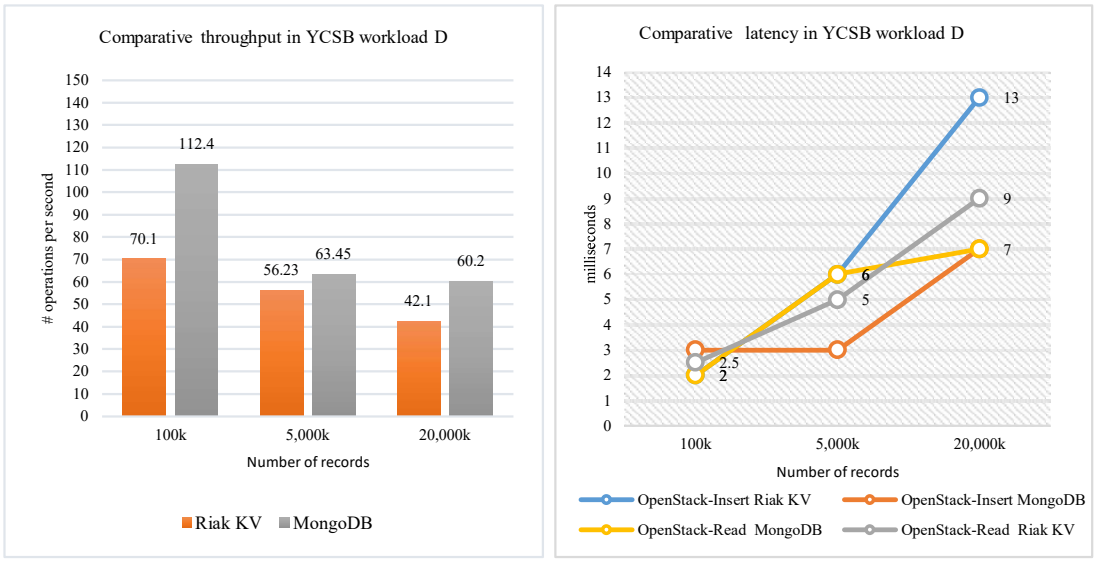


Figure 4.29 Experiment 2, workload D in cloud cluster

The results illustrated in Figure 4.29 are indicative of the superiority of MongoDB over Riak KV for all database volumes. With an increase in data size, both MongoDB and Riak KV started having lower throughput time, but Riak KV was not even close to MongoDB. Its highest performance reached 112.4 operations/second when the number of records was 10,000K.

5. **Workload E:** 95/5 scans and inserts.

In this experiment, MongoDB performed better than Riak KV and the throughput was generally low, as observed from Figure 4.30, due to the scanning process. The latency values were very close.

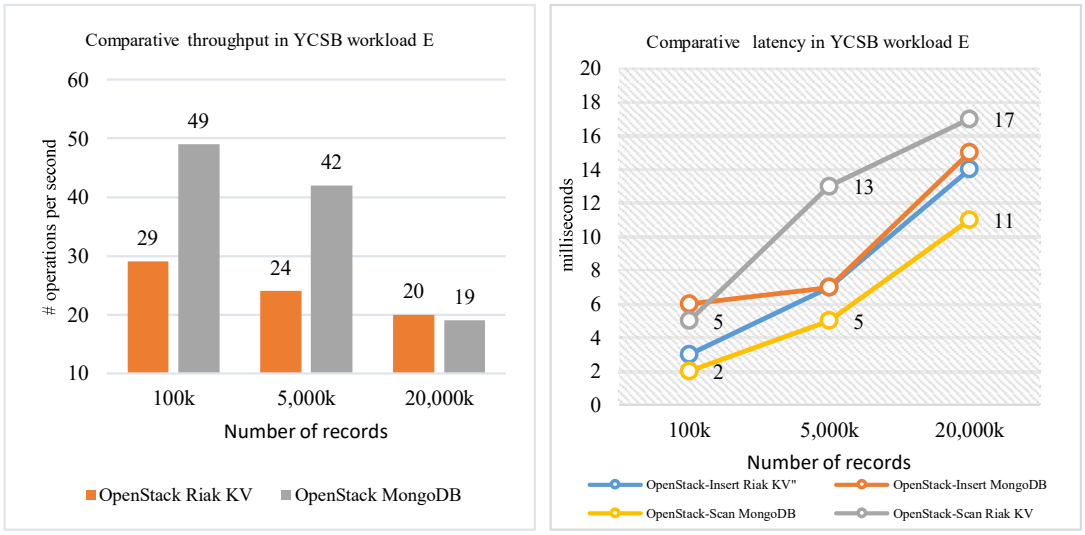


Figure 4.30 Experiment 2, workload E in cloud cluster

6. **Workload F:** 50/50 reads and read-modify-writes.

Workload F is a read-modify-write workload with 50% reads and 50% read-modify-writes. In this workload, the YCSB client reads a record, modifies it, and writes it back to the database. The results are illustrated in Figure 4.31. As the offered throughput increased, the operation latency of Riak KV increased. Riak KV showed better results compared to MongoDB when the number of keys was 100K with 66 versus 61 operations/second. The read-modify-writes latency was very high in these operations, reaching 21 milliseconds with Riak KV.

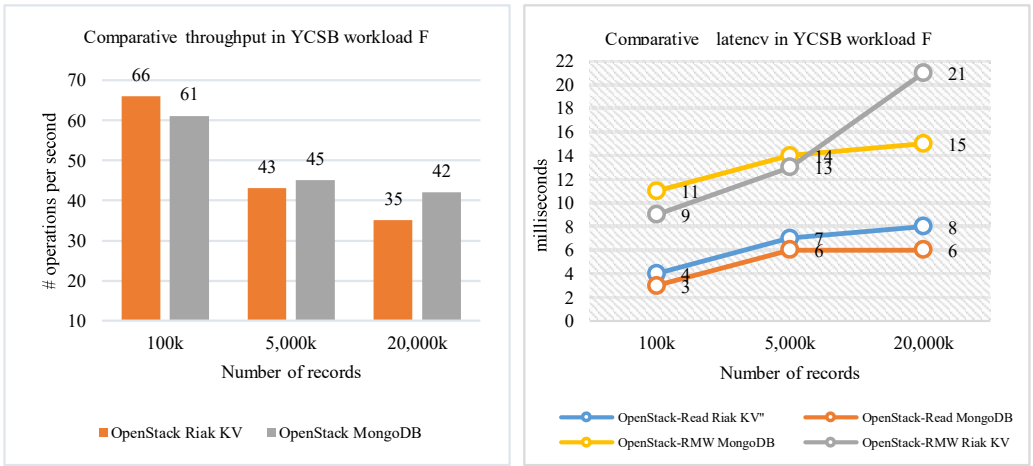


Figure 4.31 Experiment 2, workload F in cloud cluster

4.2.4 Experiment 3: Evaluating and testing in Google Cloud

4.2.4.1 Performance configuration and structure for experiment 3

This experiment was done in Google Cloud with the same configuration as the previous experiment, with 5 VM OS Ubuntu 14.04.5 x64 with 4 GB RAM available, size 4 vCPUs/40 GB. The tested versions of the NoSQL databases are MongoDB version 4.0, Riak KV version 2.2.3, and YCSB 0.15.0.

Figure 4.32 illustrates the experimental structure with details of the primary components.

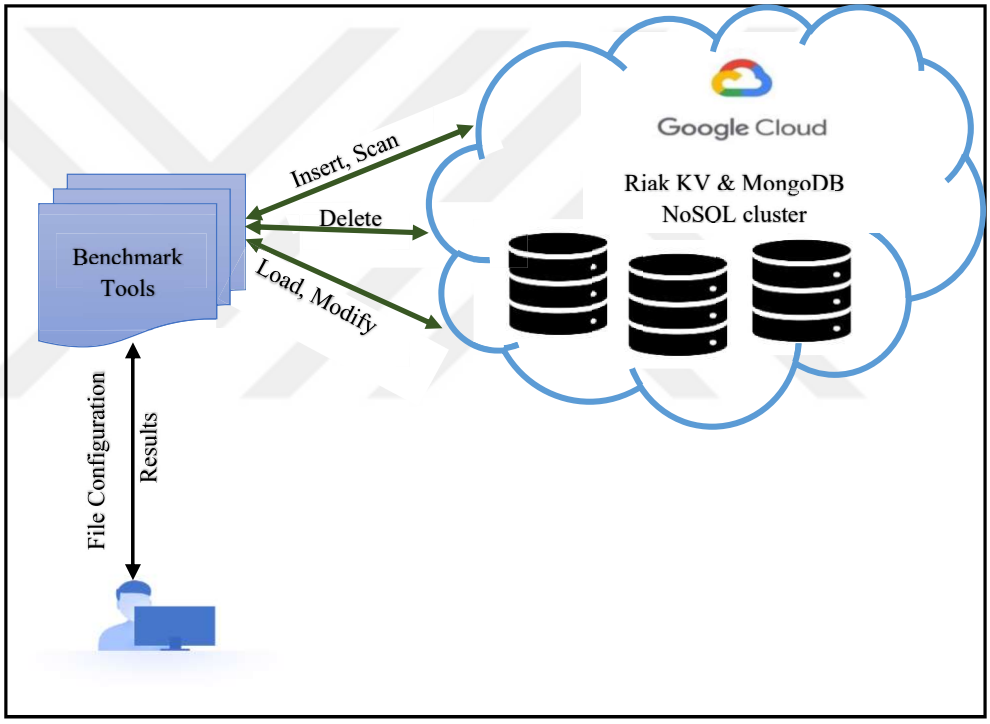


Figure 4.32 Experiment 3, structure in cloud cluster

4.2.4.2 Experimental test and results

The following section presents and analyzes the results of the YCSB-generated load for 100K, 5,000K, and 20,000K records.

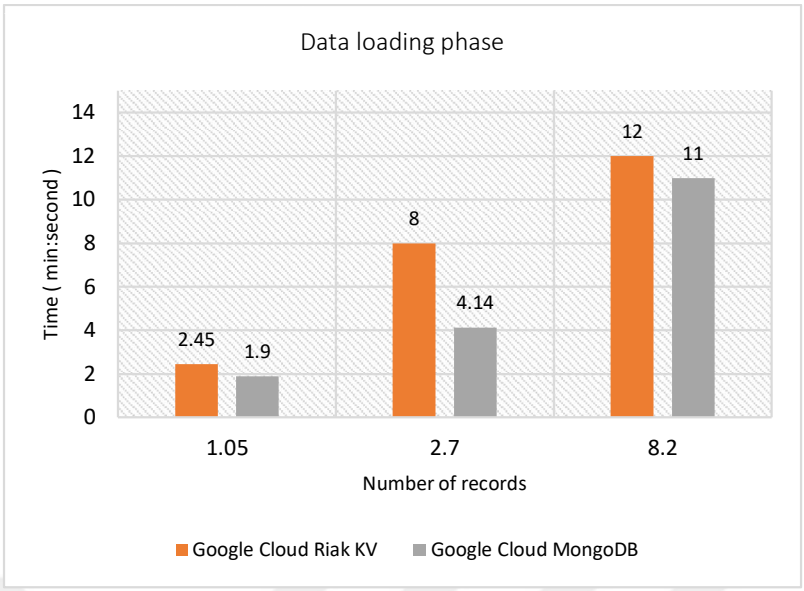


Figure 4.33 Experiment 3, data loading test in cloud cluster

The graph in Figure 4.33 shows the overall results of the loading benchmark. During the testing, no significantly different load times were observed for MongoDB and Riak KV. MongoDB had lower insert time, regardless of the number of records, compared to Riak KV. The loading time increased with the increase in the number of records.

1. **Workload A:** 50/50 reads and updates.

As Figure 4.34 shows, MongoDB in Google Cloud achieved the best throughput with 100K and 5,000K keys at 98 and 81 operations/second, respectively. When the number of keys was increased to 20,000K for Riak KV, lower throughput was obtained compared to the other database, at 17 versus 53 operations/second, respectively. MongoDB in Google Cloud achieved the best throughput and the lowest latency for reads. The update operations showed that Riak KV provided the highest latency for updates, with the highest latency value of 13 milliseconds.

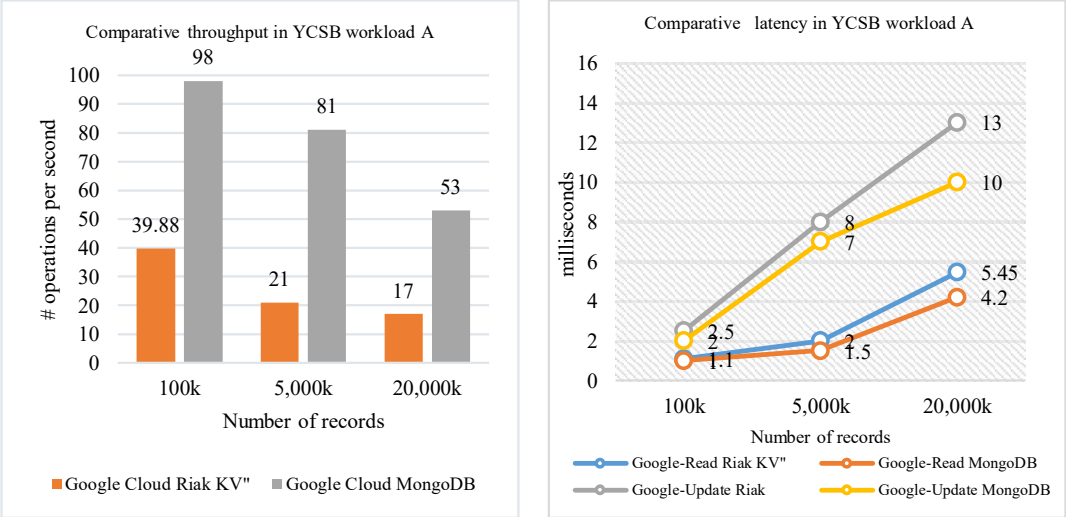


Figure 4.34 Experiment 3, workload A in cloud cluster

2. **Workload B:** 95% reads and 5% updates.

Workload B shows read and low update operations. MongoDB in Google Cloud peaked at 102 operations/second with the lowest latency for reads. Furthermore, Riak KV reached 57 operations/second when the number of records was 100K. For latency results, the delay time in reading was 6.2 and 5.5 milliseconds for Riak KV and MongoDB, respectively. For update operations, MongoDB in Google Cloud achieved the best throughput and the latency decreased; see Figure 4.35 for details.

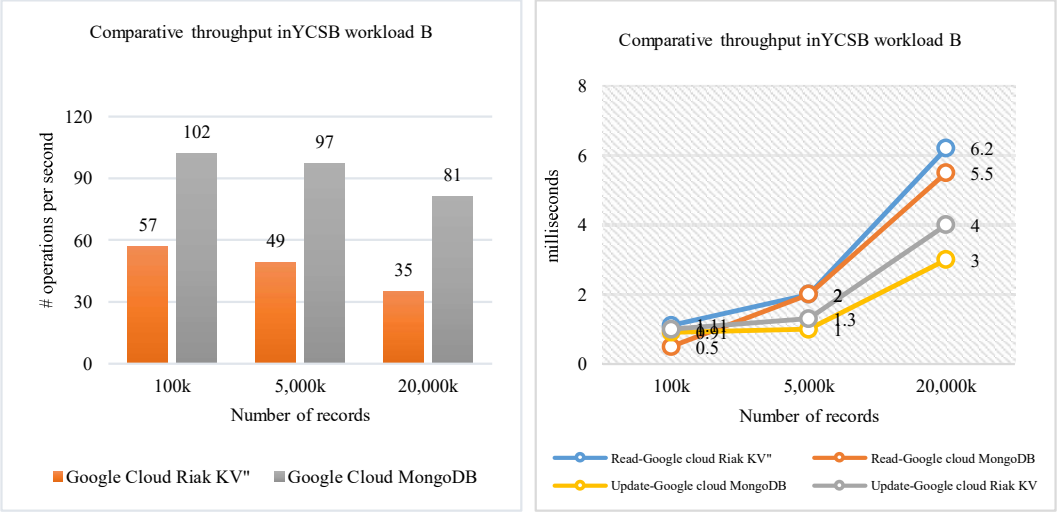


Figure 4.35 Experiment 3, workload B in cloud cluster

3. **Workload C:** (100% reads only)

In this workload, all the operations were read-only. The results are illustrated in Figure 4.36. MongoDB achieved the highest performance of 187 operations/second when the number of records reached 100K. The performance decreases when we increase the number of records. For example, for 5,000K Riak KV and MongoDB achieve 83 and 105 operations/second, respectively, and the latency is lower for Riak KV with 20,000K records at 4 milliseconds. For MongoDB, this latency value was higher at 6 milliseconds.

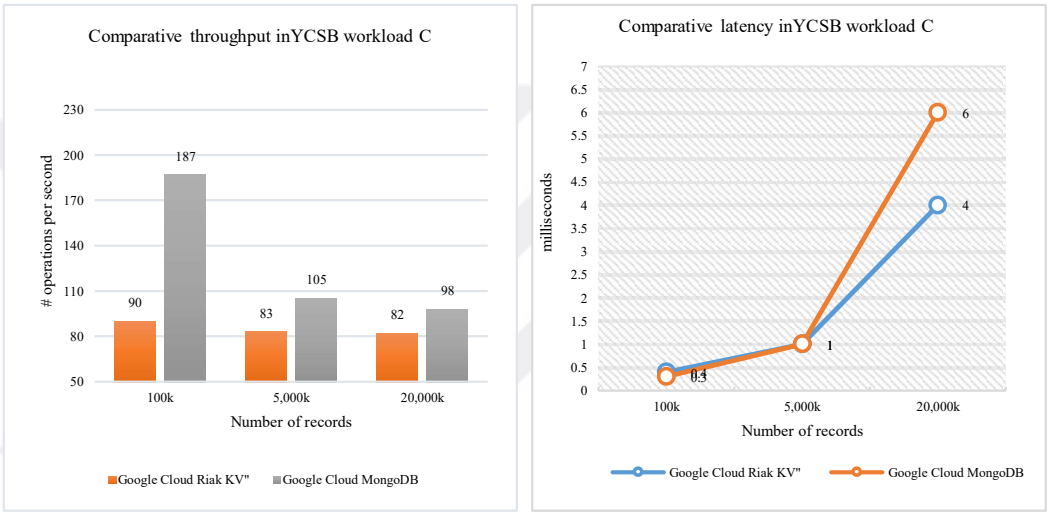


Figure 4.36 Experiment 3, workload C in cloud cluster

4. **Workload D:** 95/5 reads and inserts.

Workload D was run with 95% reads and 5% inserts. Figure 4.37 shows the results obtained while executing this read-insert workload. MongoDB showed superiority in Google Cloud for all database volumes. With an increase in data size, both MongoDB and Riak KV began showing lower throughput time, but the performance of Riak KV was not even close to that of MongoDB. MongoDB had the highest performance when the number of records was 10K, reaching 113 operations/second, and its throughput performance was also high for all dataset sizes at 113, 98, and 73 operations/second. The latencies were very close, although Riak KV had lower latency values of 3 and 5 milliseconds for 5,000K and 20,000K keys.

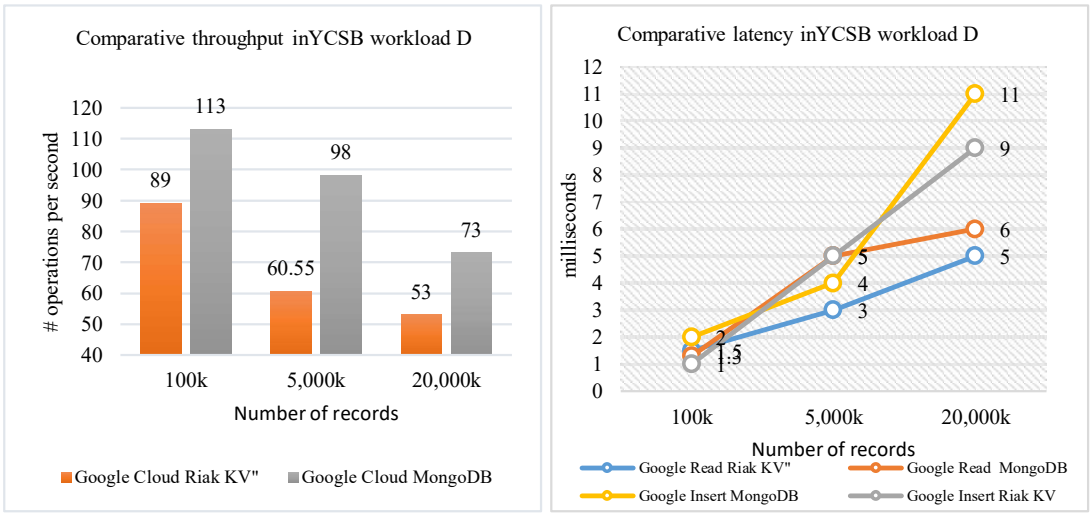


Figure 4.37 Experiment 3, workload D in cloud cluster

5. **Workload E:** 95/5 scans and inserts.

In this workload, MongoDB performed better than Riak KV and the throughput was generally low, as observed in Figure 4.38, due to the scanning process. The performance was not excellent compared to previous tests. There were no more than 52 operations/second when the number of records was 100K. Figure 4.38 shows the results.

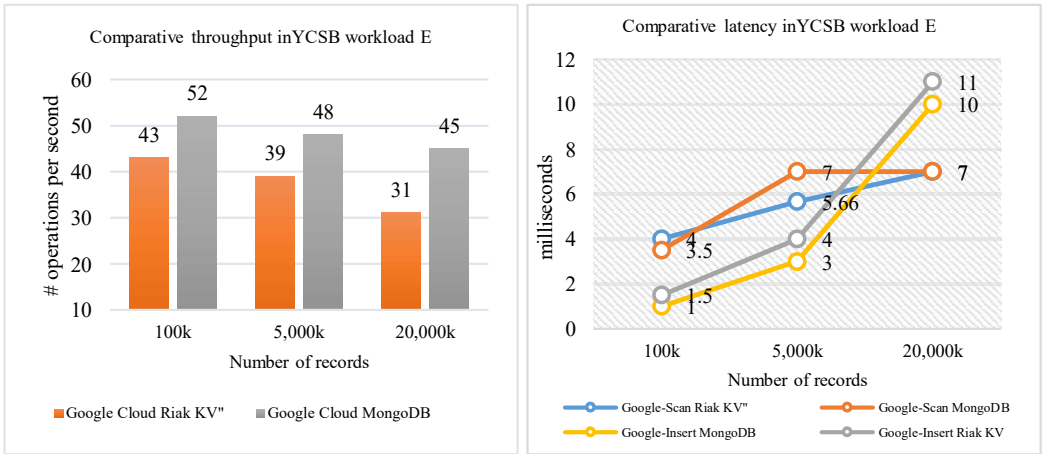


Figure 4.38 Experiment 3, workload E in cloud cluster

6. **Workload F:** 50/50 reads and read-modify-writes.

For this workload, the throughput was high for both databases, especially with the increase in the size of data. However, Riak KV achieved better performance than MongoDB; for example, at 20,000K records, the results were 63 and 56 operations/second, respectively. The latency results were expected because the process of reading and modifying takes more time than the process of reading only.

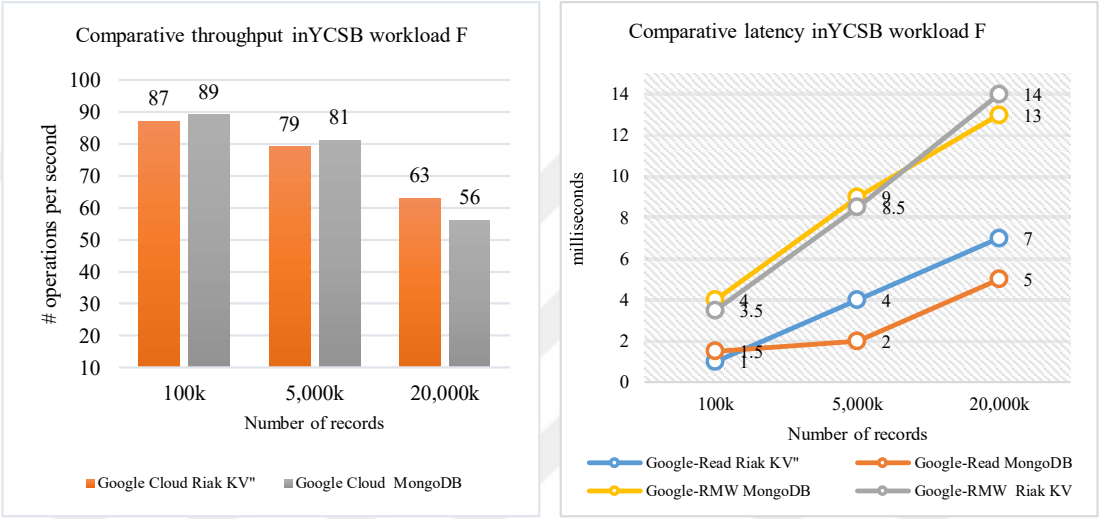


Figure 4.39 Experiment 3, workload F in cloud cluster

4.3 Summary

This chapter began with the presentation and the evaluation of the performance of the Riak KV database. In the experiments explained in Section 4.1, it was observed that Riak KV generally has better throughput performance when we increase the number of threads, which is ideal for handling big data applications and provides a powerful tool for storing massive amounts of unstructured data. In Section 4.2, we examined the use of different CC platforms for a comparative evaluation of the performance of two NoSQL databases: Riak KV and MongoDB. These results indicated that the performance of each database varies with the CC platform used. In summary, MongoDB in Google Cloud had the highest throughput performance for workloads A, B, C, and D. In general, MongoDB's performance was better, especially when using it with Google Cloud.

CHAPTER 5

RESULTS AND DISCUSSION

This chapter presents a wide and comprehensive analysis of the results obtained in Chapter 4 and compares the results with each other, also providing a framework that includes a summary of these results and comparisons.

5.1 Section 1: Internal cluster

5.1.1 Discussion of experiment 1: Evaluating the Riak KV cluster by YCSB

The results were obtained after a general process of setting up and installing the Riak KV NoSQL database management system in the cluster environment to simulate a popular use case for this NoSQL database. Commodity hardware was used as the cluster hardware for testing and evaluating Riak KV using YCSB and factors such as data size and number of threads were also tested. The execution times of this NoSQL database were compared for different types of workloads and different numbers of records.

In the tests, 1, 4, 8, and 12 threads were used, assuming that adding more threads would result in the execution time decreasing and in the best performance being obtained. However, as we saw in Chapter 4, the execution times of the small and large records were approximately equal to each other. We also found that with the increase in the number of threads, the throughput performance of Riak KV improved with bigger datasets. We found that Riak was more efficient overall with the mixed operations of 50% reads and 50% updates of workload A.

Contrary to the intuitive assumption that parallelization always enhances performance, these results show that with small amounts of data, the performance does not necessarily improve when launching more threads. There has not been much research on the effect of parallelization on the performance of NoSQL database systems when deployed in commodity hardware environments.

In summary, the Riak KV database provided the best throughput performance when the number of threads was increased. However, with smaller numbers of records, the

execution time for every workload was higher. We observed that Riak KV generally has better throughput performance when we increase the number of threads, which is ideal for handling big data applications and provides a powerful tool for storing massive amounts of unstructured data.

The examination of the effect of synchronization is very important for the sake of enhancing benchmarks. The more we know about how processing data simultaneously on these systems affects their performance, the more feasible it will be to make the benchmarks for these systems better tools for measuring them.

5.1.2 Discussion of experiment 2: Evaluating the Riak KV cluster by Basho-bench

Experiment 2 aimed at the analysis and evaluation of the read/update throughput as well as the latency of the cluster environment of the Riak KV NoSQL database management system. To achieve that goal, Basho-bench was used. In the benchmarking of NoSQL data stores from the perspective of the cluster environment and monitoring, factors such as throughput and latency are important as some differences exist among NoSQL databases and the utility differs from one application to another. In addition, system performance is still an important factor when processing large amounts of data. Measurements were performed in three sets of experiments with different numbers of operations, applying workloads A, B, and C. The read throughput and latency were measured for each of these workloads, as well as update throughput and latency. We saw that the performance was significantly affected by increased data size. We also found that with an increase in the number of threads, the throughput performance was better and the latency factor was reduced. The tests utilized 4, 8, and 12 threads, demonstrating that adding more threads resulted in the throughput performance of Riak KV improving. We found that for operations using 10 million and 200 million keys with 12 threads, the performance could be improved by 7.85 and 9.34 times, respectively, compared to 4 threads. Summarizing the results for workloads A, B, and C, we can note that the increase in the number of threads has a significant effect on the performance of Riak KV NoSQL databases, whereby increasing the number of threads improves the performance. However, performance measures vary from one experiment to another. The throughput effects of update and read operations

were found to be equal in workload A, such that they were low compared to the other experiments. Figure 5.1 shows a comparison of throughput for the three tested workloads.

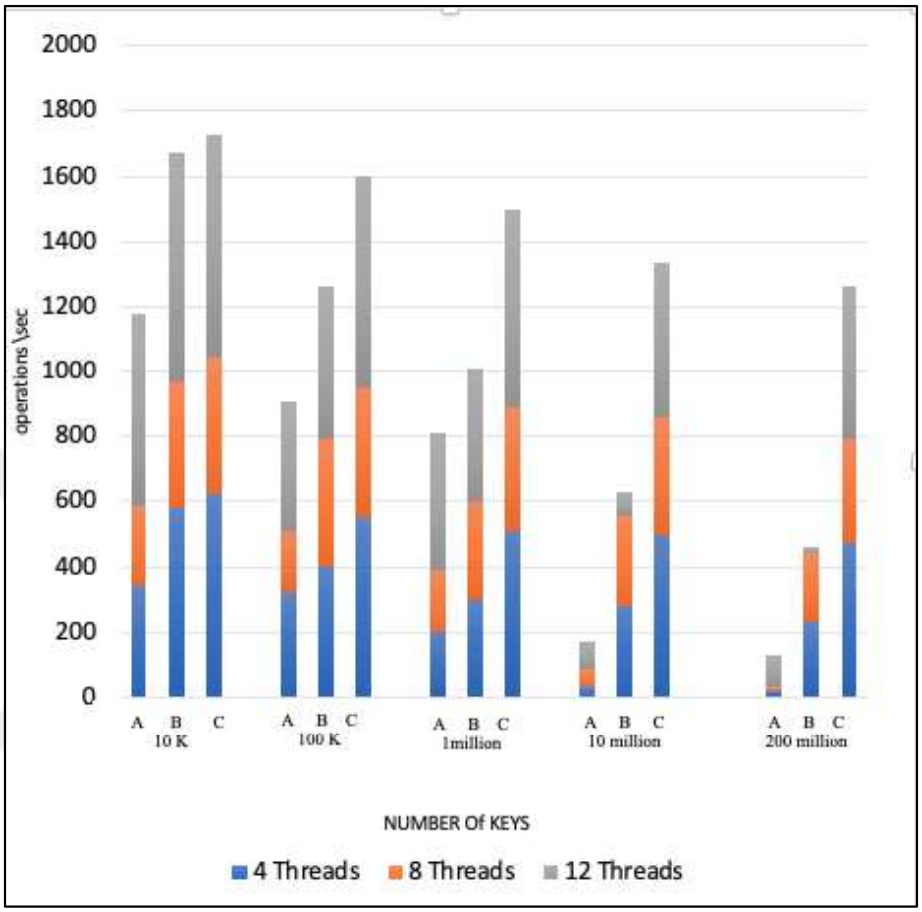


Figure 5.1 Comparing the throughput of experiment 2 in an internal cluster

5.2 Section 2: Cloud cluster environment

5.2.1 Experiment 1: Evaluating and testing in DigitalOcean

Among the experiments conducted in section 2 of Chapter 4, the first experiment was designed for evaluating and testing in the DigitalOcean cloud computing environment. The results were obtained during the execution of different workloads with different data, divided according to operation types to allow us to fully understand the performance of each database. Figure 5.2 shows the comparative throughput times and

number of operations per second for each of the tested databases. These results represent the total throughput time of all executed workloads (A, B, C, D, E, and F).

Figure 5.2 Comparing the throughput of experiment 1 in DigitalOcean

Benchmarking was done for each system with 100K, 5,000K, and 20,000K keys. The number of keys is a proxy for the density of data in the system. It was observed that MongoDB outperformed Riak KV in terms of throughput with increasing numbers of keys in workload F. Riak KV started to display reduced performance, sometimes showing poor results. In contrast, MongoDB became faster while working with an increase in data. Also, after running different workloads to analyze read, insert, scan, and update performances, it was possible to conclude that when it comes to update operations, MongoDB is faster than Riak KV, providing higher throughput time. In the figure above, we can see a large difference in throughput for data size of 100K in workload C, whereby MongoDB reached 135 operations/second and Riak KV

achieved only 88.11 operations/second. However, this difference decreased as the volume of data increased. For example, when the data volume was 20,000K in workload C, MongoDB achieved a rate of 75.22 operations/second while Riak KV achieved 74.09 operations/second.

5.2.2 Experiment 2: Evaluating and testing in OpenStack

For the second experiment, evaluating and testing in OpenStack, Table 5.1 shows the sum of the throughput for the tested databases for all workloads at once. As can be seen here, MongoDB had the highest throughput performance for all workloads and, in general, MongoDB's performance was better.

Table 5.1 The sum of throughput (operations/second)

Workload	**A**	**B**	**C**	**D**	**E**	**F**
MongoDB	152	260	296	208	110	148
Riak KV	67	122	193	170	73	144

MongoDB had the lowest read latency in most cases because it has the space capability to handle spatial queries quickly. MongoDB is fast for reads because its shards the data across nodes when a query is launched and only the concerned nodes will respond to the query. This avoids the need to go over the whole dataset. MongoDB thus shows lower latency. It does not offer the best update, and insert latency, but it has the highest throughput; this shows that MongoDB has more parallelism. Riak KV demonstrates the worst read latency for workloads C, D, and F. In general, Riak KV has the highest mean latency, as shown in Table 5.2.

Table 5.2 Mean latency (operations/second)

Workload	A		B		C	D		E		F	
	Read	Update	Read	Update	Read	Read	Insert	Scan	Insert	Read	R/M
MongoDB	5.6	7.6	2.9	4.4	4.3	5	4.3	6	9.3	5	13.3
Riak KV	8.5	10	3.7	4.8	5	5.5	7	11	8	6.3	14.3

5.2.3 Experiment 3: Evaluating and testing in Google Cloud

Table 5.3 shows the sum of throughput for the tested databases for all workloads at once. In summary, MongoDB in Google Cloud had the highest throughput performance for workloads A, B, C, and D. In general, MongoDB's performance was better, especially compared to DigitalOcean. In Table 5.3, values in green represent the best performance, while those in red signify the worst performance.

Table 5.3 The sum of throughput (operations/second)

Workload		A	B	C	D	E	F
MongoDB	DigitalOcean	205.34	263.1	288.02	236.05	**154.01**	158
	OpenStack	152	260	**296**	208	110	148
	Google Cloud	**232**	**280**	390	**284**	145	226
Riak KV	DigitalOcean	72.89	130.32	239.08	**168.33**	79.57	159
	OpenStack	**67**	**122**	**193**	170	**73**	**144**
	Google Cloud	77.88	141	255	202.55	113	**229**

Examining Table 5.4, we can compare the latency between MongoDB and Riak KV in these three clouds. Values in green indicate the lowest latency while those in red represent the highest latency. MongoDB achieved low latency in Google Cloud for workloads A, B, and D. In contrast, Riak KV had the highest mean latency in OpenStack for workloads A, B, C, E, and F.

Table 5.4 Mean latency (operations/second)

Workload		A		B		C	D		E		F	
		R	U	R	U	R	R	I	S	I	R	RMW
MongoDB	DigitalOcean	**2.2**	**5.9**	3.1	2.8	3	4.5	5.5	6.1	5.5	2.8	12.1
	OpenStack	5.6	7.6	**2.9**	4.4	4.3	5	**4.3**	6	**9.3**	5	13.3
	Google Cloud	2.4	6.3	3.1	2.6	2.4	4.1	5.6	5.8	**4.6**	2.8	**8.6**
Riak KV	DigitalOcean	2.5	7.1	**3.7**	3	2.8	4.4	5.3	**7.6**	7	**2.7**	11.8
	OpenStack	**8.5**	**10**	**3.7**	**4.8**	**5**	**5.5**	**7**	11	8	**6.3**	**14.3**
	Google Cloud	2.6	7.8	3.1	**2.1**	**1.8**	**3.1**	5	**5.5**	5.5	4	**8.6**

R: Read. U: Update. I: Insert. S: Scan. RMW: Read-Modify-Write.

5.3 Evaluation summary

In the experiments of section 1, the Riak KV database was tested and evaluated using benchmark tools (YCSB, Basho-bench) for big data clusters, where huge amounts of data are stored and retrieved in different amounts in a distributed database environment. The execution times of the NoSQL database over different types of workloads and different sizes of data were compared. The results show that Riak KV is stable in execution time for both small and large amounts of data, and the throughput performance increases as the number of threads increases.

From the experiments of section 2, a decision tree was designed based on the analysis and evaluation of big data clusters in CC environments. YCSB was used to benchmark the KV and document store for Riak KV and MongoDB in DigitalOcean, OpenStack, and Google Cloud and the decision tree was designed based on the analysis of those experiments.

5.3.1 Decision tree

A decision tree was created based on the experiments in the cloud cluster environment and the resulting analysis. Some difficulties and the possible solutions for overcoming problems in choosing the right database and the right cloud with different variables are addressed here. For developers to reach possible solutions, it is imperative that they be able to confidently choose the most suitable NoSQL database and a good CC. The decision tree, provided in Figure 5.3, is divided into six sections according to the type of operation. A detailed analysis of the decision tree is provided below.

1. The top split in the tree is workload A; this workload comprises 50/50 reads and writes. We see from the decision tree that the highest performance was achieved with MongoDB in Google Cloud. Riak KV in OpenStack should be avoided. MongoDB in DigitalOcean has low latency for reads and updates.
2. The second split, for workload B, has the same results as for workload A, but the difference in low latency represents the best and most suitable option. MongoDB in OpenStack is recommended for read operations and Riak KV in Google Cloud is recommended for updates.

3. Workload C focuses on read operations only, showing results similar to those obtained for workload A. However, because of the difference in latency for read-only operations, developers would do better to choose Riak KV and Google Cloud to avoid the high latency seen with Riak KV in OpenStack.
4. In the fourth split of the decision tree, for workload D, Google Cloud gives the highest performance with MongoDB and the lowest latency with reads in the operations. MongoDB in OpenStack achieves the lowest latency for insert operations.
5. In workload E, short ranges of records are queried. Most of the poor results in OpenStack with Riak KV are due to the limited support of this cloud and the difficulty of dealing with it. We can see in the decision tree that high latency occurred in OpenStack with Riak KV for scan operations as well as with MongoDB for inserts.
6. In workload F, the client reads a record, modifies it, and writes back the changes. For the first time, Riak KV achieves the highest performance in Google Cloud. Thus, this is the best choice when operations need to be read, modified, and written. To avoid high latency, developers should choose Riak KV and DigitalOcean.

The new database added to the framework as explained in Section 3.1 has the following steps:

(I): Install the required database in the first step (pre-processing) on all VMs or commodity hardware.

(II): Configure benchmark file for a new database.

(III): Moving to the cluster architecture setup, we establish the network topology that the database works with and the creation of the architecture based on the number of nodes used.

(IV): Throughput and latency are some of the most common ways that database performance is measured. Many different metrics may be used in the evaluation and analysis of NoSQL databases. Being able to evaluate the efficiency of the system provides a metric for measuring database performance. Measuring the level of throughput or latency can also help identify performance issues in the system.

(V): The results obtained in the previous step are presented in several forms (graphs, tables, curves, and decision trees).

R: Read **U:** Update **I:** Insert **S:** Scan **RMW**: Read-Modify-Write

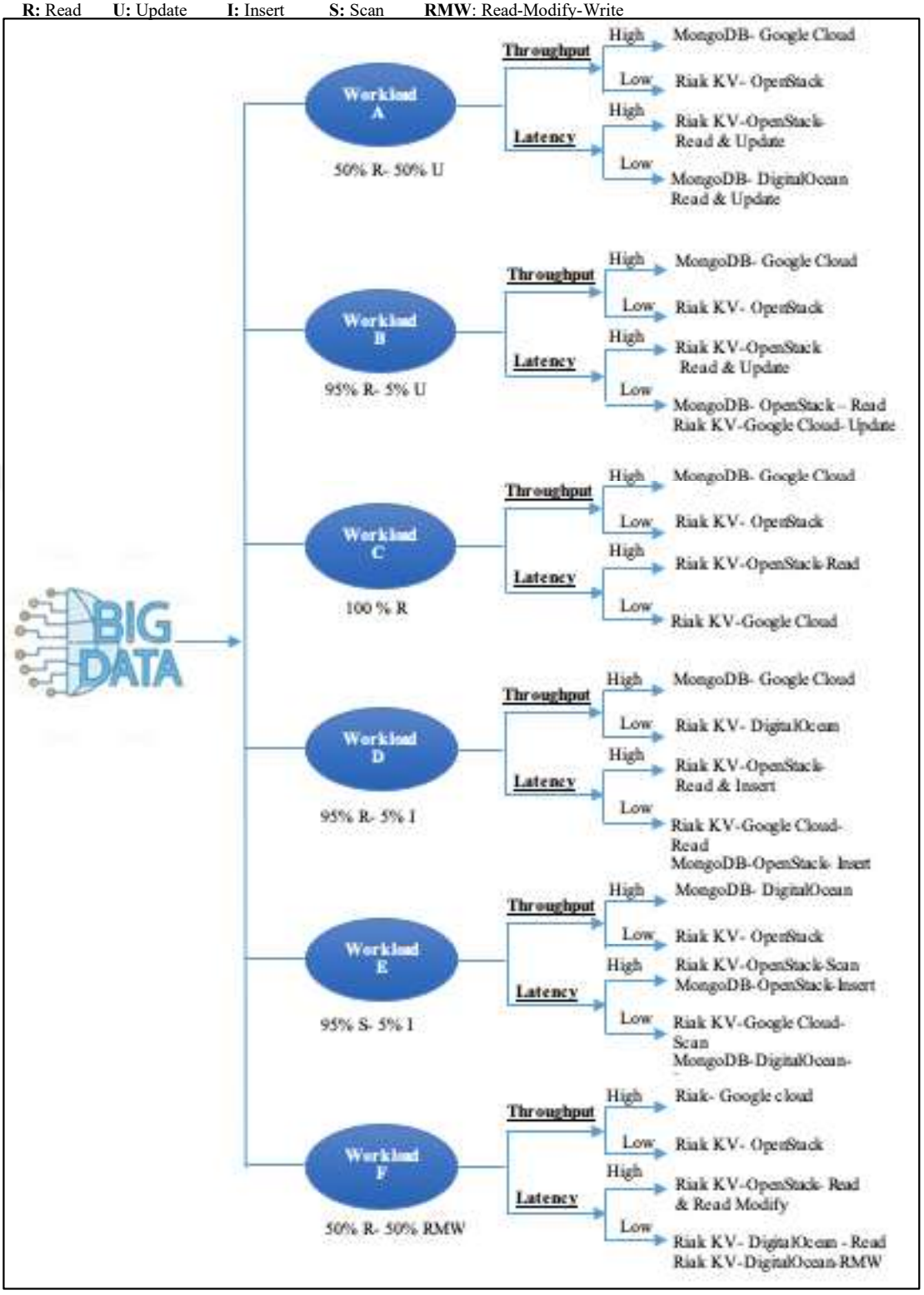


Figure 5.3 Decision tree for evaluating and testing in cloud computing platforms

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

With the growth in data continuing rapidly, the opportunities for storing and processing data with NoSQL databases have also evolved. NoSQL databases are expected to respond to all of the demands of different web applications and enterprise systems in terms of fast and dynamic data storage and retrieval in the not-so-distant future. This is because they bring a set of additional features that distinguish them from standard RDBMSs, such as not requiring a rigid database schema to be defined and having easy horizontal scalability. Meanwhile, the concept of big data has also appeared, and it corresponds to data whose volume, velocity, and variability are difficult to process using traditional data management tools and techniques. Big data must be extracted from the web and stored for future querying. NoSQL databases have become important alternatives to traditional relational databases. These databases are prepared by the management of large datasets that are continuously and variably changing. They are widely used for cloud databases and distributed systems. With NoSQL databases, static schemes and many other restrictions are avoided. In the era of big data, such databases provide scalable solutions with high availability.

This dissertation began with a presentation of the scope of the research, describing the motivation and the research problem in Chapter 1 together with an identification of the research objectives and the contributions of the study. Chapter 2 focused on the related works, providing a literature review of NoSQL databases, big data, CC, and other relevant topics. Chapter 3 introduced the methodology, outlining the architecture and the design framework based on the identified motivations and the review of related research works and core technologies. Chapter 4 then presented the experimental analysis and research implementation, while Chapter 5 discussed those results and evaluated and analyzed them. Finally, in this chapter, conclusions are drawn and the

contributions of this dissertation are outlined, together with suggestions for future work.

Although there are many NoSQL databases, engineers are often confused in deciding which type of database will be most suitable for the analysis and evaluation of big data and which architecture will best fit huge amounts of data. This is, in fact, still an open problem. There has not been much research on solutions for big data that provide a practical, experimentally driven characterization of the efficiency and suitability of analysis of big data. The field of big data remains an open question, and solutions particularly depend on the technology used and what one is trying to achieve through the use of big data. Big data is an essential concept entailing data that do not conform to the common structure of a traditional database. This concept of big data comprises different types of key technologies that work together to obtain end purposes like extracting values from data that would have been considered dead before.

In this dissertation, a performance evaluation model was developed to evaluate and analyze the benchmarking of NoSQL databases using a cloud environment where these systems are deployed (i.e., in a cloud or a local data center cluster). The model was then evaluated with the most popular NoSQL data stores, MongoDB and Riak KV, in cloud computing and internal cluster environments to analyze the performance of those two systems. A number of aspects of performance were compared, including monitoring throughput and latency. The model architecture has the ability to use other databases as well as other clouds.

In an internal cluster, commodity hardware was used for testing and evaluating the cluster hardware with Riak KV using the YCSB and Basho-bench benchmarks, and factors such as data size and number of threads were also tested. The execution times of this NoSQL database were compared for different types of workloads and different numbers of records. We saw that the execution times of small and large numbers of records were approximately equal to each other. It was also found that with the increase in the number of threads, the throughput performance of Riak KV improved with bigger datasets. Riak KV was more efficient overall for mixed operations by YCSB and for read-only operations by Basho-bench.

In a cloud cluster environment, three different cloud computing platforms were used. For all the experimental analyses, YCSB was applied, as it is one of the most used benchmarks to test NoSQL databases. The model showed that the best performance can be achieved constantly with MongoDB in Google Cloud. The results also showed the limitations of Riak KV in both the Google Cloud and OpenStack clouds, especially in terms of latency. The main finding of this work was that, with the studied model, the performance of Riak KV increases significantly with the increase of the number of threads in the system. A decision tree was then presented based on the many experiments to determine the performance criteria for selecting among these databases and CC platforms. Using the decision tree, it is possible to calculate the rates of higher throughput as well as lower throughput. The results for throughput and latency obtained and summarized in the decision tree (see Figure 5.3) can be listed as follows: The best performance was obtained by MongoDB in Google Cloud compared to MongoDB in DigitalOcean and Riak KV in Google Cloud and DigitalOcean. Due to the low performance obtained by Riak KV in OpenStack and DigitalOcean, developers may choose Riak in Google Cloud to avoid high latency. On the other hand, we observed high latency for almost all experiments using Riak KV in OpenStack. Developers should take this into consideration to obtain the best performance. The decision tree shows that Riak KV used in DigitalOcean and MongoDB used in DigitalOcean and OpenStack have low latencies.

The following are the contributions made by this dissertation, as originally highlighted in Section 1.2.

1. **To generate a fictitious workload and a data access pattern on the cluster that matches the workloads of real-world applications and monitor its performance in a CP:** This dissertation relied on two benchmarks to generate a fictitious workload. Different sizes of data were generated from 10 MB to 200 GB, and each experiment comprised 10,000 operations. Each of these benchmarks has specific tasks and functions, which were explained in Section 3.2.3.

2. **To observe the performance of NoSQL databases (Riak KV, MongoDB) with large data volumes and various workloads (read, update, mix of reads**

and updates): To monitor the performance of NoSQL databases with a massive amount of data, the proposed model in Section 3.1 was implemented (see Chapter 4) using cloud technologies and benchmark tools including MongoDB and Riak KV.

3. **To monitor the performance of NoSQL databases (throughput, latency) when data are being read, inserted, and scanned and during update operations:** Throughput and latency were carefully monitored in all experiments, as shown in Chapters 4 and 5, to identify the strengths and weaknesses of each NoSQL database.

4. **To create a decision tree for developers and database operators to choose the best CP for applications:** A binary decision tree was presented in Section 5.3.1 that reflects the results obtained from different cloud experiments. The decision tree shows us the ease of confidently choosing the most suitable NoSQL database and CC infrastructure. The decision tree also allows one to select the most suitable CP. As a result, developers can easily follow the decision tree to determine the optimum CP for their needs and enable them to provide their services in the future using the NoSQL database in a CP.

6.2 Future Work

In this research, the performance of databases in various clouds was analyzed and compared. In future work, different CC platforms should be used, because there are a large number of them, as well as different databases over which they may be used, such as column-oriented databases including HBase and Redis. Building big data clusters in CC platforms with different numbers of nodes, with the number of nodes increasing in each test (e.g., 10, 20, 30), would also be helpful in order to study the effect of different configurations on the performance of this architecture. The emergence of IoT applications in different domains will make storing the data generated from them a big challenge. With the continuous generation of data from IoT applications, the challenges will be focused in the areas of storing, managing, and transferring the data efficiently. It is important that future research investigate the presented framework in terms of how to store and manage such data efficiently. This

framework could be expanded in future work to compare the most popular NoSQL databases and traditional RDBMSs in different CCPs in order to find the most suitable database and CCP matches for IoT applications.

REFERENCES

[1] Veronika Abramova, Jorge Bernardino."NoSQL Databases: MongoDB vs Cassandra ." C3S2E '13 Proceedings of the International Conference on Computer Science and Software Engineering, pp 14-22. 2013.

[2] Anasuya N Jadagerimath, P. S. "Efficient IoT Data Management for Cloud Environment using Mongo DB." Third International Conference on Current Trends in Engineering Science and Technology ICCTEST. Grenze Scientific Society. January 2017.

[3] Rakesh Kumar, S. C. "Effective Way To Handling Big Data Problems Using Nosql Database (Mongodb)." Journal of Advanced Database Management & Systems, 42–48, 2015.

[4] Raj R. Parmar, S. R. "MongoDB as an Efficient Graph Database: An Application of Document Oriented NOSQL Database." In M. B. Mittal, Data Intensive Computing Application for Big Data. IOS Press. 2018.

[5] L. Chih-Wei, H. Chih-Ming, C. Chih-Hung, Y. Chao-Tung, " An Improvement to Data Service in Cloud Computing with Content Sensitive Transaction Analysis and Adaptation ." Computer Software and Applications Conference Workshops (COMPSACW), IEEE 37th Annual, pp. 463–468. 2013.

[6] Ibrahim Targio Hashem, Ibrar Yaqoob , Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, Samee Ullah Khan."The rise of “big data” on cloud computing: Review and open research issues." .Information Systems 47 . 98–115. 2015.

[7] Muniswamaiah, Manoj, Tilak Agerwala, and Charles Tappert. "Big Data in Cloud Computing Review and Opportunities." arXiv preprint arXiv:1912.10821, 2019.

[8] F. Bajaber, S. Sakr, O. Batarfi, A. Altalhi, A. Barnawi, "Benchmarking big data systems: A survey." Computer Communications, pp. 241–251.2020.

[9] Alexandros Labrinidis, H. V. (n.d.). "Challenges and Opportunities with Big Data." Proc VLDB Endowment 5(12):2032–2033. https://DOI: 10.14778/2367502.2367572. August 2012.

[10] Balaraju.J and P.V.R.D Prasada Rao. "Recent advances in big data storage and Security schemas of HDFS.". a survey. Journal of Engineering Technology. Special Issue (Emerging Trends in Engineering Technology), PP. 132-138. Mar. 2018.

[11] Ali Hammood, Saran M. "Comparison of NoSQL Database Systems: A Study on MongoDB, Apache Hbase, and Apache Cassandra." international conference on computer science and engineering UBMK. Tekirdag/Turkey: researchgate. 2016.

[12] John Klein, I. G. "Performance Evaluation of NoSQL Databases: A Case Study." .PABS '15 Proceedings of the 1st Workshop on Performance Analysis of Big Data, PP. 5-10. https://DOI: 10.1145/2694730.2694731.15 February 2015.

[13] Arora, Yojna, and Dinesh Goyal. "Review of data analysis framework for variety of big data." Emerging Trends in Expert Applications and Security. Springer, Singapore, 2019.

[14] D.E. O'Leary, "Artificial intelligence and big data, IEEE Intell. " Syst. 28, PP. 96–99. 2013.

[15] K. Ahmed, etc. ,"Age of Big Data and Smart Cities: Privacy Trade Of.", International Journal of Engineering Trends and Technology (IJETT) – Volume 16 Number 6 – Oct 2014.

[16] J.K. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornet, T.-M.-T. Do, O. Dousse, J. Eberle, M. Miettinen, "The mobile data challenge: Big data for mobile computing research.", Workshop on the Nokia Mobile Data Challenge, in: Proceedings of the Conjunction with the 10th International Conference on Pervasive Computing, pp. 1–8.2012.

[17] Ibrahim Abaker Targio Hashem a,n. "The rise of “big data” on cloud computing: Review and open research issues.". Information Systems 47:98-115 ,DOI: 10.1016/j.is.2014.07.006. July 2014.

[18] Sayeth S., Elankovan S., Azuraliza Abu ."Parallel implementation of Apriori algorithms on the Hadoop-MapReduce platform - An evaluation of literature.". Journal of Theoretical and Applied Information Technology. Vol.85. No.3. 31st March 2016.

[19] Maatuk, Abdelsalam, Akhtar Ali, and Nick Rossiter. "A framework for relational database migration.", 2019.

[20] Han, J., E, H., Le, G., & Du, J. "Survey on NoSQL Database." 6th International Conference on Pervasive Computing and Applications. Port Elizabeth, South Africa.IEEE. https://DOI: 10.1109/ICPCA.2011.6106531. 26-28 Oct 2011.

[21] Adity Gupta, S. T. "NoSQL Databases: Critical Analysis and Comparison." International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN). Gurgaon, India:. https://DOI: 10.1109/IC3TSN.2017.8284494. 12-14 Oct. 2017.

[22] Lazar J. Krstić, Krstić . "Testing the Performance of NoSQL Databases via the Database Benchmark Tool." military technical courier. PP.614-639. https://DOI: 10.5937/vojtehg66-15928. July 2018.

[23] Zaki, A. K. "NoSQL databases: new millennium database for big data, big users, cloud computing and its Security challenges." IJRET: International Journal of Research in Engineering and Technology. Volume: 03 Special Issue: 03. https://DOI: 10.15623/ijret.2014.0315080. May 2014.

[24] Eshtay, Mohammed, Azzam Sleit, and Monther Aldwairi. "Implementing Bi-Temporal Properties into Various NoSQL Database Categories." International Journal of Computing, pp. 45-52, 2019.

[25] RIAK KV NoSQL Website. Retrieved from http://basho.com: http://basho.com/products/riak-kv/. Accessed: 2018-12-10.

[26] Redis NoSQL Website. Retrieved from redis: https://redis.io/. Accessed: 2018-10-13.

[27] Apache HBase NoSQL Website. Retrieved from Apache HBase: http://hbase.apache.org/. Accessed: 2018-10-13.

[28] Cassandra NoSQL Website. Retrieved from Apache Cassandra: http://cassandra.apache.org/ . Accessed: 2018-10-13.

[29] Qi, M. "Digital Forensics and NoSQL Databases." 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD). Xiamen, China: 19-21. https://DOI: 10.1109/FSKD.2014.6980927. Aug. 2014.

[30] MongoDB NoSQL Website. Retrieved from MongoDB: https://www.mongodb.com/ . Accessed: 2018-10-13.

[31] neo4j NoSQL Website. Retrieved from neo4j: https:// neo4j.com. Accessed: 2018-10-13.

[32] Santiago Lucas Obrutsky."Cloud Storage: Advantages, Disadvantages and Enterprise Solutions for Business .", Eastern Institute of Technology, Hawke's Bay, New Zealand. 2016.

[33] Mohammad Ubaidullah Bokhari, e.tc, "CLOUD COMPUTING SERVICE MODELS: A COMPARATIVE STUDY.", IEEE Network. 2016.

[34] J. NarenS.K. SowmyaP. Deepika, " Layers of Cloud – IaaS, PaaS and SaaS: A Survey.", IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (3), 4477-4480. 2014.

[35] Urmila R. Pol ,"Cloud Computing with Open Source Tool :OpenStack.", American Journal of Engineering Research (AJER) e-ISSN : 2320-0847 p-ISSN : 2320-0936 Volume-3, Issue-9, pp-233-240. 2014.

[36] Shah, J., "Cloud Computing: The Technology for Next Generation." Journal International of Advances in Computer Science and Technology, 3(3): pp. 152-155, 2014.

[37] Sajid, M. and Raza, Z. "Cloud. Computing: Issues & Challenges.", in International Conference on Cloud. pp. 35-41, 2013.

[38] Marisol, G.-V., Cucinotta, T., and Lu, C., " Challenges in real-time virtualization and predictable cloud computing.", Journal of Systems Architecture (ELSEVIER): pp. 1-15, 2014.

[39] Kumar, S. and Aramudhan, M., "Performance Analysis of Cloud under different Virtual Machine Capacity.", International Journal of Computer Applications, 68(8): pp. 1-4, 2013.

[40] Ray, S. and De Sarkar, A., "Execution Analysis Of Load Balancing Algorithms In Cloud Computing Environment.". International Journal on Cloud Computing: Services and Architecture (IJCCSA), 2(5): pp. 1-13, 2012.

[41] Hafiz Jabr Younis,"Efficient Load Balancing Algorithm in Cloud Computing.", Supervised By: Dr.Alaa El Halees, A Thesis Submitted as Partial Fulfillment of the Requirements for the Degree of Master in Information Technology, Feb., 2015.

[42] Bernice M. Purcell, "Big data using cloud computing .", Journal of Technology Research , 2013.

[43] Aslam, U., Ullah, I, & Ansara, S. " Open source private cloud computing." Interdisciplinary Journal of Contemporary Research in Business. 2(7), 399-407. 2010.

[44] Dataflair Team, ". Features of Cloud Computing 10 Major Characteristics of Cloud Computing.", https://data-flair.training/blogs/features-of-cloud-computing/. Accessed: 01-10-2019.

[45] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob , Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, Samee Ullah Khan."The rise of “big data” on cloud computing: Review and open research issues." .Information Systems . 2015.

[46] Manoj Muniswamaiah, Tilak Agerwala and Charles Tappert. "Big Data In Cloud Computing Review And Opportunities.", International Journal of Computer Science & Information Technology (IJCSIT) Vol 11, No 4, 2019.

[47] J. DeLayne Stroud. , " UNDERSTANDING THE PURPOSE AND USE OF BENCHMARKING.", https://www.isixsigma.com/methodology/benchmarking/understanding-purpose-and-use-benchmarking/. Accessed:10-10-2019.

[48] Competitive Solutions. http:// csipbl.com/6-benefits-benchmarking/. Accessed: 10-10-2019.

[49] Ana Hobden. "Why Benchmarking Distributed Databases Is So Hard. ", https://hoverbear.org/blog/benchmarking-is-hard/. Accessed:10-10-2019.

[50] George Kousiouris, Andreas Menychtas, Dimosthenis Kyriazis, Theodora Varvarigou. “ Comparison of Database and Workload Types Performance in Cloud Environments”. Conference Paper in Lecture Notes in Computer Science.. DOI: 10.1007/978-3-319-29919-8_11. February 2016.

[51] Yishan Li, Sathiamoorthy Manoharan. “A performance comparison of SQL and NoSQL databases”. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM). DOI: 10.1109/PACRIM.2013.6625441. 2013.

[52] Veronika Abramova, Jorge Bernardino. “NoSQL Databases: MongoDB vs Cassandra”. C3S2E '13 Proceedings of the International Conference on Computer Science and Software Engineering. Pages 14-22. 2013.

[53] Ou, Andrew Yi-Zong and J. Bradley Chen. “Head-to-Head: Which is the Better Cloud Platform for Early Stage Start-up? Docker versus OpenStack.”. 2015.

[54] Prasant Vishwakarma.”Comparitive Performance Analysis of MongoDB and HBase on YCSB”. National College of Ireland. Project: Data Storage and Management. DOI: 10.13140/RG.2.2.15661.95201. December 2018.

[55] Christine Niyizamwiyitira and Lars Lundberg. “Performance Evaluation of Sql And Nosql Database Management Systems In A Cluster.”, International Journal of Database Management Systems (IJDMS) Vol.9, No.6, December 2017.

[56] Hameeza Ahmed, Muhammad Ali Ismail, Muhammad Faraz Hyder, Syed Muhammad Sheraz, Nida Fouq.” Performance Comparison of Spark Clusters Configured Conventionally and a Cloud Service”. Symposium on Data Mining Applications, SDMA, , Riyadh, Saudi Arabia. 30 March, 2016.

[57] Jianxi Yang, Chaoxiao Shen, Yaping Chi, Ping Xu, Wei Sun. “An Extensible Hadoop Framework for Monitoring Performance Metrics and Events of OpenStack Cloud”. IEEE 3rd International Conference on Big Data Analysis. Date of Conference: 9-12 March 2018

[58] Ming Lu, Xu Zhou. “A Big Data on Private Cloud Agile Provisioning Framework Based on OpenStack “.The 3rd IEEE International Conference on Cloud Computing and Big Data Analysis. 2018.

[59] Ioannis Konstantinou and et al . ”TIRAMOLA: Elastic NoSQL Provisioning Through a Cloud Management Platform”. SIGMOD ’12, May 20–24, Scottsdale, Arizona, USA. 2012.

[60] Binod Kumar Adhikari, Wanli Zuo, Ramesh Maharjan .” A performance analysis of OpenStack Cloud vs Real System on Hadoop Clusters”. Date of Conference: 25-30 June 2017.

[61] Abramova V, Bernardino J and Furtado P. " Testing Cloud Benchmark Scalability with Cassandra". In IEEE conference on 10th World Congress on Services. https://doi.10.1109/SERVICES.2014.81. 2014.

[62] Tsuyuzaki, K. & Onizuka, “ MNoSQL Database Characteristics and Benchmark System.”, NTT Technical Review. 2012.

[63] Balaraju.J and P.V.R.D Prasada Rao. “Recent advances in big data storage and Security schemas of HDFS: a survey.Journal of Engineering Technology”. Special Issue (Emerging Trends in Engineering Technology), PP. 132-138. Mar. 2018.

[64] Sung-Soo Kim's Blog," Choosing distribution models.", Stop Thinking, Just Do!, https://sungsoo.github.io/2014/06/07/choosing-distribution-models-mater-slave-versus-peer-to-peer.html, Accessed: 10-01-2020.

[65] Vadym Lobzakov," MongoDB Replica Set with Master-Slave Replication and Automated Failover.", jelastic, November 5, 2019.

[66] Qi, M. “Digital Forensics and NoSQL Databases”. 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD). Xiamen, China: IEEE. 2014.

[67] RIAK KV NoSQL. Retrieved from http://basho.com: http:// basho. com/ products/ riak-kv/ . Accessed: 10-11-2018.

[68] Muhammad, Y. “ Evaluation and Implementation of Distributed NoSQL Database for MMO Gaming Environment.”. Swedish: Uppsala universitet: Institutionen för informations teknologi Department of Information Technology. 2011.

[69] Anasuya N Jadagerimath, P. S. “Efficient IoT Data Management for Cloud Environment using Mongo DB.”. Third International Conference on Current Trends in Engineering Science and Technology ICCTEST. Grenze Scientific Society. January 2017.

[70] Whitepaper: Riak Kv Enterprise Technical Overview. " A Technical Overview Of Riak KV Enterprise .", Basho Technologies. 2016.

[71] MongoDB NoSQL Website, https://www.mongodb.com/. Accessed: 12-01-2018

[72] Cornelia G, Robert G, George P, Andrada O, " A Comparative Study: MongoDB vs. MySQL.". Conference: The 13th International Conference on Engineering of Modern Electric Systems, Oradea. June 2015.

[73] Rubin Porwal, "How does MongoDB store data? ." quora website: https://www.quora.com/How-does-MongoDB-store-data.Sep 2, 2017.

[74] MongoDB Documentation Team, " Structure your Data for MongoDB.", https://docs.mongodb.com/guides/server/introduction/, Accessed: 20-10-2019.

[75] MongoDB Documentation Team, " MongoDB Drivers and ODM . ", https://docs.mongodb.com/ecosystem/drivers/: 2008, Accessed: 20-10-2019

[76] Margaret Rouse, "DigitalOcean.", techtarget: https://searchcloudcomputing.-techtarget.com/definition/DigitalOcean. 2016.

[77] Amine Barkat, Alysson Diniz dos Santos, Thi Thao Nguyen Ho Politecnico di Milano. “OpenStack and CloudStack: Open Source Solutions for Building Public and Private Clouds.”, 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. DOI 10.1109/SYNASC.2014.64. 2014.

[78] Qi, M. “ Digital. Forensics and NoSQL Databases.” 11th. International Conference on Fuzzy Systems and Knowledge Discovery (FSKD). Xiamen, China: IEEE. 2014.

[79] Baji Shaik Avinash Vallarapu. " Google Cloud." , In book: Beginning PostgreSQL on the Cloud:, DOI: 10.1007/978-1-4842-3447-1_5.March 2018.

[80] Nicholas J. MitchellKazi Zunnurhain. " Google cloud platform Security.", Conference: the 4th ACM/IEEE Symposium, DOI: 10.1145/3318216.3363371, November 2019.

[81] Conner Forrest, " Google Cloud Platform: A cheat sheet.".techrepublic.: https:// www.techrepublic.com/article/google-cloud-platform-the-smart-persons-guide/April 15, 2019.

[82] Google cloud, "Compute Engine documentation.", https://cloud.google.com/compute/docs/. Accessed : 10-09-2019.

[83] Sphe Malgas, " Cloud Architect: How to build architectural diagrams of Google Cloud Platform(GCP) .", medium website, Feb 8, 2018.

[84] MongoDB NoSQL, "NoSQL Performance Benchmarks.", https://www.mongodb.- com/scale/nosql-performance-benchmarks. Accessed 03-02-2019.

[85] KVA DAV, "An Empirical Study on Performance Evaluation of NoSQL Databases.", International Journal of Electronics Engineering Volume 10 • Issue 1 pp. 235-244 Jan 2018-June 2018.

[86] Basho bench :https://github.com/basho/basho_bench. Accessed: 09-10- 2018.

[87] Amir Ghaffari, N. C. Scalable persistent storage for Erlang: theory and practice. Proceedings of the twelfth ACM SIGPLAN workshop on Erlang. Boston, Massachusetts, USA —: ACM. PP, 73-74. http://DOI: 10.1145/2505305. 2505315. September 28. 2013.

[88] Gerard Haughian,” Benchmarking Replication in NoSQL Data Stores ”, Submitted in partial fulfilment of the requirements for the MILLISECONDSc Degree in Computing Science of Imperial College London. September 2014.

[89] Surya Narayanan Swaminathan,”Quantitative Qnalysis of Scalable NoSQL Databases”, Master of Science in computer science the university of texas at arlington. December 2015.

[90] Jérôme Darmont, "Data Processing Benchmarks", Université de Lyon (Laboratoire ERIC), France, 2017.

[91] N. S. Zhivchikova, Y. V. Shevchuk."Riak KV performance in sensor data storage application.", Program Systems: Theory and Applications, no.3(34), pp. 61–85. 2017.

[92] Zatrochova, Zuzana. "Analysis and testing of distributed NoSQL datastore Riak.", 2015.

[93] Hendawi, Abdeltawab, et al. Benchmarking large-scale data management for the Internet of Things. The Journal of Supercomputing, 75.12: 8207-8230, 2019.

[94] Introduction To Nosql Databases, (https://holowczak.com/nosql-key-value-column-stores/3/), 2020.

[95] Basho docs, Cluster Capacity Planning. https://riak.docs.hw.ag/riak/kv/latest/setup/planning/cluster-capacity/, 2020 (accessed 27 May 2020).

[96] QuABaseBD, Riak Scalability Features. https://quabase.sei.cmu.edu/mediawiki/index.php/Riak_Scalability_Feat ures, 2020 (accessed 17 May 2020).

[97] Riak, Massive Scalability. https://riak.com/products/riak-kv/massive scalability/index.html?p=10912.html, 2020 (accessed 10 April 2020).

[98] Kaushal Patel, Mongodb Schema Design, (http://pingax.com/mongodb-schema-design/), MAY 18, 2014.

[99] Dipina Damodaran, B., Shirin Salim, and Surekha Marium Vargese. "Performance evaluation of MySQL and MongoDB databases." International Journal on Cybernetics & Informatics (IJCI) Vol 5 (2016).

APPENDICES

Appendix A: Install and configuration Clustering in Riak KV

Download and add the key GPG using the following command:

```
curl https://  packagecloud.io/gpg.key | apt-key add –
```

Next, add the repository by editing /etc/aptt/sources.listt file with the nano /etc /apt / sources.list, command. Then, add the following line:

```
deb https://pacikagecloud.io/basho/ riak/ubuntu/ xeniel main

deb src https://pacikagecloud.io/basho/riak/ubuntu/ xeniel
main
```

Save and close the file when you're done. Then, update the repository and install Riak KV, using the following command:

```
apt-get update -y
apt-get install riak -y
```

On the first node: open /etc /riak /riak.conf file with the nano / etc/ riak/ riak. conf command. Next, make the following settings:

```
nodename-= riak@192.168.99.100
erlang.schedulers.force_wakeup-interval-= 500
erlang.schedulers.compaction-of_load-= false
ring_size-= 64
listener.http.internal-= 192.168.0.105:8098
listener.protobuf.internal-= 192.168.0.105:8087
```

Next, start Riak service with the riak start command. On the second node: open /etc /riak /riak. conf file with the nano /etc /riak /riak. conf command. Then, make the following settings:

```
“nodename = riak@192.168.99.101”
“erlang.schedulers.force_wakeup interval = 500”
“erlang.schedulers.compaction_of_load = false”
“ring_siez = 64”
“listener.http.internal = 192.168.99.101:8098”
“listener.protobuf.internal = 192.168.99.101:8087”
```

Save and close the file. Then, start Riak service with the Riak start command, and repeat these steps for the cluster nodes. The Riak KV basic setup is now complete. Now you need to set up the cluster by linking nodes-2 and-3 to node 1. To do this, on node-2, run the next command to link node-2 to node 1:

```
“riak-admin cluster join riak@192.168.99.100”
```

The output will be as follows:

```
“Success: staged join request for 'riak@192.168.99.101' to
'riak@192.168.99.100'”
```

Next, on node-3 run the next command to join node-3 to the node-1:

```
“riak-admin cluster join riak@192.168.99.100”
```

The output will be as so:

```
“Success: staged join request for 'riak@192.168.99.102' to
'riak@192.168.99.100'”
```

```
“riak-admin cluster join riak@192.168.99.100”
```

The output will be as so:

```
“Success: staged join request for 'riak@192.168.99.102' to
'riak@192.168.99.100'”
```

Following, check the cluster status through running the next command on the node 1:
riak-admin cluster status

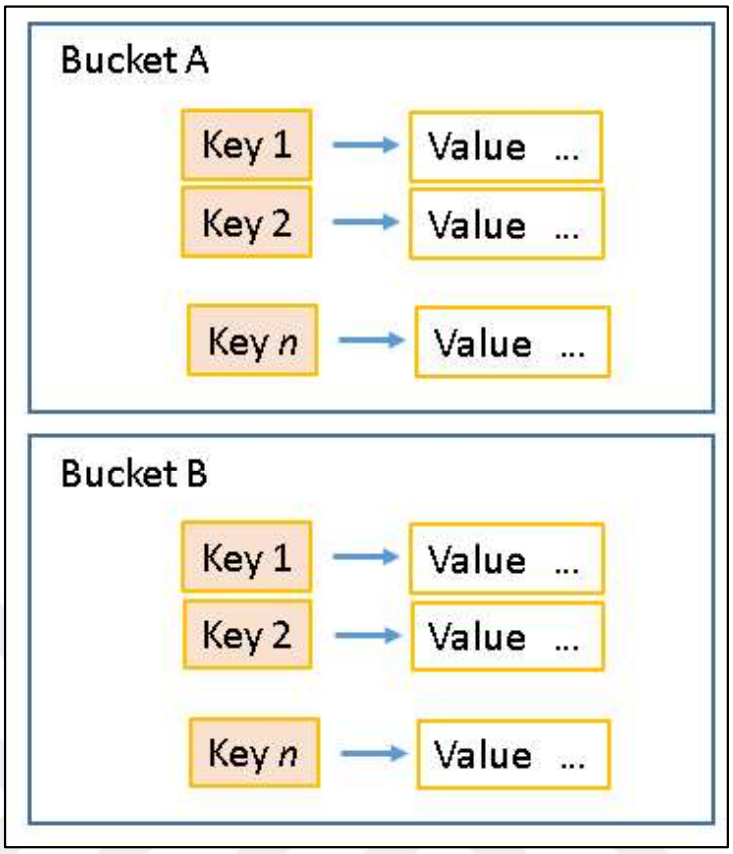
```
---- Cluster Status ----
Ring ready: true

+------------------------+------+-------+-----+-------+
|          node          |status| avail |ring |pending|
+------------------------+------+-------+-----+-------+
| (C) riak@192.168.99.100 |valid |  up   | 62.5|  34.4 |
|     riak@192.168.99.101 |valid |  up   | 20.3|  32.8 |
|     riak@192.168.99.102 |valid |  up   | 17.2|  32.8 |
+------------------------+------+-------+-----+-------+

Key: (C) = Claimant; availability marked with '!' is
unexpected
```

Appendix B: Riak KV architecture

As a distributed database, the goal of Riak is producing a maximum level of data availability via the distribution of data among many different servers. If a Riak client has access to a Riak server, then it should have the ability to successfully write data. Riak may be utilized as a compatible system, as the data that the Riak users or clients want to be able to read need to remain generally accessible even in potential cases of failure. This, however, might not necessarily be the data's most recently updated version [91,12]. All of the nodes in a Riak cluster are similar to each other, all containing a copy of the whole Riak software package that is both independent and complete. This architecture does not contain master nodes; no individual node will be assuming greater responsibility compared to the rest of the nodes, and, furthermore, no individual node will be assigned unique or special tasks that the rest of the nodes will not also be performing. Thanks to this uniformity, the scalability and the fault tolerance of Riak are both ensured [92, 93]. However, Riak is not a suitable choice for data storage projects that are extremely centralized, with unchanging or fixed data structures, and the use of Riak should be avoided in such cases. In general, being a very scalable, distributed, and fault-tolerant NoSQL database with the capability of storing map/reduce, JSON, HTTP, and REST queries, Riak can be said to be an ideal choice for various web applications.

Data Model: Riak will store data (keys, values) in buckets.

- **Key:** Binary values will be used to identify objects.
- **Value:** Values include binaries, strings, objects, numbers, etc.
- **Bucket:** A bucket is the virtual space for the storage of Riak objects.



Appendix Figure 1 Data model: Riak [93]

Each node in the ring (also called physical nodes) runs a set number of virtual nodes (Vnodes). Each of these Vnodes occupies one partition in the ring. The partition size of the ring is defined when configuring Riak or when the cluster is initialized [68]. In order to understand more clearly how Riak forms a ring cluster, consider that the partition size of the ring is 32, as shown in Figure 2. The number of Vnodes is calculated as follows:

Vnodes = (number of partitions) / (number of nodes)

Each node in the cluster will be responsible for 1 / (total number of physical nodes); that is, if we have 4 nodes, each node is responsible for 8 partitions in a ring, as shown in Figure 2.

As a result, the following points can be concluded:

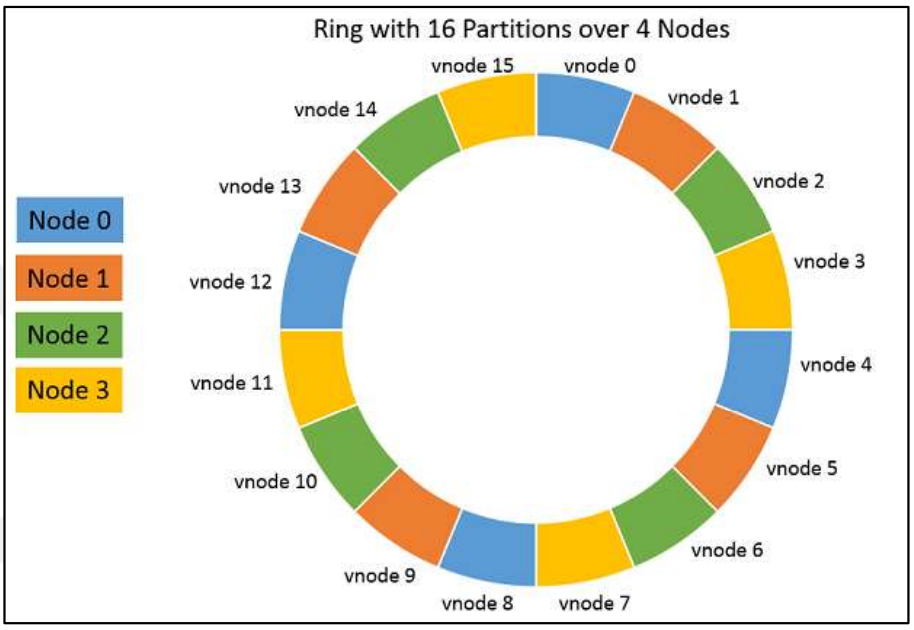
Without a master node, all nodes in Riak have the same purpose.

Riak uses compatible hashing to share data across the cluster.

Each node in the cluster is able to satisfy every client request.

In addition, Riak clusters can shrink and expand dynamically; in other words, they can add or remove any node from the cluster. The cluster expands on its own to distribute data accordingly [25].

The ring size is configured before starting the cluster and is already set in the configuration file. This applies to smaller clusters, but if it is planned to expand the cluster to more than 5 nodes, a larger ring size is recommended [94].



Appendix Figure 2. The architecture of the 4 nodes in the Riak cluster [94]

The following elements of the Riak architecture impact scalability:

- Improving the hardware and adding more nodes impacts scalability.
- Riak KV is designed to solve data availability problems easily with ease of scale.
- Riak KV may increase capacity on demand without data sharing or manual reorganization of the cluster.
- Riak KV solves problems by using commodity hardware to scale horizontally and eradicates the need for sharding by growing and shrinking the cluster elastically while balancing the load on every node.
- Riak KV, with a foundation for scalability and fault tolerance, has a structure that provides replication of the nodes [95].

- Scaling the database necessitates horizontally expanding the database content across multiple nodes. Only a complete copy of the database can be replicated to multiple nodes in some situations, which curbs the scalability to the capacity of one node. This is effectively scaling up. In Riak, the following options are supported: replication and horizontal partitioning [96].
- When a new capacity is added to the Riak database, the performance will be improved linearly. When new nodes are added or removed to balance data across nodes, Riak KV definitely redistributes data across the cluster. The framework continues to measure the performance as long as nodes are being removed or added for the database [97].

Appendix C: Install and configuration clustering in MongoDB

On the terminal, begin the next command to import the MongoDB public GPG key from it.

```
“wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc
| sudo apt-key add –“
```

The action should respond with the "OK."
Nevertheless, if you receive an error showing that gnupg is not installed, you can:
Use the following command to install the gnupg and its required libraries:

```
“sudo apt-get install gnupg”
```

Once installed, repeat introducing the key:

```
“wget -qO - https://www.mongodb.org/ static/pgp/server-4.2.asc
| sudo apt-key add –“
```

Create a /etc /apt /sources.list.d /mongodb-enterprise.list file for MongoDB.

```
“echo "deb [ arch=amd64,arm64,s390x ]
http://repo.mongodb.com/ apt/ubuntu xenial/mongodb
enterprise/4.2 multiverse" | sudo tee /etc/apt/
sources.list.d /mongodb-enterprise.list ”
```

Reload the local package and install the MongoDB package database.
You can install the specific version of MongoDB or latest stable version of MongoDB.

```
sudo apt-get update
sudo apt-get install -y mongodb-org
```

The above steps will help you to install MongoDB on an Ubuntu nodes, you will require to follow the same method in 3 nodes.

MongoDB-Primary IP(Mongo1) = 192.168.99.100
MongoDB-Secondary IP(Mongo2) = 192.168.99.101
MongoDB-Third IP(Mongo3) = 192.168.99.102

Initialize the replica set with the following command

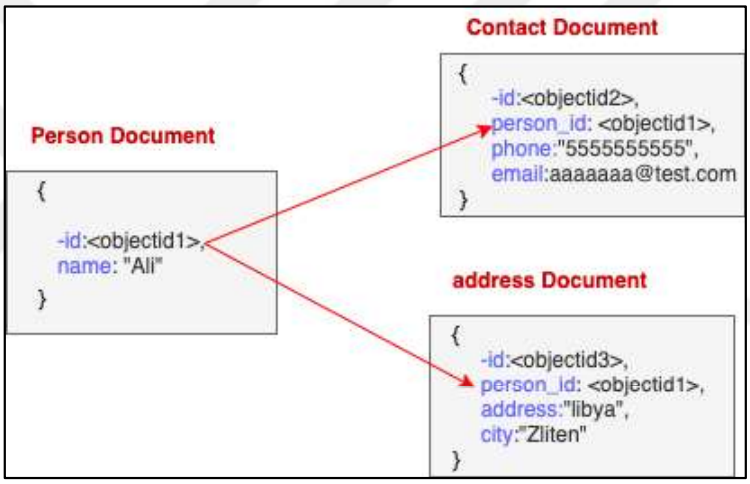
```
"rs.initiate()
rs.add("ip address of 2nd MongoDB server")
rs.add("ip address of 3rd MongoDB server")"
```

In every of the three database server nodes, connect to mongo shell and begin rs.status() to see the real value (One of the nodes should show as Primary and other two nodes should show as Secondary)

Appendix D: MongoDB architecture

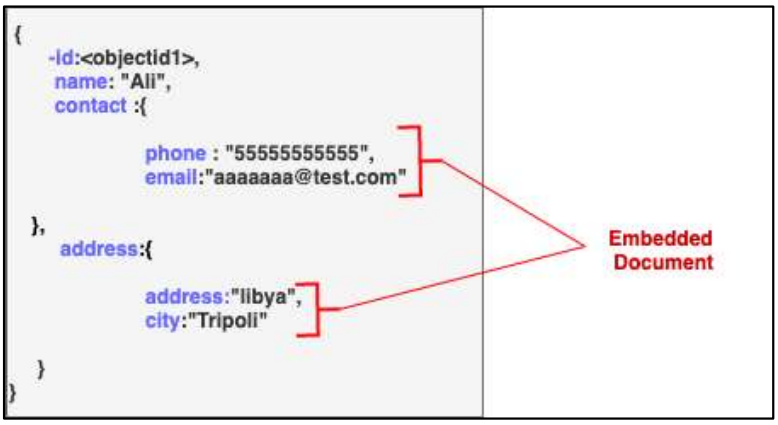
As it stores data in the form of BSON (Binary JavaScript Object Notation) documents, MongoDB does not support joins and it is a schema-less database, but it has other organizational tools for achieving those goals. These are known as linking (also known as “References”) and embedding documents.

- **Linking:** Linking creates relationships between data by including field references from one document to another. Links are a standardized form of data. Linked documents can add some custom codes to handle the data in the application, a useful asset to be considered when needed.



Appendix Figure 3. Linking in MongoDB

- **Embedding Documents:** In a single document called an embedded document, this technique can be used to structure data in subdocuments or arrays. This method will enable the application to manipulate or access data in a single database operation. The embedded document is in a denormalized form [98].

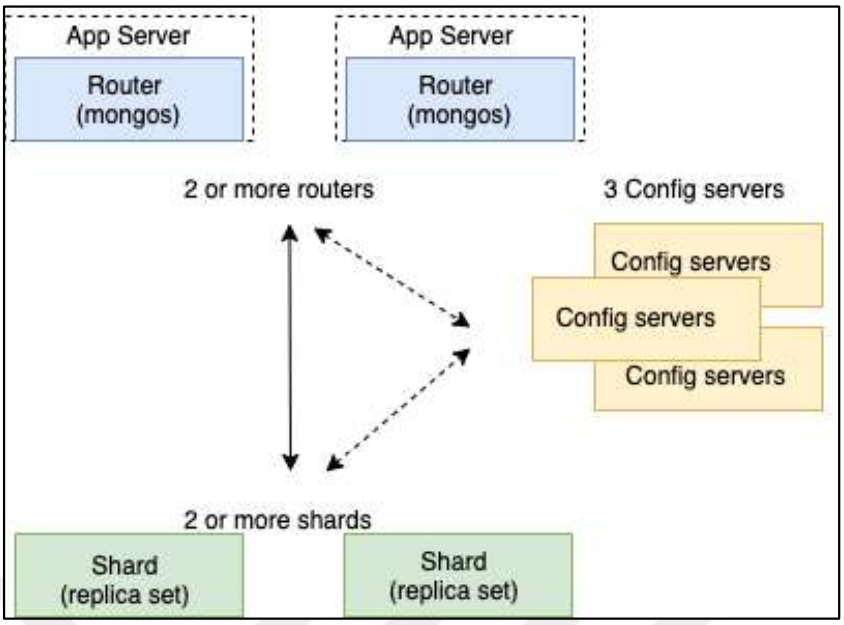


Appendix Figure 4. Embedding documents in MongoDB

The architecture of MongoDB supports single or independent instance operations. Replica sets produce high-performance replications by automatic failure handling, while sharded clusters can divide large datasets on different machines that are transparent to users. MongoDB users combine replica sets and sharded clusters to provide high levels of redundancy of datasets, which are transparent for applications [99]. A MongoDB cluster has the following components: shards, query routers, and config servers. Shards provide high availability and data consistency; in a production sharded cluster, each shard is a replica set.

It is possible to query the interface between the router and the client application and direct the operation to the appropriate fragment or fragments. It is also possible to query the router to process the transaction and assign the target operation to the shard, and then the result is returned to the client. A sharded cluster can contain multiple query routers to divide the client request load. The client sends the request to a query router. Most sharded clusters have many query routers.

Configuration servers store cluster metadata, which is data that contains the mapping of cluster data into shards. The query prompt uses metadata to target operations to specific portions. Homogeneous production clusters contain exactly three servers configured.



Appendix Figure 5. Sharding in MongoDB