Applied Computer Systems
ISSN 2255-8691 (online)
ISSN 2255-8683 (print)
2025, vol. 30, no. 1, pp. 174–186
https://doi.org/10.2478/acss-2025-0019
https://content.sciendo.com

1862
RIGA TECHNICAL UNIVERSITY

Comparative Analysis of Pipeline Architecture, Resource Deployment, and Configuration for Cassandra API-Compatible Databases: ScyllaDB vs. Cassandra

Iryna Tyshchenko[1], Rand Kouatly[2], Talha Ali Khan[3*]
[1–3]*University of Europe for Applied Sciences, Potsdam, Germany*

***Abstract* – This work builds a benchmarking pipeline for resource deployment, configuration input, and the evaluation of Cassandra and ScyllaDB performance. It investigates performance under different scenarios, workload types, and internal structures. Insights and future improvements are provided. Analysing comparisons of both databases, there were several notable differences between Cassandra and ScyllaDB. ScyllaDB demonstrated superior performance in production-ready materialized views, global secondary indices, lightweight transactions, change data capture, row-based data cache, and adaptive behaviour to real-world workloads. On the other hand, Cassandra exhibited advantages, such as a well-established size-tiered compaction approach and the ability to leverage existing Java Virtual machine (JVM) tuning techniques.**

***Keywords* – Bigdata, benchmarking, Cassandra, ScyllaDb, YCSB, AWS, terraform, deployment, CI-CD pipeline.**

I. Introduction

Benchmarking pipeline architecture, resource deployment, and configuration for Cassandra API-compatible databases is a crucial process for evaluating the performance of such databases. In this context, two databases that have gained significant popularity, ScyllaDB [1] and Cassandra [2], are examined. This comparison aims to build a benchmarking pipeline and compare the performance of ScyllaDB and Cassandra under various scenarios using the developed pipeline. This includes testing the pipeline architecture, resource deployment, and configuration for these databases to determine their performance under different workloads. The comparison will involve setting up both databases with the same configuration and workload and then running a series of tests to evaluate their performance. The tests will cover different aspects of the databases, such as read and write performance, latency, and throughput. Several research questions may be investigated when related to benchmarking and databases. This paper will focus only on the following five points:

- How does the performance of Cassandra and ScyllaDB differ under varying resource deployment and configuration parameters?
- How can a benchmark pipeline architecture be developed and integrated into a Continuous Integration and Continuous Delivery (CI/CD) process [3] for evaluating the performance of Cassandra and ScyllaDB?
- What are the key components and considerations to ensure accurate and reliable performance testing results?
- How well does the benchmark pipeline architecture perform in estimating the performance of Cassandra and ScyllaDB under various workload types, including read-heavy, write-heavy, and mixed workloads?
- What are the current implementation limitations and potential improvements for the benchmark pipeline in accurately assessing the performance of the databases under these conditions?

This paper is organised as follows. Section II presents the theoretical and technical background. In contrast, Section III provides the proposed benchmarking architecture, and Section IV presents the benchmark configuration procedures. Experimental results are discussed in Section V. Conclusions, and recommendations are formulated in Section VI.

II. Technical Background

The research delves into the performance evaluation of NoSQL databases, mainly focusing on Cassandra and ScyllaDB, under various benchmarking scenarios. Several peer-review articles and publications have contributed to this study. For instance, the paper by Kuhlenkamp et al. [4] evaluates Cassandra and HBase's performance on different EC2 infrastructure setups. Another study by Mahgoub et al. [5] compares the suitability of Cassandra and ScyllaDB for IoT-generated data, highlighting their performance differences. Anusha et al. [6] discuss Cassandra and MongoDB's pros and cons, while Shirinbab et al.'s paper [7] compares VMware and Docker container virtualisation technologies when running

* Corresponding author's e-mail: talhaali.khan@ue-germany.de
Article received 2025-04-15; accepted 2025-11-21

Cassandra. Pramukantoro et al. [8] explore MongoDB, Cassandra, and HBase's performance on a 10TB data framework. Another study by Seghier et al. [9] compares Redis, MongoDB, and Cassandra using the Yahoo Cloud Serving Benchmarking Tool. The work by Cui et al. [10] compares HBase and Cassandra's read-and-write performance. Abramov et al. [11] analyse Cassandra's scalability properties and response time using the Yahoo! Cloud Serving Benchmark.

Furthermore, the work benefits from books such as "Cloud Service Benchmarking" [12] for benchmarking cloud ser- vices, "Designing Data-Intensive Applications" [13] for data-intensive application design principles, and "Cassandra: The Definitive Guide" [14] for comprehensive insights into Cassandra's architecture.

Terraform's usage was guided by "Terraform: Up and Running" [15], while resources like "Database Benchmarking 101" by Severalnines [16] provided foundational knowledge on database benchmarking.

Additionally, ScyllaDB resources like the comparison with Apache Cassandra [17], "Compaction" [18], "Scylla Architecture" [19], and "SSTable" [20] were used for insights into ScyllaDB's architecture and data organisation.

Overall, the research aims to bridge the gap in benchmarking studies and provide practical guidance for database selection in big data applications.

A. Benchmark Workload

A workload in the context of databases refers to a series of operations executed on the database. These operations can be of different types and are carried out in a specific order, forming a sequence. The workload also specifies how the operations are accessed, either by time or sequence and the operations' variation. Workloads can be generated either from real-world trace-based data or from synthetic data modelled after real-world load patterns [21].

B. Benchmark Types

Benchmarks can be classified into two main types: synthetic and real-world benchmarks. Synthetic benchmarks simulate a workload and can help determine how the mix of hardware and database configuration performs under a specific type of workload. However, synthetic benchmarks do not reflect real-world workload and cannot accurately predict how the application will behave on a given hardware/configuration mix. On the other hand, real-world benchmarks use a data set and queries related to the application, providing a more accurate understanding of the interactions among the application, hardware, and database configuration [16].

C. Benchmarking Framework

The general benchmarking framework typically involves the following four steps:

- *Step 1.* Objective – it is usually a first step that includes setting a clear goal for the benchmarking experiment, defining the process for benchmarking, and identifying the constraints that may be faced during the process.
- *Step 2*. Identification of related entities (delamination) – in this step, it is necessary to identify all the relevant entities, resources, artefacts, and data related to the benchmarking experiment.
- *Step 3*. Benchmarking involves measuring or calculating the relevant data and metrics for all the options identified in Step 2 by running the benchmark.
- *Step 4*. Analysis – in this final step, the results obtained in Step 3 with the identified benchmarks will be compared [21].

D. Pitfalls in Benchmark Execution

When measuring the performance of a code block, it is important to consider running it multiple times if it executes very quickly [22]. Before each measurement, memory should be freed up, and any unused references should be cleared. Monitoring for strange results like a continual decrease in performance from run to run in the benchmarking results is beneficial. During measurement, only the relevant functionality should be executed. Logging should be avoided or turned off, and test data should be constructed outside the measured code block. When benchmarking databases, it is essential to consider the impact of transactions on performance. Query operations can be optimised by using indices. It is also important to consider the number of objects being operated on.

To ensure accurate benchmark results, it is recommended to run the benchmark multiple times and record the results in a Comma Separated Value (CSV) or Tab Separated Value (TSV) format that can be easily analysed in a spreadsheet application. It is essential to check the data quality by identifying outliers or calculating the average and variance of all measurements. One of the most crucial points to remember during benchmarking is to understand the behaviour of the database or hardware before concluding the results [23].

E. Apache Cassandra

Apache Cassandra is an open-source distributed and de-centralised storage system that can handle massive amounts of data [2]. It is a column-oriented database that can store any data and automatically distribute it across all nodes in the cluster, allowing for easy addition or removal of nodes without downtime. Its peer-to-peer architecture eliminates the shortcomings of the master-slave paradigm and provides scalability and high availability.

F. ScyllaDb

ScyllaDB is an open-source NoSQL database with high performance, low latency, and scalability [1]. It is based on the Apache Cassandra database and is designed to efficiently handle large amounts of data. ScyllaDB offers many features, such as automatic sharding, automatic data replication, and tuneable consistency levels. It also supports a variety of data models, including key-value, document, and graph. ScyllaDb is compatible with Apache Cassandra, meaning that existing applications built on Cassandra can be migrated to ScyllaDB with ease. Additionally, ScyllaDb is built with a shared-nothing architecture, allowing it to scale linearly by adding new nodes to the cluster. ScyllaDb is rapidly gaining traction as a popular wide-column store with a growing user base, including prominent companies such as Comcast, Discord, Disney+,

Crypto.com, OpenSea, Starbucks, and Zillow [24]. While Cassandra remains the most widely adopted database management system, ScyllaDB is making its mark as the 7th most popular wide column store, as per DB-Engines rankings [25].

G. Critical Differences between ScyllaDb and Cassandra

ScyllaDb and Cassandra differ in several aspects:

- ScyllaDb's materialized views are production-ready while they are still experimental in Cassandra.
- Only ScyllaDb supports global secondary indices that span all nodes in a cluster.
- ScyllaDb's Lightweight Transactions implementation is more efficient than Cassandra's. It uses Paxos consensus protocol to provide Atomicity, Consistency, Isolation, and Durability (ACID) [26] like consistency on a single partition basis.
- ScyllaDb offers Change Data Capture (CDC) [27] as standard Cassandra Query Language (CQL)-queryable tables, making it easily consumable by applications and event streaming systems like Apache Kafka. Cassandra's CDC implementation does not use the CQL interface and is not as elegant or efficient.
- ScyllaDb uses a highly efficient row-based data cache. In contrast, Cassandra uses multiple layers of caching, including the operating system level cache, which is inelegant and inefficient.
- ScyllaDb allows queries to bypass the cache entirely on a per-operation basis, which is helpful for range and full table scans.
- ScyllaDb adapts in real time to real-world workloads. Cassandra requires complex manual JVM tuning, which cannot reflect changing real-world conditions.
- ScyllaDb, written in C++, does not suffer from Cassandra's "stop-the-world" pauses caused by Java Garbage Collection (GC).
- Workload Prioritization: ScyllaDb allows users to set prioritization for specific workloads, minimising the chances they have to be retried or dropped.

The summary of the comparison of these two wide-column databases is provided in Table I.

TABLE I

COMPARISON OF CASSANDRA AND SCYLLADB [14]

Characteristic	Cassandra	ScyllaDb
Storage	Column-oriented	Column-oriented
Transactions	Local	Local
Development language	Java	C++
Protocol type	TCP/IP	TCP/IP
Query language	CQL	CQL
CQL driver	supports	supports
Shard-aware CQL drivers	no support	supports
Tuneable constancy	supports	supports
Horizontal scaling	supports	supports
Vertical scaling	no support	supports
Production-ready materialized views	no support	supports
No "stop the world" garbage collection (GC) pauses	no support	supports
High performance	no support	supports
Workload prioritization	no support	supports
Secondary indexes	restricted	supported
Partition mechanism	sharding	sharding
Secondary db models	no support	key-value store

H. Write and Read Path of ScyllaDb and Cassandra

The write paths for Cassandra and ScyllaDb are shown in Fig. 1. Both databases use an in-memory structure (memtable) for reading and writing data to tables. When the size of the memtable exceeds a certain threshold, it is written to a disk table (SSTable) that is immutable and time-stamped. A compaction methodology is applied to reduce the number of SSTables and remove redundant data. One difference between the two systems is that ScyllaDb triggers SSTable compaction. Whenever a new SSTable is written, Cassandra uses a size-tiered compaction triggered, when a specific number of similarly sized SSTables is present (default is 4) [20]. This means that ScyllaDb's compaction process occurs more frequently, which improves search times for read operations. However, the process is not entirely predictable, which can result in performance variability since compaction is a compute-intensive process [20].

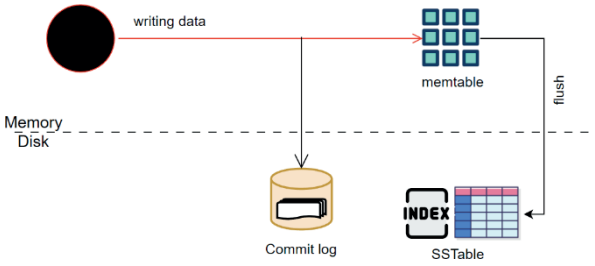

Fig. 1. Write path of Scylla database [28].

I. Benchmarking Tool Choice

The Yahoo! Cloud Serving Benchmark (YCSB) is an open-tool used for evaluating cloud-based database management systems (DBMS) [29]. It is the most widely recognised benchmarking suite for NoSQL databases. It is commonly used to compare the performance of multi-node and NoSQL database management systems on distributed infrastructures, including the public cloud [30].

YCSB advantages. YCSB offers several advantages in benchmarking databases:

- It enables the comparison of databases with different architectures and measures their performance under various workloads.
- The framework automates essential tasks, such as defining core workloads, connecting to databases with drivers, executing workloads, and collecting performance data.
- It supports over 40 NoSQL databases and all JDBC-enabled SQL and NewSQL databases.
- The modular architecture of YCSB has led to several extensions, such as YCSB++ for transactional requests and data consistency assessment, YCSB-T for transactional performance measures, and geoYCSB for geo-workload data.
- Ability to easily extend databases and workloads at the code level.

- Its focus is on standard database operations that are supported by most databases.

YCSB architecture. The YCSB Client is a software program written in Java, which is responsible for generating data to be loaded into the database and for creating a workload by generating a series of operations. The architecture of the client is depicted in Fig. 2. The workload executor controls multiple client threads, each of which executes a sequence of operations to load the database during the load phase and execute the workload during the transaction phase. The client threads adjust the request rate to control the offered load to the database and measure the latency and throughput of their operations. These measurements are then reported to the statistics module. At the end of the experiment, the statistics module aggregates the measurements. It reports average, 95th and 99th percentile latencies and either a histogram or time series of the latencies.

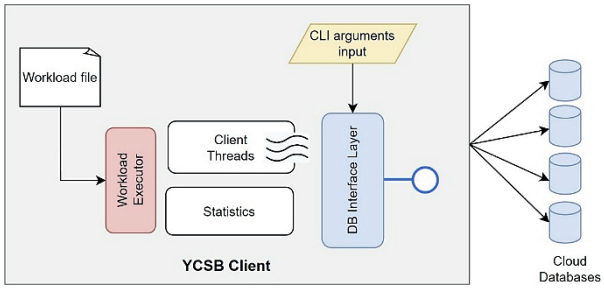

Fig. 2. YSCB simplified architecture. Source: [31].

YCSB workloads. Each workload included in YCSB is designed to evaluate a specific combination of factors, such as data sizes, request distributions, and read/write operations [29]. YCSB workload types are described in Table IX.

J. Quality Metrics

Quality metrics serve as functions that express variations in quality dimensions. It is possible to have multiple metrics that describe the same quality dimension.

When considering quality metrics, specific requirements need to be fulfilled. The primary objective is to provide a meaningful and accurate representation of the assessed quality. This requirement applies to all quality metrics in general.

K. YCSB Results

The YCSB benchmarking platform returns the sum of all database operations and their frequency every 10 seconds, with separate statistics for each operation class, such as READ and INSERT. At the end of the measurement, important performance indicators such as runtime, throughput, and various latency metrics are displayed. Here are some of the Key Performance Indicators (KPIs) provided by YCSB:

- *Throughput* [ops/sec]: This metric quantifies the number of operations (read/write) a system can perform per second.
- *Latency*: Latency measures a system's time to respond to an operation (read/write).
- *CPU Utilisation*: CPU utilisation measures the percentage of CPU time the system utilises during the benchmark.
- *Memory Utilisation*: Memory utilisation measures the system's memory during the benchmark.

L. YCSB Metrics

Average Throughput

Average throughput (T) is calculated as the total number of operations (ops) divided by the total time period (t_i) in seconds:

$$T = \overline{m} = \frac{\sum^{n} m_i}{\sum^{n} t_i}, \tag{1}$$

where n is the number of throughput measurements, m_i –the number of operations for the i-th measurement, and t_i represents the time period for the i-th measurement. Throughput measures the number of requests a cloud service can handle concurrently and is typically represented as requests per second [12].

Average and Upper-Percentile Response Time

Response time is the duration between sending a service request and receiving a successful response. Two important metrics are the average response time, which is computed as the sum of all response time measurements m_i divided by the total number of measurements n, and the upper-percentile response time, which represents the value at a specific percentile of the response time distribution (e.g., 95).

Average Response Time (RT)

Average Response Time is calculated using:

$$RT = \overline{m} = \sum_{i}^{n} m_i / n, \tag{2}$$

where n is the number of response time measurements, m_i represents the response time for the i-th measurement.

Upper-percentile response time RT_p

The Upper-Percentile response time is calculated using:

$$RT_p = F^{-1}(p), \tag{3}$$

where p is the desired percentile value (e.g., 95 % or 99 %), F^{-1} is the inverse cumulative response time distribution function. Latency measures the time required to complete an individual request and is typically reported in milliseconds [12].

III. Benchmarking Architecture

The methodology outlines the proposed benchmarking architecture for Cassandra API-compatible databases, focusing on Cassandra and ScyllaDB using the YCSB framework. The proposed architecture is shown in Fig. 3.

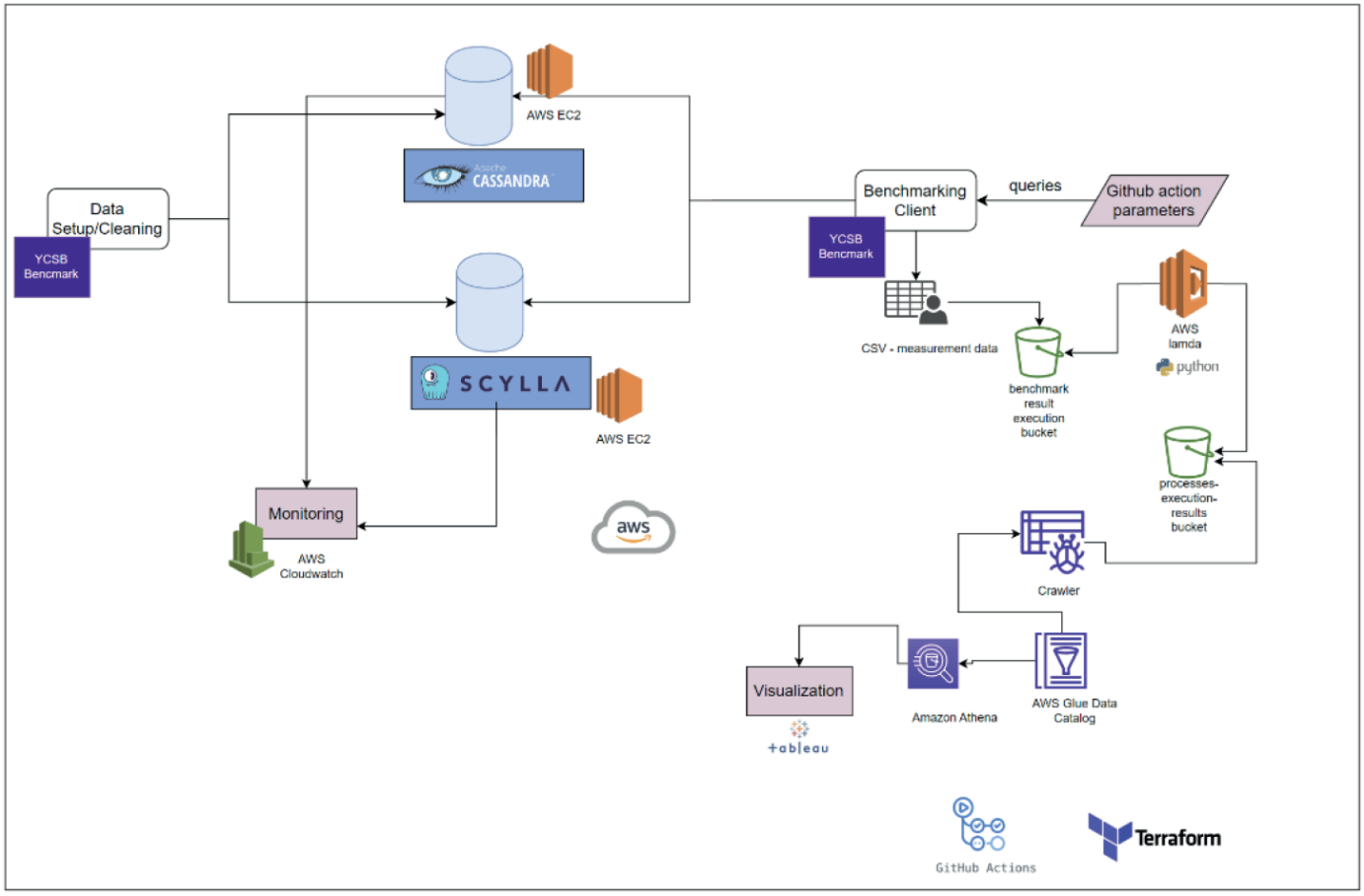

Fig. 3. The proposed architecture of benchmarking pipeline.

A. Data Setup/Cleaning Component

Data setup is executed using the YCSB benchmarking framework. To perform benchmark testing on a database system, the first step is to set up the system on a single machine or a cluster based on the desired configuration. The next step is to create or set up storage buckets (keyspaces) to store records. Before running the YCSB Client, creating the necessary tables is essential [32], [33].

The cleanup script automatizes cleaning up after tests have been run. It consists of the following stages:

- Runs `nodetool cleanup` to clean up the node's data.
- Runs `nodetool clear snapshot` to remove all existing snapshots on the node.
- Stops the Cassandra service using `systemctl stop, Cassandra`.
- Deletes all data stored in `/var/lib/cassandra/data/` directory using `sudo rm -rf "/var/lib/cassandra/data/*"`
- Starts Cassandra service again using `systemctl start cassandra`.
- After the above operations are performed on all instances in the cluster, the script creates a keyspace and a table in Cassandra using the CQL statements provided in the CQL variable. The script uses `cqlsh` to execute these statements on each instance in the cluster.

B. YCSB Benchmarking Client Setup

To set up the benchmarking client node before benchmarking, several minimal machine specifications should be followed, as shown in Table II.

TABLE II
YCSB CLIENT MACHINE SPECIFICATION

Instance type	t2.small
OS	Linux 22.04 LTS, amd64
Storage	8 GB

After the machine is set up, the *./scripts/client-setup.sh* will be executed, which installs and sets up the necessary dependencies for running the Yahoo! Cloud Serving Benchmark (YCSB) against a Cassandra cluster.

C. Cassandra Installation

Machine specification for Cassandra installation is shown in Table III.

TABLE III
CASSANDRA MACHINE SPECIFICATION

Instance type	m3.large
OS	Linux 22.04 LTS, amd64
Storage	8 GB

The instance type was selected based on the AWS guidelines [34]. Cassandra uses several ports that must be open for communication, as shown in Table IV.

TABLE IV
CASSANDRA SECURITY GROUPS

Port	Description	Protocol
9042	CQL (native transport port)	TCP
7000	Inter-node communication (RPC)	TCP
7001	SSL inter-node communication (RPC)	TCP
7199	JMX management	TCP
10000	Scylla REST API	TCP
9160	Legacy client port (Thrift)	TCP

Security groups used to function like a firewall to allow or forbid data flow and access [35]. The bash script is written to install Apache Cassandra on a Debian-based operating system [36]. The script will install the latest version of Cassandra. At the moment of installation, it was Cassandra 4.1.

After installation, the database setup takes place. The following changes are made to the *Cassandra.yaml* file on each instance in the Cassandra cluster, as shown in Table V.

These changes ensure the Cassandra cluster is configured correctly to work in an AWS environment and use the correct IP addresses to communicate between nodes.

TABLE V
CASSANDRA YAML CONFIGURATION

Setting	Value
seeds	the comma-separated string of public IP addresses of all the instances in the cluster
RPC_address	0.0.0.0
num_tokens	256
cluster_name	CassandraBench
endpoint_snitch	Ec2MultiRegionSnitch
broadcast_rpc_address	the public IP address of the instance
listen_address	the private IP address of the instance

D. Scylla Installation

Machine specification of Scylla installation is shown in Table VI.

TABLE VI
SCYLLA MACHINE SPECIFICATION

Instance type	i3.large
OS	22.04 LTS, amd64
Storage	30 GB
Scylla Version	5.2

Scylla uses a variety of ports for different purposes [37]. Table VII lists these ports along with their corresponding descriptions and protocols.

TABLE VII
SCYLLA'S SECURITY GROUPS

Port	Description	Protocol
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 (Optionally)	TCP
9160	Scylla client port (Thrift)	TCP
19042	Native shard-aware transport port	TCP
19142	Native shard-aware transport port (SSL)	TCP

Amazon Machine Image (AMI) [38] on AWS cloud exists to install Scylla for testing purposes. In the script, several changes are made to the *Scylla.yaml* configuration file using sed commands via Secure Shell (SSH) on remote instances described in Table VIII.

TABLE VIII
SCYLLA YAML CONFIGURATION

Setting	Value
seeds	the comma-separated string of public IP addresses of all the instances in the cluster
rpc_address	0.0.0.0
cluster_name	ScyllaBench
endpoint_snitch	Ec2MultiRegionSnitch
broadcast_rpc_address	the public IP address of the instance
listen_address	the private IP address of the instance

Both databases are compatible with the Application Programming Interface (API); thus, the step of the YCSB database setup is similar for both. It will recreate the databases.

E. Monitoring

This component monitors the resource usage of the deployed databases on the AWS cloud. There are several available by default services that can be used, such as CloudWatch Metrics, CloudWatch Logs, CloudWatch Alarms, and Cloud-Watch Dashboards [39].

F. Benchmarking Client Setup

It is a shell script that runs YCSB workload tests on a Cassandra or Scylla database. The script takes several input parameters such as the database host addresses, read and write consistency levels, workload name, thread count, cluster size, database name, operation count, record count, and result directory.

The script runs two phases – load and run – for each thread count and operation count combination. In the load phase, YCSB loads data into the Cassandra or Sylla database, and in the run phase, YCSB runs the workload test.

The script generates a CSV file in the specified result directory for each phase and parameter combination. After the benchmark execution, the CI/CD pipeline step transfers the CSV result files from Amazon Elastic Compute (EC)2 machines to the Storage Service 3 (S3) AWS bucket.

G. S3 Bucket and Lambda

Benchmark results are obtained by executing the bucket, which contains the result for each test run in CSV format. A lambda function written in Python must be executed on each file created with prerequisite conditions [40], [41].

H. Visualization

Tableau has been used to visualise the results. The connector to Amazon Athena [42] is used to connect Tableau to AWS. The processing in the database starts with a crawler. The crawler in AWS Glue connects to the S3 bucket and scans the

data to generate metadata about the files. This metadata creates a schema describing the data stored in the S3 bucket [43].

After the crawler has finished scanning the data in the S3 bucket, it generates a table definition that describes the data stored in the bucket. Using Athena, this table definition is used to query the data stored in the bucket [44].

After the table has been created, the Tableau API connector executes the SQL queries against it using Athena. Tableau has a set of predefined connectors [45], but AWS Athena requires downloading and setting up a driver [46].

IV. Benchmark Configuration

A. Deployment Resources with Terraform

Two workflow files are used to configure the benchmarking pipeline. The overall steps are shown in Fig. 4 and described below:

1) Git checkout: This step checks out the repository code using the `actions/checkout` action.
2) Configure AWS credentials: The AWS credentials are configured using the `aws-actions/configure-aws-credentials` action.
3) Setup Terraform: The Terraform setup uses the `hashicorp/setup-terraform` action. It installs the specified version of Terraform.
4) Terraform fmt: This step uses the `terraform fmt` command to check the formatting of the Terraform files.
5) Terraform Init: The Terraform initialisation is done using the `terraform init` command. It initialises the Terraform working directory with the specified backend configuration.
6) Terraform Validate: The `terraform validate` command is used to validate the Terraform configuration files.
7) Terraform Plan: This step runs the terraform plan command to generate an execution plan. It only executes if the event triggering the action is a pull request.
8) GitHub Script: The `actions/github-script` action is used to create a comment on the pull request.
9) Terraform Plan Status: If the outcome of the `terraform plan` step is a failure, this step is triggered. It runs the `exit 1` command to indicate a failure.
10) Terraform Apply: This step runs the `terraform apply` command to apply the changes to the infrastructure after completing the merge request.

The code and other resources are published in GitHub [47].

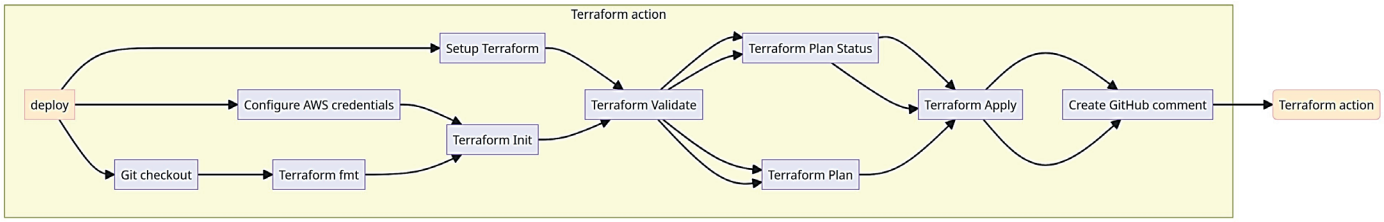

Fig. 4. Terraform CI-CD workflow.

B. CI/CD Benchmarking Pipeline

Another workflow file, called benchmarking, is used to set up resources before benchmarking and to execute load, run, and clean phases of the YCSB benchmarking utility:

1) **Benchmark**: This workflow is triggered by completing another workflow called “Terraform action”, as well as push and pull request events on the “main” branch.
2) **Cassandra-setup**: This job sets up Cassandra by providing necessary AWS credentials and running a bash scriptto set up Cassandra.
3) **Scylla-setup:** This job sets up Scylla by performing similar steps as the Cassandra-setup job but runs a different setup script written for Scylla.
4) **ycsb-benchmark:** This job performs the YCSB benchmark by configuring AWS access and running the YCSB script. Here are the specific benchmarking parameters:
   - *host addr:* Host address;
   - *read consistency:* Read consistency level;
   - *write consistency:* Write consistency level;
   - *workload name:* Workload name;
   - *threads:* Number of threads;
   - *cluster size:* Size of the cluster;
   - *database:* Database name;
   - *operations count:* Number of operations;
   - *recordcount:* Number of records;
   - *resultdir:* Directory for storing results.
5) **move-to-s3:** This job transfers benchmark results to an S3 bucket by running a pre-configured script.
6) **Clean-up-Scylla:** This job cleans up the Scylla setup after completing the benchmark by executing a cleanup script.
7) **Cleanup- Cassandra:** This is a similar step as mentioned above but executed for Cassandra.

In general, the CI/CD process for benchmarking, including all described steps, is shown in Fig. 5.

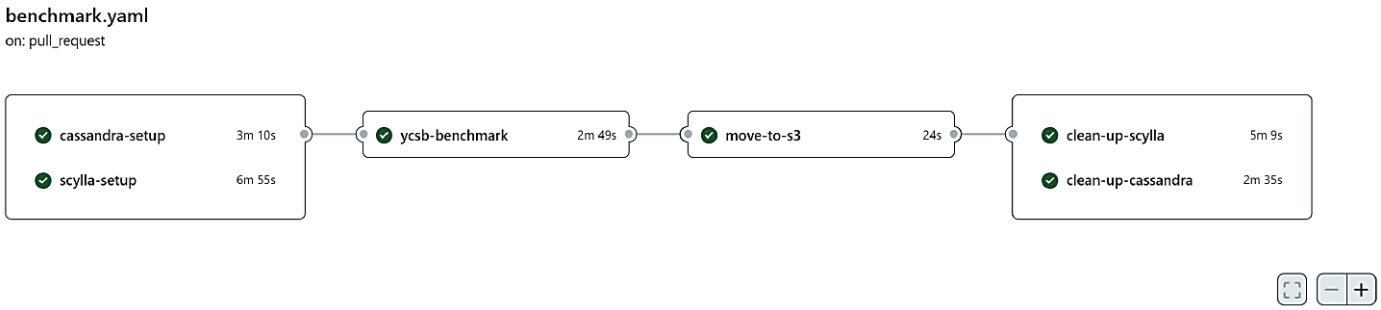

Fig. 5. CI/CD steps with GitHub action for the benchmarking process.

C. AWS Cloud Cost Estimation

AWS pricing is determined based on three key components: computation, storage, and data transfer.

To achieve cost optimisation within AWS, four fundamental steps can be followed: right-sizing services, utilising reserved instances, leveraging the spot market, and utilising the Cost Explorer [48], [49].

Description	Service	Upfront	Monthly	First 12 months total	Currency
m3.large cassandra	Amazon EC2	0	4.890833	58.69	USD
i3.large scylla	Amazon EC2	0	5.341667	64.1	USD
t2.small client	Amazon EC2	0	0.820417	9.85	USD
Estimate summary					
Upfront cost	Monthly cost	Total 12 months cost	Currency		
0	11.05	132.6	USD		

Fig. 6. Cost estimation for benchmarking pipeline.

As per the table in Fig. 6, this estimation shows the cost of running benchmarking EC2 instances, including Cassandra, Scala, and small benchmarking clients, for one hour daily. The general length of one configuration change bench-mark execution is around 15 min. Thus, the price of running one benchmark pipeline will be around:

$$\frac{11.05\$}{30\,\text{days}\times 60\,\text{minutes}}\times 15\,\text{minutes} = 0.0917\$\,, \qquad (4)$$

which is relatively cheap. It is worth mentioning that the AWS calculator does not show all expenses but provides an approximate estimation.

V. Experiments Results

Experiments consist of comparison performance of two NoSQL databases: Cassandra and ScyllaDb. Both databases are API compatible, but their architecture and implementation are entirely different. In these experiments, the workload selection was limited to Workload A, Workload B, Workload C, and Workload D, see Table IX.

For each test, the number of operations chosen for testing are 1000, 2000, 3000, 4000, 5000, and 6000. The count of records was less than that used in a production environment. Still, this evaluation was used primarily to show the ability to run these benchmarks using the proposed benchmark pipeline. The distribution of selecting the records to be operated was set to Uniform. The Replication Count was set to 3. The Scylla and Cassandra ran on a 3-node setup; the connections between nodes were established so that the nodes could communicate. The update operations on workloads C and D were not executed, as they are read-only and read-modify-write, respectively.

TABLE IX
Workload YCSB Types and Their Applications [29]

Workload	Operations	Record Selection	Practical Example
A. Update-heavy	Read: 50 % Update: 50 %	Zipfian	E-commerce platform where customers frequently browse products (reads) and update their shopping carts (updates).
B. Read heavy	Read: 95 % Update: 5 %	Zipfian	News website where users primarily read articles (reads), with occasional user profile updates (updates).
C.Read-only	Read: 100 %	Zipfian	Content delivery network serving static files like images, where users only retrieve files without modifying them.
D.Read latest	Read: 95 % Insert: 5 %	Latest	Social media platforms where users view the latest posts (reads) and occasionally publish new content (inserts).

For each test, the number of operations chosen for testing are 1000, 2000, 3000, 4000, 5000, and 6000. The count of records was less than that used in a production environment. Still, this evaluation was used primarily to show the ability to run these benchmarks using the proposed benchmark pipeline. The distribution of selecting the records to be operated was set to Uniform. The Replication Count was set to 3. The Scylla and Cassandra ran on a 3-node setup; the connections between nodes were established so that the nodes could communicate. The update operations on workloads C and D were not executed, as they are read-only and read-modify-write, respectively.

Experiment 1 – Analysing throughput over thread count increase for both databases

The throughput of a system can be affected by the number of threads being used to perform operations. In general, as the number of threads increases, the system's throughput increases up to a certain point, after which the throughput may start to decrease or level off. The reason is that increasing the number of threads can help increase the system's parallelism, allowing more operations to be performed simultaneously. However, at some point, the system may become saturated, and adding more

threads may result in the exhaustion of shared resources such as CPU time, memory, or I/O bandwidth.

The exact relationship between thread count and throughput can depend on various factors, including the hardware and software configuration of the system, the type of workload being performed, and the specific implementation of the code. In general, it is a good idea to perform experiments to determine the optimal thread count for a given system and workload. This can involve gradually increasing the number of threads and measuring the throughput at each step or using tools such as load-testing frameworks to simulate different levels of load on the system.

The provided chart in Fig. 7 depicts the throughput over an increase in thread count for Cassandra and Scylla databases, specifically for workloads A and B. This benchmarking comparison aims to evaluate how the throughput of the databases scales as the number of threads increases. As noticed for workload A, as the thread count increases for Cassandra, the throughput initially rises gradually and then reaches a peak. After the peak, the throughput starts to decrease slightly. Scylla shows a similar trend to Cassandra, gradually increasing as the thread count increases. For workload B, the throughput initially rises rapidly as the thread count of Cassandra increases. It reaches a peak and then starts to decline gradually. For Scylla, similar to Cassandra, the throughput rapidly increases with an increase in thread count.

In general, for workloads A and B, the throughput of both databases shows the same behaviours. However, it is slightly higher for Cassandra.

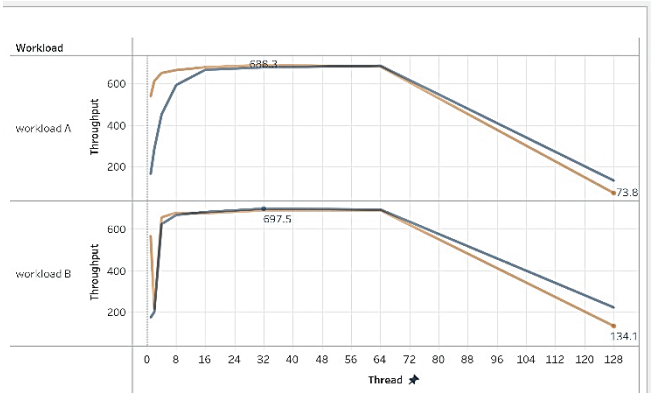

Fig. 7. Throughput over thread count increase for Cassandra and Scylla Databases, workload A and B.

Examining workload C for Cassandra, the throughput initially rises rapidly as the thread count increases, reaches its peak, and stays almost constant. For Scylla, similar to Cassandra, the throughput shows a rapid increase with an increase in thread count but then reaches a peak and decreases rapidly. That shows that Cassandra outperforms Scylla on workload C in throughput.

In the case of workload D, for Cassandra, the throughput initially rises with fluctuations as the thread count increases, reaches its peak, and maintains almost constant throughput. For Scylla, it first grows rapidly and then stays almost constant after reaching a peak, generally showing a better throughput than Cassandra. These results are shown in Fig. 8.

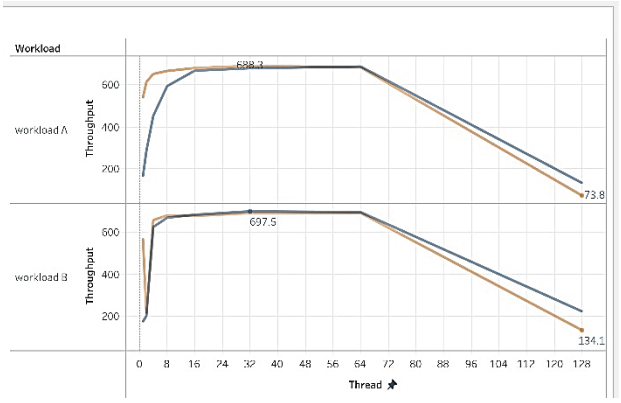

Fig. 8. Throughput over thread count increase for Cassandra and Scylla Databases, workload C and D.

In general, these charts demonstrate that Cassandra and Scylla experience an increase in throughput as the thread count increases for workload A and workload. B. However, Scylla consistently outperforms Cassandra by maintaining higher throughput levels across the range of thread counts. This suggests that Scylla is better equipped to handle increased concurrency and scale effectively with higher thread counts, resulting in improved overall throughput performance compared to Cassandra.

Experiment 2 – Comparison of average latency for Cassandra and Scylla databases with thread count increase

This experiment compares the average latency of Cassandra and Scylla databases with the increase in thread count for workloads A, B, C, and D. The X-axis represents the number of threads running the workload. In contrast, the Y-axis represents the average latency of the database operation.

The results for workloads A and B are shown in Fig. 9. The chart shows that Scylla consistently outperforms Cassandra for both workloads, with lower average latency values across the range of thread counts tested. As the thread count increases, the average latency of both databases increases, but the increase is much steeper for Cassandra than Scylla. This chart indicates that Scylla is better equipped to handle high thread counts and provides lower average latency than Cassandra for workloads A and B.

The results for workloads C and D are shown in Fig. 10. The chart shows workloads C, Scylla, and Cassandra have relatively similar average latency values across different thread counts. However, as the thread count increases, Cassandra exhibits a slightly higher average latency increase than Scylla.

For workload D, Cassandra outperforms in terms of average latency. Cassandra maintains consistently lower latency values across the range of thread counts. At the same time, Scylla shows a more significant increase in average latency as the thread count increases.

Another chart shown in Fig. 11 depicts the changes in average latency for different workloads when the thread count is set to 64. The observed pattern maintains, with Scylla having a slightly lower average latency.

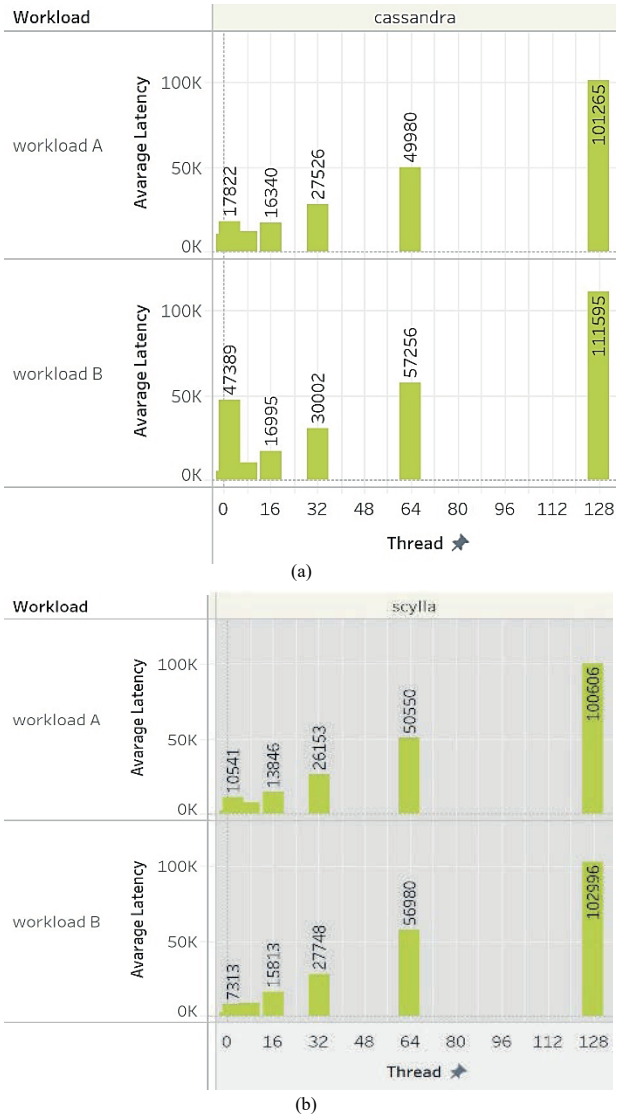

Fig. 9. Average latency over Thread count increase for (a) Cassandra and (b) Scylla Databases, workload A and B.

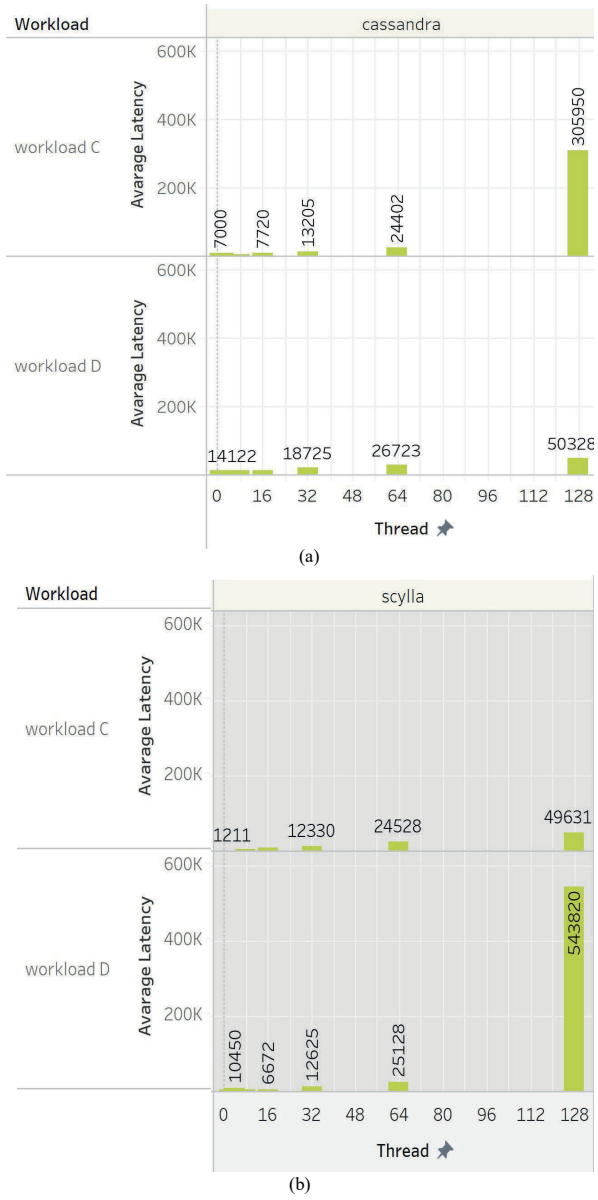

Fig. 10. Average latency over Thread count increase for (a) Cassandra and (b) Scylla databases, workload C and D.

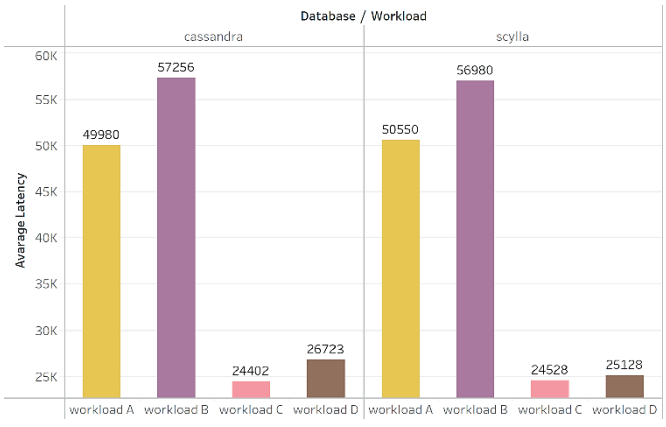

Fig. 11. Average latency over Workload type for Cassandra and Scylla Databases, thread count = 64.

Experiment 3 – Percentages of latencies for different workload types utilising Cassandra and Scylla databases

The benchmarking chart provided in Fig. 12 shows insight into the comparative performance of the Scylla database and Cassandra databases across various workloads and operation types based on their average latencies. Workloads A, B, and C do not have Insert Average latency as they are read-only workloads. The x-axis of the chart represents different workloads or operation types. The y-axis of the graph represents the latency percentage values, indicating the time taken for the operations to complete. Only workload type D contains Insert operations, and for the Scylla database, accumulated average insert latencies are significantly lower compared to Cassandra. It might indicate that Scylla's architecture and design are well-suited for high write throughput. Accumulated read average latencies are primarily the same for each workload for different databases and accumulated update average latencies. Therefore, additional benchmark tests can be executed because both databases handle this number of records for update and insert operations equally well.

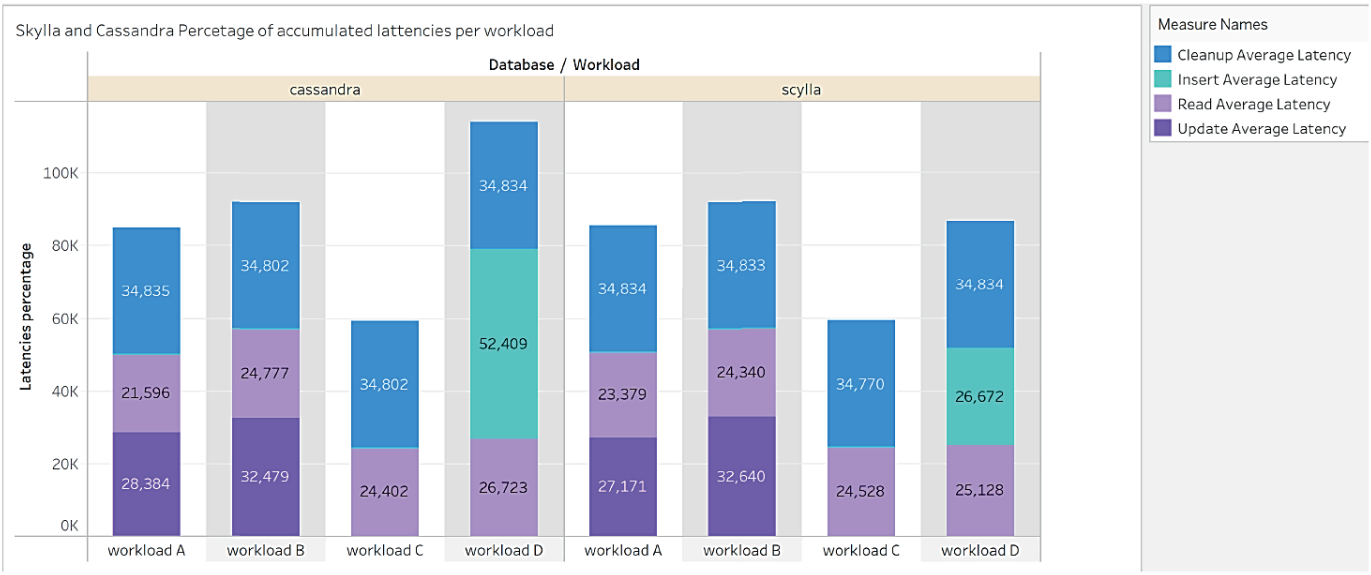

Fig. 12. Average Latency with fractions of Insert, Update, Read Average Latency for Scylla and Cassandra databases for different workloads.

Experiment 4 – Examining throughput over different workloads for Cassandra and Scylla databases

The chart in Fig. 13 compares the throughput performance of Scylla and Cassandra in handling different workloads. For workload A, the throughput for Scylla is consistently lower, indicating worse performance in processing workload A. However, for workload B, Scylla outperforms Cassandra with higher throughput for workload B. It can be explained by the fact that Scylla is best for workload type B rather than type A.

Analysing the result for workload C, Scylla demonstrates superior throughput compared to Cassandra for workload C. For workload D, Scylla exhibits higher throughput than Cassandra for workload D. The throughput for Scylla is notably higher for workload D, suggesting better performance and faster processing for these workloads. Thus, this indicates that Scylla can handle larger workloads more efficiently and achieve better performance in terms of throughput.

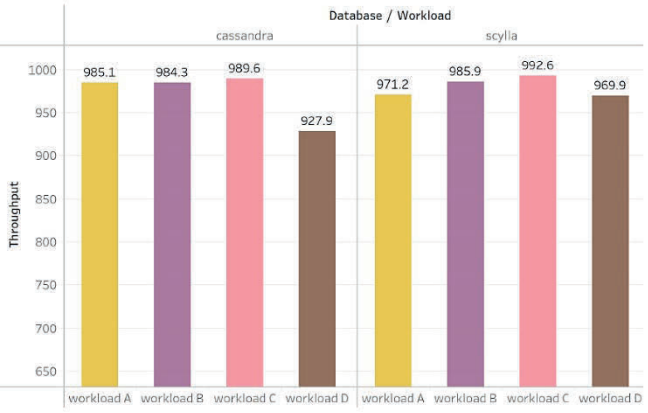

Fig. 13. Throughput over different workload types for Scylla and Cassandra databases.

Experiment 5 – Examining Overall runtime over different workload types for Cassandra and Scylla databases

The chart in Fig. 14 shows the overall runtime of different workload types for Scylla and Cassandra databases. The chart aims to compare the performance of Scylla and Cassandra in terms of the time taken to complete the benchmarking workloads.

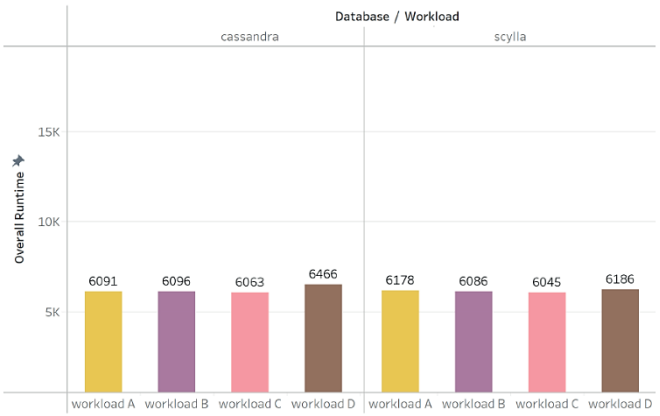

Fig. 14. Overall runtime over different workload types for Scylla and Cassandra databases.

Examining workload A results, Cassandra demonstrates significantly better performance compared to ScyllaDB. The runtime for Cassandra is considerably lower, indicating faster processing of workload A. For workload B, the runtime is relatively low for both Scylla and Cassandra, indicating efficient processing of workload B. Scylla exhibits slightly better performance than Cassandra, as the runtime for Scylla is slightly shorter. For workload C, Scylla shows a notable advantage over Cassandra, with a significantly shorter runtime. Scylla's performance in workload C is considerably better, indicating faster processing and lower execution time. Investigating workload D, Scylla again outperforms Cassandra in workload D, showcasing a shorter runtime. The runtime for Scylla is significantly lower, indicating faster processing and better performance. Thus, the chart demonstrates that Scylla generally exhibits better performance and shorter runtimes than Cassandra across different workload types (A, B, C, and D). This suggests that Scylla can deliver faster processing and improved efficiency for the benchmarked workloads.

VI. Conclusion

During the work within this paper, the benchmarking architecture using Tableau and Athena is proposed to execute Benchmarking workloads, visualise and analyse benchmark results for Cassandra and ScyllaDB. This setup enables query-based analysis, data processing with Lambda, and interactive visualisation. The benchmarks' results help organisations select suitable database solutions and optimise performance. The research contributes an architecture for evaluating database performance and the CI-CD pipeline of the database benchmark, assisting future researchers and practitioners. Practical insights for administrators and developers are also provided, as internals of the databases were compared. Benchmark tests are beneficial during migrations or software upgrades to validate and compare different releases.

Future research on Cassandra API-compatible databases should focus on scalability and performance, workload generation, comparative analysis of different databases, except for Scylla and Cassandra in this study, optimisation strategies, dynamic resource provisioning, execution result automatic visualisation, evaluation of other benchmarking tools, and security considerations.

References

[1] N. Suneja, "Scylladb optimizes database architecture to maximize hardware performance," *IEEE Software*, vol. 36, no. 4, pp. 96–100, Jun. 2019. https://doi.org/10.1109/MS.2019.2909854

[2] A. Wahid and K. Kashyap, "Cassandra – a distributed database system: An overview," in *Emerging Technologies in Data Mining and Information Security – Proceedings of IEMIS 2018*, 2019, pp. 519–526. https://doi.org/10.1007/978-981-13-1951-8_47

[3] M. R. Pratama and D. S. Kusumo, "Implementation of continuous integration and continuous delivery (ci/cd) on automatic performance testing," in *2021 9th International Conference on Information and Communication Technology (ICoICT)*, Yogyakarta, Indonesia, Aug. 2021, pp. 230–235. https://doi.org/10.1109/ICoICT52021.2021.9527496

[4] J. Kuhlenkamp, M. Klems, and O. Röss, "Benchmarking scalability and elasticity of distributed database systems," *Proceedings of the VLDB Endowment*, vol. 7, no. 12, pp. 1219–1230, Aug. 2014. https://doi.org/10.14778/2732977.2732995

[5] A. Mahgoub, S. Ganesh, F. Meyer, A. Grama, and S. Chaterji, "Suitability of NoSQL systems – Cassandra and ScyllaDB — for IoT workloads," in *2017 9th International Conference on Communication Systems and Networks (COMSNETS)*, Bengaluru, India, Jan. 2017, pp. 476–479. https://doi.org/10.1109/COMSNETS.2017.7945437

[6] K. Anusha, N. Rajesh, M. Kavitha, and N. Ravinder, "Comparative study of MongoDB vs Cassandra in big data analytics," in *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, Erode, India, May 2021, pp. 1831–1835. https://doi.org/10.1109/ICCMC51019.2021.9418441

[7] S. Shirinbab, L. Lundberg, and E. Casalicchio, "Performance evaluation of container and virtual machine running Cassandra workload," in *2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech)*, Rabat, Morocco, Oct. 2017, pp. 1–8. https://doi.org/10.1109/CloudTech.2017.8284700

[8] E. S. Pramukantoro, D. P. Kartikasari, and R. A. Siregar, "Performance evaluation of MongoDB, Cassandra, and HBase for heterogenous IoT data storage," in *2019 2nd International Conference on Applied Information Technology and Innovation (ICAITI)*, Denpasar, Indonesia, Sep. 2019, pp. 203–206. https://doi.org/10.1109/ICAITI48442.2019.8982159

[9] N. B. Seghier and O. Kazar, "Performance benchmarking and comparison of NoSQL databases: Redis vs MongoDB vs Cassandra using YCSB tool," in *2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI)*, Tebessa, Algeria, Sep. 2021, pp. 1–6. https://doi.org/10.1109/ICRAMI52622.2021.9585956

[10] X. Cui and W. Chen, "Performance comparison test of HBase and Cassandra based on YCSB," in *2021 IEEE/ACIS 19th International Conference on Computer and Information Science (ICIS)*, Shanghai, China, Jun. 2021, pp. 70–77. https://doi.org/10.1109/ICIS51600.2021.9516864

[11] V. Abramova, J. Bernardino, and P. Furtado, "Testing cloud benchmark scalability with Cassandra," in *2014 IEEE World Congress on Services*, Anchorage, AK, USA, Sep. 2014, pp. 434–441. https://doi.org/10.1109/SERVICES.2014.81

[12] D. Krishnamurthy and D. Bermbach, *Cloud Service Benchmarking: Measuring Performance, Latency, and Cost*. Springer, 2017.

[13] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, Inc., 2017.

[14] J. Carpenter and E. Hewitt, *Cassandra: The Definitive Guide,* 3rd ed. O'Reilly Media, Inc., 2020.

[15] Y. Brikman, *Terraform: Up and Running: Writing Infrastructure as Code*. O'Reilly Media, Inc., 2017.

[16] "Benchmarking databases 101: Part 1 what are benchmarking and performance metrics?" [Online]. Available: https://severalnines.com/blog/benchmarking-databases-101-part-1/

[17] "Scylla. Scylladb vs. apache cassandra: A technical comparison," [Online]. Available: https://www.scylladb.com/scylladb-vs-cassandra/

[18] "Scylla compaction." [Online]. Available: https://docs.scylladb.com/stable/kb/compaction.html

[19] "ScyllaDB shard-per-core architecture." [Online]. Available: https://www.scylladb.com/product/technology/shard-per-core-architecture/

[20] "Sstable." [Online]. Available: https://www.scylladb.com/glossary/sstable/
[21] "What is database benchmarking?" [Online]. Available: https://benchant.com/blog/database-benchmarking
[22] S. Nandgaonkar and V. Khatavkar, "CI-CD pipeline for content releases," in *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*, Bangalore, India, Oct. 2022, pp. 1–4. https://doi.org/10.1109/GCAT55367.2022.9972129
[23] "How to benchmark database performance and ObjectBox." [Online]. Available: https://objectbox.io/how-to-benchmark-database-performance-and-objectbox/
[24] "ScyllaDB users." [Online]. Available: https://www.scylladb.com/users/
[25] "DB-engines ranking." [Online]. Available: https://db-engines.com/en/ranking
[26] B. Medjahed, M. Ouzzani, and A. Elmagarmid, "Generalization of ACID properties." In *Encyclopedia of Database Systems*, L. Liu, M.T. Özsu, Eds. Springer, Boston, MA., 2009. https://doi.org/10.1007/978-0-387-39940-9_736
[27] "Change data capture." [Online]. Available: https://developer.salesforce.com/docs/atlas.en-us.change_data_capture.meta/change_data_capture/cdc_intro.htm
[28] "How data is written." [Online]. Available: https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/dml/dmlHowDataWritten.html
[29] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *SoCC '10: Proceedings of the 1st ACM symposium on Cloud computing*, Jun. 2010, pp. 143–154. https://doi.org/10.1145/1807128.1807152
[30] "YCSB." [Online]. Available: https://www.scylladb.com/glossary/ycsb/
[31] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi "YCSB++: Benchmarking and performance debugging advanced features in scalable table stores," in *SOCC '11: Proceedings of the 2nd ACM Symposium on Cloud Computing*, Oct. 2011, Art. no. 9. https://doi.org/10.1145/2038916.2038925
[32] "YCSB – Yahoo! Cloud serving benchmark." [Online]. Available: https://hse-project.github.io/apps/ycsb/
[33] "Running a workload." [Online]. Available: https://github.com/brianfrankcooper/YCSB/wiki/Running-a-Workload
[34] "Amazon EC2 instance types." [Online]. Available: https://aws.amazon.com/ec2/instance-types/
[35] "Amazon EC2 security groups." [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-security-groups.html
[36] "Configuration of your 5-nodes cluster." [Online]. Available: https://maelfabien.github.io/bigdata/EC2_Cassandra/
[37] "ScyllaDB administration guide." [Online]. Available: https://opensource.docs.scylladb.com/stable/operating-scylla/
[38] "Amazon machine images (AMI)." [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html
[39] B. Beach, "CloudWatch metrics and dimensions," in *Pro Powershell for Amazon Web Services: DevOps for the AWS Cloud*. Springer, 2014, pp. 273–278. https://doi.org/10.1007/978-1-4302-6452-1_16
[40] "AWS Lambda developer guide – programming model (Python)." [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/lambda- python.html
[41] "Lambda with Pandas." [Online]. Available: https://korniichuk.medium.com/lambda-with-pandas-fd81aa2ff25e
[42] "Amazon Athena." [Online]. Available: https://aws.amazon.com/de/athena/
[43] "AWS glue." [Online]. Available: https://aws.amazon.com/glue/
[44] "Connect to Amazon Athena from tableau." [Online]. Available: https://tarsolutions.co.uk/blog/connect-to-athena-from-tableau/
[45] "Tableau connectors." [Online]. Available: https://exchange.tableau.com/connectors
[46] "Tableau support and drivers." [Online]. Available: https://www.tableau.com/support/drivers
[47] I. Tyshchenko, "Scylla Cassandra benchmarking pipeline." [Online]. Available: https://github.com/FairyFox5700/
[48] H. Singh, "AWS pricing and cost management," in *Practical Machine Learning with AWS: Process, Build, Deploy, and Production Your Models Using AWS*. O'Reilly Media, Nov. 2020, pp. 29–44. https://doi.org/10.1007/978-1-4842-6222-1_2
[49] "AWS pricing calculator: Calculate AWS cost like the pros." [Online]. Available: https://spot.io/resources/aws-cost-optimization/aws-pricing-calculator-calculate-aws-cost-like-the-pros/

Iryna Tyshchenko has a Master degree in Software Engineering from the University of Europe for Applied Sciences in Potsdam, Germany, obtained in 2023, and Bachelor degree in Computer Sciences from the National Technical University of Ukraine "Kyiv Polytechnic Institute", obtained in 2017. She is a Software Engineer with strong .NET expertise and a proven track record across global companies. Currently, she is contributing to high-impact projects at Microsoft in Dublin. Previously, she has delivered enterprise solutions at ML!PA Consulting GmbH in Berlin, focusing on system design, .NET development, and scalable architecture. Earlier at EPAM Systems, she progressed from a Junior to a Software Engineer, gaining solid experience in C#, ASP.NET, database development, and cross-functional collaboration. She is skilled in solving complex tasks and continuously expanding knowledge in React.js and reactive programming.
E-mail: tyschenk20@gmail.com

Rand Kouatly is an accomplished academic leader and an expert in software engineering. He serves as a Programme Director for Software Engineering and Vice Dean for Business & Art & Design at the UE Innovation Hub. He holds a PhD in Digital Communication and Machine Learning. He brings vast experience in Java, Python, frontend/backend development, data mining, big data, blockchain, NLP, speech/audio processing, parallel processing, microservices, and multithreading. Before his current role, he was a Postdoctoral Researcher at TU Berlin and the University of Almería. He has lectured since 2016 on Java, ML, and Big Data. His non-academic experience spans project management, consulting, and entrepreneurship; he has been Technical Project Manager, Senior Consultant, and founder of Advanced Solution.
E-mail: rand.kouatly@ue-germany.de

Talha Ali Khan is a Professor of Data Science at the University of Europe for Applied Sciences in Potsdam, where he also serves as a Vice President of Research and Programme Leader for the Data Science MSc degree. His expertise spans Data Analytics, Machine learning, AI, and Optimization. He holds a PhD in Engineering (AI) from the University of Technology Sydney, Sydney, Australia. Before academia, he worked as an electrical engineer and has contributed to R&D in AI across various sectors.
E-mail: talhaali.khan@ue-germany.de
ORCID iD: https://orcid.org/0000-0001-8962-5602