Comparison between Couchbase and ScyllaDB Key-Value NoSQL stores

Rallakis
Electrical and Computer Engineering
National Technical University of Athens
el18150@mail.ntua.gr

Vasios
Electrical and Computer Engineering
National Technical University of Athens
el19404@mail.ntua.gr

Vatistas
Electrical and Computer Engineering
National Technical University of Athens
el18020@mail.ntua.gr

***Abstract*—The project objective is to evaluate the performance of two prominent Key-Value NoSQL Databases ScyllaDB and Couchbase by performing benchmarking techniques. More specifically, we will analyze the installation and configuration process of the two databases, we will describe how we managed to generate data and load them into the databases, we will evaluate the performance of them in various cases and we will quote performance metrics for direct comparison.**

I. Theory

A. Big Data

Big data refers to the massive volume of structured, semi-structured, and unstructured data that is generated by individuals, businesses, and other organizations on a daily basis. With the widespread use of digital devices, social media, and other sources of information, the amount of data being produced is growing exponentially. This data can include everything from online purchases and social media interactions to scientific research data and medical records [1].

The challenge with big data is not just collecting it, but also processing and analyzing it in a way that is meaningful and actionable. Traditional data processing tools and techniques are not always sufficient for handling the sheer volume and complexity of big data, which is why new technologies and methods have emerged to address these challenges.

Big data has enormous potential to drive innovation and improve decision-making across industries, from healthcare and finance to retail and manufacturing. However, to fully realize its benefits, organizations must have the right infrastructure, tools, and expertise to manage and make sense of this data.

B. NoSQL Key/Value Stores Databases

NoSQL key-value stores databases are a type of non-relational database that store and retrieve data using a simple key-value pair. In these databases, data is not stored in tables with predefined schemas like traditional relational databases. Instead, each data item is stored as a key-value pair, where the key is a unique identifier for the data and the value is the data itself. This flexible data model allows for efficient storage and retrieval of large amounts of data with low latency.

NoSQL key-value store databases are often used in applications where speed and scalability are critical, such as real-time analytics, e-commerce, and content management. They are also well-suited for use cases where data structures are not well-defined or may change frequently.

One of the key advantages of NoSQL key-value store databases is their ability to handle large volumes of data and scale horizontally across multiple servers. They are also designed to be highly available and fault-tolerant, with built-in mechanisms for replication and data distribution. However, they can be more difficult to use than traditional relational databases due to their lack of a standard query language and their reliance on custom APIs and data models. As a result, they require specialized expertise to implement and manage effectively.

Some of their advantages are their speed, scalability, simplicity and the ability to distribute horizontally. Their shortcomings are that many data structures can't be easily modeled as key value pairs.

In our case we are utilizing ScyllaDB and Couchbase NoSQL databases.

II. Concepts

We provide important concepts of modern cluster architecture.

A. CAP Theorem

In a distributed system we are not able to satisfy all of the three important parameters which are Consistency, Availability and Partition-tolerance. We can have at most two of them satisfied simultaneously.

B. Node

It consists of the database server software running on a computer server. All nodes work together to provide service even if one node becomes unavailable due to some failure.

C. Keyspace

Keyspace is a collection of tables with attributes that define how data is replicated across nodes.

D. Table

Table is how data is stored and can be thought of as a set of rows and columns.

E. Cluster

A Cluster is a collection of nodes to store the data. The nodes are logically distributed like a ring. A minimum cluster typically consists of at least three nodes. Data is automatically replicated across the cluster, depending on the Replication Factor.

F. Replication Factor

Replication Factor defines the number of copies of the data to ensure no single point of failure. This means that even if one node goes down, the data will still be available. It ensures reliability and fault tolerance

G. Partition

Partition is a subset of data that is stored on a node and replicated across nodes. Each partition is identified by a partition key.

H. Partitioner (Partition Hash function)

Determines where data is stored on a given node in the cluster. It does this by computing a token for each partition key.

I. Consistency Level

The number of nodes that must acknowledge a read or write operation before it is considered successful. The most common one is the QUORUM.

III. Infrastructure and Software

For the purposes of our project we requested resources from **Okeanos, GRNET** which delivers production–quality IaaS to the Greek Academic and Research Community.

A. Virtual Machines

We managed to obtain three Virtual Machines in total with the following system specifications.

- Operational System: CentOS 7
- Architecture: x86_64
- Processor: Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
- Logical Cores per VM: 4 in total
- Memory: 8GB
- Disk: 32GB Virtual I/O device
- Network: Download: 900Mbps, Upload: 900Mbps

We communicated through the bash shell using the SSH protocol.

B. Operational System

For the purposes of our project we set up the operational system CentOS 7. The CentOS Linux distribution is a stable, predictable, manageable and reproducible platform derived from the sources of Red Hat Enterprise Linux (RHEL).

The reason we selected CentOS as our working platform is because it is both supported by ScyllaDB and Couchbase.

C. YCSB

We installed the YCSB benchmark software [2] which provides powerful tools to evaluate the performance of different "key-value" stores. The goal of the Yahoo Cloud Serving Benchmark (YCSB) project is to develop a framework and common set of workloads for evaluating the performance of different "key-value" and "cloud" serving stores. The project comprises two areas:

- The YCSB Client, an extensible workload generator.
- The Core workloads, a set of workload scenarios to be executed by the generator.

Installation proccess

```
$ git clone git://github.com/brianfrankcooper/
    YCSB.git
$ cd YCSB
$ mvn clean package
```

D. Maven 3

The installation of the YCSB software required the use of the Maven tool. YCSB requires the use of Maven 3.

Maven is a build automation and dependency management tool for Java projects. It provides a set of conventions and plugins that help to manage the build process of Java applications, including compiling the source code, creating executable JAR files, running unit tests, and managing project dependencies.

Maven uses a declarative XML file called the "pom.xml" file (Project Object Model) to define the project's configuration and dependencies. The POM file describes the project's structure, dependencies, plugins, and other settings that are required for building the project.

Installation proccess

```
$ sudo yum install maven
$ sudo yum autoremove
Detected Maven Version: 3.0.5 is not in the
    allowed range [3.1.0,3.6.2),(3.6.2,).
$ sudo yum install wget
$ wget https://dlcdn.apache.org/maven/maven
    -3/3.6.3/binaries/apache-maven-3.6.3-bin.
    tar.gz
$ sudo tar xf /tmp/apache-maven-3.6.3-bin.tar.
    gz -C /opt
```

E. OpenJDK

For the purposes of our project we installed OpenJDK java 8 and then we set up Java's path variables.

OpenJDK is an open-source implementation of the Java Platform, Standard Edition (Java SE). OpenJDK includes a Java Development Kit (JDK), which includes the Java Runtime Environment (JRE), the Java Compiler, and other tools for developing Java applications. It supports a wide range of platforms, including Windows, Linux, and macOS.

Installation proccess

```
$ yum -y update
$ yum install java-1.8.0-openjdk
$ update-alternatives --config java
(/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.362.b08
    -1.el7_9.x86_64/jre/bin/java)
```

```
$ vim .bash_profile
$ export JAVA_HOME=/usr/lib/jvm/java-1.8.0-
    openjdk-1.8.0.362.b08-1.el7_9.x86_64/jre/
    bin/java
```

F. CQLSH

Cqlsh (Cassandra Query Language shell) is a command-line tool for interaction with the Apache Cassandra database using the Cassandra Query Language (CQL). CQL is a SQL-like language that is used to query and manipulate data in Cassandra.

We utilized the cqlsh script for the communication with the ScyllaDB Cluster. In order to properly install it we had to install python, python-p, epel-release and makecache.

Installation proccess

```
$ yum makecache
$ yum install epel-release
$ yum install python-pip
$ yum install python34-pip
$ pip3 install cqlsh
```

IV. Database System Setup

A. ScyllaDB

After the aforementioned software installments we proceeded with the installation of the ScyllaDB [3].

1) System Requirements: Firstly, we checked the system requirements in order to confirm the system's ability to run the database. All the provided information is gathered from the official website.

- OS Support: ScyllaDB Open Source supports x86_64 for all versions
- CPU: any number will work since Scylla scales up with the number of cores
- Memory: 2GB per logical core
- Disk Space: Adequate amount of free disk space available on a node

2) Installation Process: Following the commands below.

```
$ epel-release
$ sudo curl -o /etc/yum.repos.d/scylla.repo -L
    http://downloads.scylladb.com/rpm/centos/
    scylla-5.1.repo
$ sudo yum install scylla
```

3) Configuration: Fo the first node we work on we change config file scylla.yaml parameters:

- cluster_name: Test Cluster
- seeds: 192.168.0.1
- listen_address: 192.168.0.1
- rpc_address: 192.168.0.1

Afterwards, we proceeded with opening the following ports for the proper communication between nodes and between the client and node:

9042 - CQL (native_transport_port) - TCP
9142 - SSL CQL (secure client to node) - TCP
7000 - Inter-node communication (RPC) - TCP
7001 - SSL inter-node communication (RPC) - TCP
7199 - JMX management - TCP
10000 - Scylla REST API - TCP
9180 - Prometheus API - TCP
9100 - node_exporter - TCP
9160 - Scylla client port (Thrift) - TCP
19042 - Native shard-aware transport port - TCP
19142 - Native shard-aware transport port (ssl) - TCP

4) Run Database: After the above steps we proceed with starting the scylla server. We check that everything runs correctly with the help of nodetool and cqlsh.

```
$ systemctl start scylla-server
$ nodetool status
$ cqlsh
```

5) Cluster Creation: For the creation of the cluster we completed all the aforementioned steps regarding the installation of the ScyllaDB in the three available machines. The difference is in the IP addresses that add in the scylla.yaml configuration file. More specifically, we change the seeds to match the master's node IP and rpc_address, listen_address to match each other's node IP. After that we have a database cluster with three nodes and we can verify that using the nodetool status command. Below we can see the output in the shell:

```
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address      Load       Tokens       Owns    Host ID                               Rack
UN  192.168.0.1  2.32 MB    256          ?       51ef5f80-643b-4eb7-aec4-9d7979367140  rack1
UN  192.168.0.2  2.59 MB    256          ?       e22dc54e-8c9c-4f96-8a5f-9a4225876d3c  rack1
UN  192.168.0.3  ?          256          ?       54c577b2-73cd-45cf-8019-2df5e7158c7e  rack1
```

Fig. 1. 3 scylla nodes appearing as up and normal.

At that stage the ScyllaDB cluster is ready to execute operations.

B. Couchbase

1) System Requirements: Firstly, we checked the system requirements in order to confirm the system's ability to run the database. All the provided information is gathered from the official website of Couchbase project [4].

- OS Support: CentOS 7.x (Deprecated)
- CPU: 2 GHz dual core x86_64 CPU
- Memory: 4 GB (physical)
- Disk: 8 GB

2) Installation Process: Following the commands below.

```
$ curl -O https://packages.couchbase.com/
    releases/couchbase-release/couchbase-
    release-1.0-x86_64.rpm
$ sudo rpm -i ./couchbase-release-1.0-x86\_64.
    rpm
$ sudo yum install couchbase-server-community
    -7.1.1-3175
```

3) Configuration: Couchbase Server uses multiple TCP ports to facilitate communication between server components, as well as with Couchbase clients. These ports must be open for Couchbase Server to operate correctly.

Below are the necessary ports:
Unencrypted: 9119, 9998, 11213, 21200, 21300
Unencrypted: 4369, 8091-8094, 9100-9105, 9110-9118, 9120-9122, 9130, 9999, 11209-11210, 21100
Unencrypted: 8091-8097, 9123, 9140 [3], 11210, 11280

Command to open ports:

```
$ firewall-cmd --add-port=4369/tcp --permanent
    &
firewall-cmd --add-port=8091-8097/tcp --
    permanent &
firewall-cmd --add-port=9100-9105/tcp --
    permanent &
firewall-cmd --add-port=9110-9123/tcp --
    permanent &
firewall-cmd --add-port=9130/tcp --permanent &
firewall-cmd --add-port=9140/tcp --permanent &
firewall-cmd --add-port=9998/tcp --permanent &
firewall-cmd --add-port=9999/tcp --permanent &
firewall-cmd --add-port=11213/tcp --permanent
    &
firewall-cmd --add-port=11209/tcp --permanent
    &
firewall-cmd --add-port=11210/tcp --permanent
    &
firewall-cmd --add-port=11280/tcp --permanent
    &
firewall-cmd --add-port=21100/tcp --permanent
    &
firewall-cmd --add-port=21200/tcp --permanent
    &
firewall-cmd --add-port=21300/tcp --permanent
```

4) Run Database: After the above steps we proceed with starting the couchbase server.

```
$ systemctl start couchbase-server
```

5) Remote Desktop: In order to be able to view the server's settings and cluster over the network we use the Couchbase Server Web Console through a windows remote desktop. To do that we proceed with the following bash commands:

```
$ sudo yum -y update
$ sudo yum install -y epel-release
$ sudo yum install -y xrdp
$ sudo systemctl enable xrdp
$ sudo systemctl start xrdp
$ sudo firewall-cmd --add-port=3389/tcp --
    permanent
$ sudo firewall-cmd --reload
$ sudo yum install -y epel-release
$ sudo yum groupinstall -y "Xfce"
$ sudo reboot
```

Reconnect and then run:

```
$ echo "xfce4-session" > ~/.Xclients
```

After that we install google chrome on CentOS:

```
$ wget https://dl.google.com/linux/direct/
    google-chrome-stable\_current\_x86\_64.rpm
$ sudo yum install ./google-chrome-
    stable_current\_*.rpm
$ google-chrome --no-sandbox
```

At this point we are ready to access the web console through the localhost address with port 8091. The initial screen is the following:

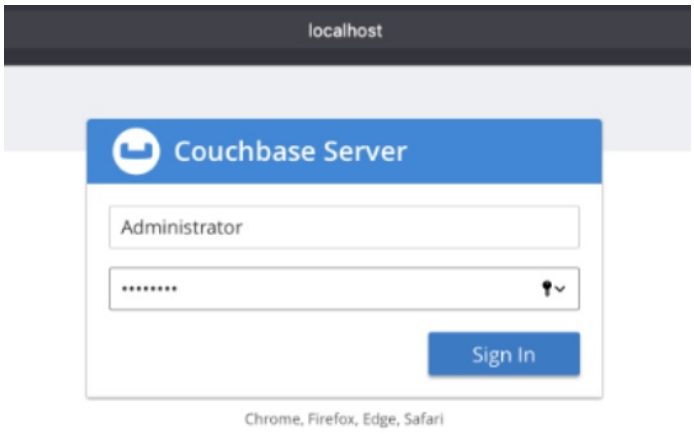

Fig. 2. Couchbase server UI.

6) Cluster Creation: Cluster setup takes place on the web console in which we make all the necessary configurations. We set the following parameters: Cluster Name, Admin Username, Password. After that we agree with the terms and conditions and we configure the couchbase server:

- hostname: 127.0.0.1
- IP Family Preference: IPv4
- Service Memory Quotas: Default values
- Data Disk Path: Default
- Indexes Disk Path: Default
- Eventing Disk Path: Default
- Analytics Disk Paths: Default

After that we proceed by provisioning the coordinator node with the help of the couchbase-cli command.

```
$ couchbase-cli cluster-init -c 192.168.0.1 \
--cluster-username admin \
--cluster-password password \
--services data,index,query \
--cluster-ramsize 4096 \
--cluster-index-ramsize 256
```

After that we initialize the other nodes:

```
$ /opt/couchbase/bin/couchbase-cli node-init -
    c 192.168.0.1 \
-u admin -p password \
--node-init-data-path /opt/couchbase/var/lib/
    couchbase/data \
--node-init-index-path /opt/couchbase/var/lib/
    couchbase/data \
--node-init-eventing-path /opt/couchbase/var/
    lib/couchbase/data \
--node-init-analytics-path /opt/couchbase/var/
    lib/couchbase/data \
--node-init-java-home /opt/couchbase/bin/java
    \
```

```
--node-init-hostname 127.0.0.1 \
--ipv4
```

At that stage the Couchbase cluster is ready to execute operations.

V. Benchmarking

YCSB Utilization The tool that we use for the benchmarking is the YSCB. For the successful utilization of it we had to proceed with the following steps:
1) Set up the database system.
We have already implemented this.
2) Choose the appropriate DB interface layer.
In our case we use the scylla and couchbase interface layer with the commands load and run.
3) Choose the appropriate workload.
For the purposes of our benchmark we chose to proceed with the following workloads:

- Workload A: This is a combination of read/write in a 50/50 manner.
- Workload C: Read only workload with 100% read operations.

4) Choose the appropriate runtime parameters
- threads: number of client threads. For our operations we set number of threads to be 84
- target: number of operations per second. For our operations we set the target number to be 10.000. We set this number because it is a capacity which can be handled by the machines.
-s: status of the running workload every 10 seconds.

5) Load the Data
Loading data is the process of inserting the data into the database system. It is executed with the command load. Parameters:
-P: load property files
-p: parameters like recordcount, fieldcount (in our case 10), fieldlength (in our case 128), insertstart=0, insertcount, cassandra.username and cassandra.passward.

6) Run the data. Running data is the transaction process in which defines the operations to be executed against the data set.

We describe the loading and running steps in detail for both ScyllaDB and Couchbase.

A. ScyllaDB

1) scylla_setup_rf1.cql: In order to be able to quickly setup the database we made a script that creates a keyspace named YCSB with the desired replication factor. It can be viewed below:

```
create keyspace YCSB WITH REPLICATION = \{'
    class' : 'SimpleStrategy', 'replication\
    _factor': 1 \} ;
use YCSB;
create table usertable (y\_id varchar primary
    key, field0 varchar, field1 varchar,
    field2 varchar, field3 varchar, field4
    varchar, field5 varchar, field6 varchar,
    field7 varchar, field8 varchar, field9
    varchar);
```

2) scylla_cleanup.cql: We also made another script that deletes that keyspace YCSB.

```
drop keyspace ycsb;
```

At this stage can execute all the necessary benchmarking workloads by running a bash script that includes the aforementioned and also the basic ycsb commands for load and runs.

3) scylla_script.bash: We also made another script that deletes that keyspace YCSB.

```
#!/bin/bash


USER="cassandra"
PASSWORD="cassandra"
THREADS="10"


WORKLOADS=("workloada" "workloadb" "workloadc")
REPLICATION_FACTORS=("3")


for replication_factor in "${REPLICATION_FACTORS[@]}
    "
do
    for workload in "${WORKLOADS[@]}"
    do
        cqlsh 192.168.0.1 -f "scylla_setup_rf${
            replication_factor}.cql"


        bin/ycsb load scylla -s -P "workloads/
            $workload" \
            -threads "$THREADS" \
            -p "cassandra.username=$USER" \
            -p "cassandra.password=$PASSWORD" \
            -p scylla.hosts
                =192.168.0.1,192.168.0.2,192.168.0.3
                 >> "scylla_outputs/output_load_${
                workload}_rf${replication_factor}.
                txt"


        bin/ycsb run scylla -s -P "workloads/
            $workload" \
            -threads "$THREADS" \
            -p "cassandra.username=$USER" \
            -p "cassandra.password=$PASSWORD" \
            -p scylla.hosts
                =192.168.0.1,192.168.0.2,192.168.0.3
                 >> "scylla_outputs/output_run_${
                workload}_rf${replication_factor}.
                txt"
        cqlsh 192.168.0.1 -f scylla_cleanup.cql
    done
done
```

B. Couchbase

For the purposes of the loading and running the operations we created the following bash script.

```
#!/bin/bash


HOST="localhost"
PORT="8091"
USER="admin"
PASSWORD="password"
RAM="4487"
```

```
BUCKET="default"


WORKLOADS=("workloada" "workloadb" "workloadc")
REPLICATION_FACTORS=("1" "2" "3")


for replication_factor in "${REPLICATION_FACTORS[@]}
    "
do
    for workload in "${WORKLOADS[@]}"
    do
        /opt/couchbase/bin/couchbase-cli bucket-
            create -c localhost:8091 \
            -u $USER -p $PASSWORD --bucket=$BUCKET \
            --bucket-type=couchbase --bucket-ramsize
                =$RAM --bucket-replica=
                $replication_factor

        bin/ycsb load couchbase2 -s -P "workloads/
            $workload" \
            -p "couchbase.host=$HOST" -p "couchbase.
                port=$PORT" \
            -p "couchbase.user=$USER" -p "couchbase.
                password=$PASSWORD" \
            -p "couchbase.bucket=$BUCKET" -threads
                84 >> "outputs/output_load_${
                workload}_rf${replication_factor}.
                txt"


        bin/ycsb run couchbase2 -s -P "workloads/
            $workload" \
            -p "couchbase.host=$HOST" -p "couchbase.
                port=$PORT" \
            -p "couchbase.user=$USER" -p "couchbase.
                password=$PASSWORD" \
            -p "couchbase.bucket=$BUCKET" -threads
                84 >> "outputs/output_run_${workload
                }_rf${replication_factor}.txt"


        /opt/couchbase/bin/couchbase-cli bucket-
            delete -c localhost:8091 \
            -u $USER -p $PASSWORD --bucket=$BUCKET
    done
done
```

VI. Benchmarking Parameters

After completing the aforementioned steps we are ready to proceed with the benchmark of the two databases.

1) OLTP Workload: A transactional or OLTP (online transaction processing) workload is a workload typically identified by a database receiving both requests for data and multiple changes to this data from a number of users over time where these modifications are called transactions.

2) Throughput (Load): The number of incoming requests per unit of time. Latency: The round trip time for a request to be processed.

The ultimate goal of an OLTP database performance test is to find out what the latencies of requests are for various throughput rates.

3) Percentile: Latency distribution that describes how many requests were worse than some specific latency target. For example, 99th percentile means that 99% of the requests should be faster than given latency.

4) Utilization: Maximum load that a system can handle.

It's enough to check the optimal point when the throughput is at the maximum point for a given latency goal.

VII. Benchmarking Diagrams

ScyllaDB VS Couchbase benchmarking results:

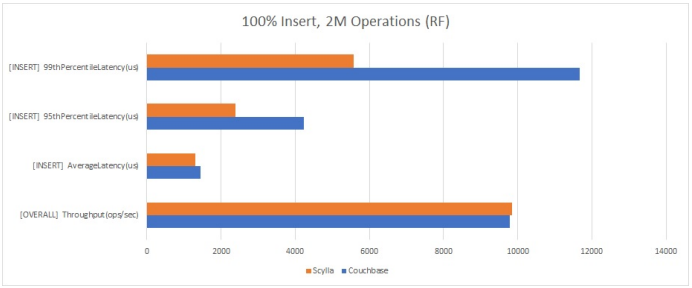

Fig. 3. 100% Insert, 2M Operations (RF1)

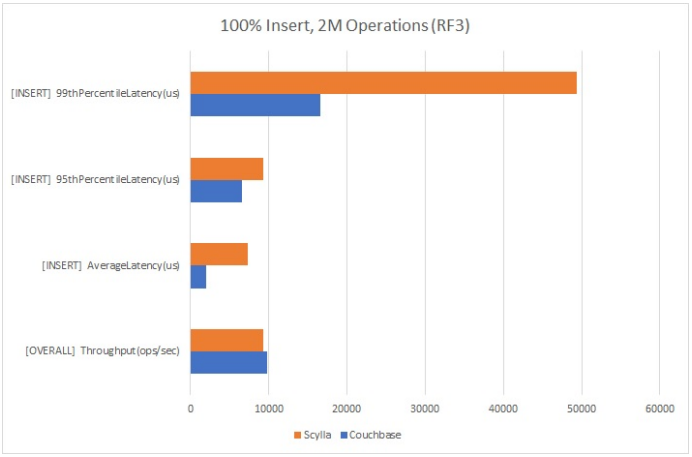

Fig. 4. 100% Insert, 2M Operations (RF3)

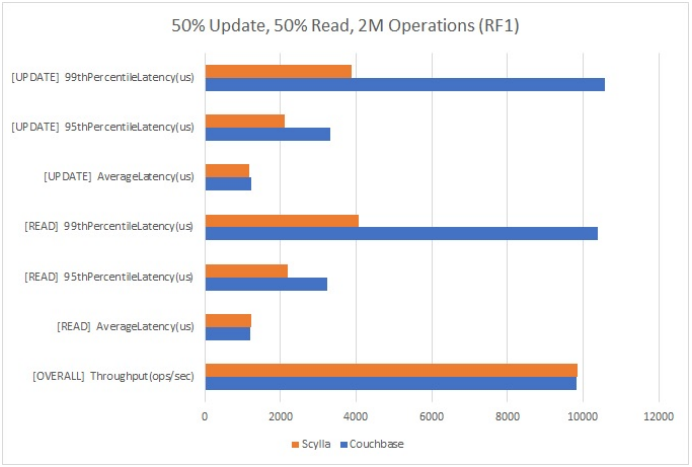

Fig. 5. 50% Update, 50% Read, 2M Operations (RF1)

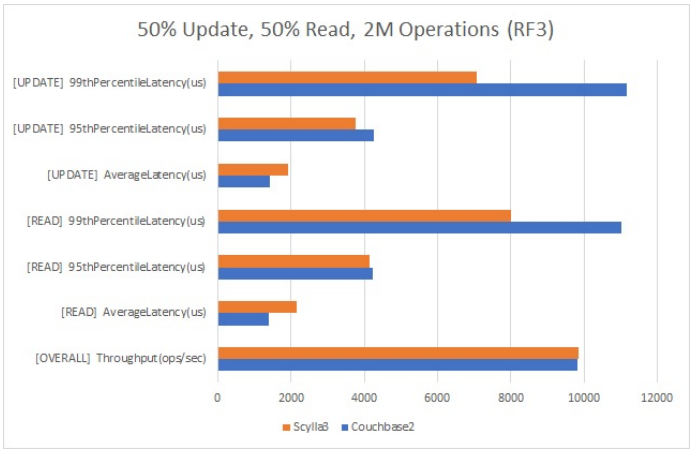

Fig. 6. 50% Update, 50% Read, 2M Operations (RF3)

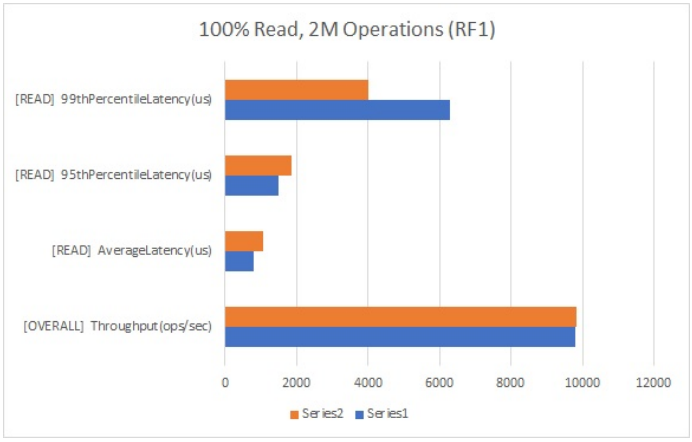

Fig. 7. 100% Read, 2M Operations (RF1)

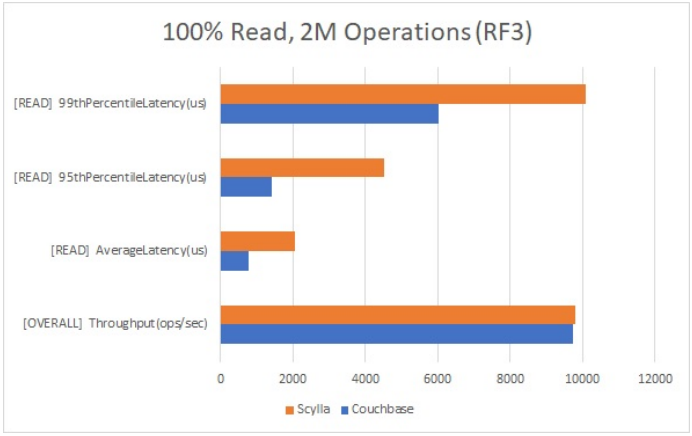

Fig. 8. 100% Read, 2M Operations (RF3)

VIII. Results

100% Insert - RF1 - LOAD

In the loading phase when we insert 2M data we can see clearly that the Scylla suspasses Couchbase in its capability to handle data. Both average latency and the latencies of 95th/99th percentile are lower for Scylla. Also, overall throughput is slightly higher for Scylla which shows its ability to handle insertion operation at that level.

50% Update 50% Read - RF1 - RUN

In the transaction phase we can see that the Scylla handles the update operations and the Couchbase handles the read operations better. Throughput remains at the same level. Again, Scylla is better at the metric 99th percentile which shows that it can perform better at heavy load and adverse conditions.

100% Read/RF1

In this case we compare the two databases when they make only read operations. As we can clearly see Couchbase is better at handling these operation since it has lower average latency and lower latencies at 95th and 99th percentile.

Conclusions for RF1

We conclude that ScyllaDB is better at insertion and update operations and Couchbase is better at read operations when we have replication factor 1. Depending on the kind of the project regarding the insertion, updates and read operations we may choose a different database system.

For RF3

100% Insert - RF3 - LOAD

In the loading phase when we insert 2M data we can see clearly that the Couchbase suspasses Scylla in its capability to handle data. Both average latency and the latencies of 95th/99th percentile are lower for Couchbase which means that with RF3 can insert data more efficiently.

50% Update 50% Read - RF3 - RUN

In the transaction phase we can see that the Couchbase handles both the update operations and the read operations better but again, Scylla is better at the metric 99th percentile. This shows that even if Couchbase has lower latencies it might not be so good at handling heavy loads.

100% Read/RF1

In this case we compare the two databases when they make only read operations. As we can clearly see Couchbase is better at handling these operation since it has lower average latency and lower latencies at 95th and 99th percentile.

Conclusions for RF3

We can clearly see that Couchbase surpasses Scylla at the latencies but lags behind at the percentiles.

IX. Conclusions

Comparing the two databases we come to the conclusion that when it comes to heavy loads ScyllaDB surpasses Couchbase on 95% and 99% percentiles latencies. But when it comes to large amounts of requests, Couchbase is better at handling great throughputs.

In conclusion, comparing ScyllaDB and Couchbase key-value stores databases requires a thorough understanding of their features, performance, scalability, and ease of use. Both

databases offer high-performance, distributed storage, and advanced data management capabilities, making them ideal for modern applications with high data throughput requirements.

ScyllaDB is a high-performance, distributed NoSQL database built on Apache Cassandra's foundation. It is designed for real-time big data workloads and provides consistent, low-latency responses at high throughput. ScyllaDB's shared-nothing architecture allows it to scale linearly, making it an excellent choice for large-scale deployments with high throughput needs.

Couchbase, on the other hand, is a NoSQL document database that offers high performance, scalability, and flexibility. It provides a multi-dimensional scaling architecture that allows users to distribute and scale data across different nodes and clusters. Couchbase also supports advanced indexing and querying capabilities, making it easy to retrieve and manipulate large datasets.

Couchbase is much simpler to setup, configure and monitor than ScyllaDB, providing an administration UI web interface.

REFERENCES

[1] V. Mayer-Schönberger and K. Cukier, "Big Data: A Revolution That Will Transform How We Live, Work, and Think," Houghton Mifflin Harcourt, 2013.

[2] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in Proceedings of the 1st ACM Symposium on Cloud Computing (SOCC), Indianapolis, IN, USA, Jun. 2010.

[3] J. Haberman, A. Lotem, and A. Lomnitz, "Scylla: A Wide-Column Store Based on Per-Core Concurrency," in Proceedings of the 2018 International Conference on Management of Data (SIGMOD), Houston, TX, USA, Jun. 2018.

[4] T. E. Anderson, R. Lee, J. Li, J. J. K. Lee, A. Lu, A. Maggs, J. Papa, and V. Verma, "Couchbase Server: The Distributed NoSQL Document Database," in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, VIC, Australia, Jun. 2015.