Received: April 27, 2024. Revised: June 1, 2024.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Orchestration Framework Based on Revocable Dynamic Hash Table for Dynamic Access of Data in Multi-Cloud Environments

Zameer Ahmed Adhoni[1]* **Dayanand Lal Narayan[1]**

[1]*Department of Computer Science and Engineering, GITAM School of Technology, GITAM University, Bengaluru, Karnataka, India*
* Corresponding author's Email: zadhoni@gitam.edu

Abstract: The primary aim of this article is to propose a novel orchestration framework based on Revocable Dynamic Hash Table (RDHT) for dynamic and secure access of Electronic Health Records (EHRs) in multi-cloud environments. The proposed orchestration framework includes four processes namely, Function-Level Rewriting (FLR), Optional Function Generator (OFG), Entry Recognition (ER), and Optional File Elimination (OFE) for effectively managing the storage of EHRs. This proposed orchestration framework's performance is analysed using a web application related to hospital management systems. Once the orchestration layer authenticates the health professionals, the RDHT effectively routes the request and distributes EHRs across different cloud providers. The orchestration layer enforces access control after identifying the designated cloud providers. This process confirms that the health professionals have appropriate permission for the specifically requested operation. In comparison to the conventional methods like Quick-Functions as a Service (FaaS), Fast Healthcare Interoperability Resources (FHIR), ElGamal encryption, and Deletable Consortium Blockchain (DS-Chain), the RDHT method consumes minimal memory and running time. More precisely, the RDHT method demands a minimum response time of 1070.52 milliseconds with a memory usage of 89.68 megabyte, access time of 1187.12 milliseconds, and computation overhead of 124.56 megabyte on the Medical Information Mart for Intensive Care (MIMIC-III) dataset, especially for 300 patient records.

Keywords: Dynamic hash table, Electronic health records, Multi-cloud environment, Orchestration framework, Web application.

1. Introduction

In the recent contemporary healthcare environment, paper-based health records are gradually replaced with EHRs. The EHRs provide a centralized digital storage system of patient information such as for test results, allergies, medications, diagnoses, and medical history [1-3]. The EHRs use a standard terminology and format that enhances the interoperability between dissimilar healthcare systems. It is impractical for data owners to manage a vast amount of EHRs [4, 5]. Thus, the cloud based EHR system is an emerging solution as it automatically stores and uploads EHRs on cloud servers. The cloud based EHR system improves confidentiality and integrity of the outsourced EHRs [6]. Furthermore, the cloud based EHR system ensures data security, provides scalability for accommodating growing data volumes, and includes numerous features for cost-efficiency, automatic updates, and interoperability [7]. The cloud based EHR system contributes to improved accessibility to health records, streamlined workflows, and enhanced patient care [8]. The patients are not given the scope for generating EHRs in the traditional cloud based EHR systems [9]. The doctors generate patient's EHRs and outsource it for reducing the computational overhead and communication cost at the patient's end [10,11]. In this circumstance, it is hard to assure the flexibility and integrity of the outsourced EHRs. The semi-trusted doctors modify or forge the outsourced EHRs for hiding medical malpractices [12].

To avoid this challenge, the blockchain technology is integrated with the cloud based EHR

system for addressing the availability, confidentiality, and integrity issues [13,14]. Still, in blockchain based healthcare management system, the computational infeasibility of EHRs depends on the cloud server's unforgeability and reliability [15,16], where these systems lead to a single point of failure [17,18]. In order to address the aforementioned concerns, an efficient orchestration framework with the RDHT method is proposed in this article for securing and improving the storage and access control of EHRs in cloud based EHR systems. Generally, the orchestration framework facilitates the management and coordination of multiple services or tasks in computing environments. The orchestration framework plays a vital role to automate and streamline several processes that ensures effective synchronization and communication between dissimilar components of an application or system. The important contributions of this study are pointed as follows:

- Implemented a new orchestration framework by incorporating four processes which are: FLR, OFG, ER, and OFE. These four processes improve the user experience on a web application related to the hospital management system. In the present decade, the healthcare industry is more dynamic where this proposed orchestration framework provides flexibility alongside easily adapting to technological advancements and regulation changes.
- The proposed orchestration framework automates the repetitive and routine tasks such as inventory management, billing processes, and appointment scheduling that leads to cost and time saving. Furthermore, this framework automatically coordinates and manages several processes in the hospital management system such as resource allocation, patient admissions, etc.
- Integrated RDHT method in this orchestration framework to ensure secure storage and access of EHRs. Four evaluation metrics including memory usage, response time, computation overhead, and access time are utilized for investigating the performance of the RDHT method on MIMIC-III dataset, while the obtained results are compared with four existing methods.

This further article is structured as follows; a literature survey of the existing articles is presented in section 2. The methodology details, simulation results, and conclusion are represented in sections 3, 4, and 5, correspondingly.

2. Literature survey

Shen [19] introduced a novel framework for distributing different replicas of EHRs in multi-cloud environments. This process allowed users to recover corrupted EHRs and resisted the EHRs from copy summation attacks. In the designed framework, EHRs were stored in the form of ciphertext; here, only authorized users were allowed to decrypt the data and access sensitive information. In this framework, a Map Version Marker Table (MVMT) was utilized for data traceability. According to the system model, the MVMT allowed only the authorized doctors to access patient's EHRs for decision-making and disease diagnosis. Security analysis showed the superiority of the developed framework in cloud-based healthcare applications. However, the MVMT allowed only the authorized doctors to access patient's EHRs for decision-making and disease diagnosis. Furthermore, Kanna and Vasudevan [20] initially employed a normalization technique for isolating normal and sensitive attributes in the collected dataset. Next, the normalized data stored on the cloud platform were encrypted utilizing the ElGamal algorithm. In this particular system, a rule based statistical disclosure approach was applied for isolating sensitive data and further, the data security was assured by designing an effective access control policy. The efficacy of the presented system was evaluated using dissimilar measures including execution time, policy generation time, and encryption time. Even though, the suggested ElGamal was time-consuming while assesses with supplementary encryption methods, particularly when utilizing long keys.

Mishra [21] developed a model named DS-Chain for securing EHRs in multi-cloud environments. The DS-Chain model not only ensured the correctness and integrity of the outsourced EHRs but also guaranteed the confidentiality and privacy of sensitive EHRs. Furthermore, a collaborative multi-cloud storage model was employed within the cloud paradigm which provided a reasonable availability and durability for the outsourced EHRs. Based on the patient's EHRs, all the transactions were arranged on the blockchain network to facilitate block deletion. However, due to the computationally infeasibility of Ethereum blockchain, it has been computationally intractable to perform the hard-fork on Ethereum based DS-Chain by any adversary with limited power. Extensive empirical analysis demonstrated the practicability and feasibility of the presented DS-Chain model. Gohar [22] presented a robust and secure Patient Centric Healthcare (PCH) framework based on Internet of Things (IoT), cloud, and

blockchain technology. In this literature, a five-tier architecture was incorporated with the PCH framework for improving its interoperability. Real-time patient's EHRs were utilized for validating the efficacy of this PCH framework. The high latency in various applications due to the extension of middleware using cloud infrastructure services.

Ouchaou [23] incorporated a service publication algorithm, a cloud federation architecture, and a service based management system for effective management of Software as a Service (SaaS) services in multi-cloud environments. The primary objectives of this literature were to increase profit, automate the process of data management, and guarantee user experience. The developed model needs to enhance the context-aware and learning ability of agents in the interoperability framework. Furthermore, Rodrigues [24] deployed three cloud agnostic models for improving interoperability and portability between FaaS platforms which is known as QuickFaaS. These cloud agnostic models assisted the developers in reusing server-less functions across dissimilar cloud providers without installing extra software and changing code. In this literature, the robustness and superiority of these cloud agnostic models was analysed through six evaluation measures. The highest latency was frequently experienced throughout the research of implementation setting.

El-Kassabi [25] introduced a self-adapting model for cloud workflow monitoring, adaptation, and orchestration. This model relied on trust analysis for guaranteeing Quality of Services (QoS) of the workflow. The adaptation and monitoring models triggered various adaptation actions namely, resource scaling, migration and reconfiguration of workflow, and repair time errors. The orchestration of cloud resources was formalized by employing a state machine which extracted the dynamic characteristics within cloud execution environments. A model checker was utilized in this literature for validating the effectiveness of the presented model by means of liveness, security, and reachability. This health monitoring workflow was evaluated on MIMIC III dataset, and the obtained results proved that this workflow orchestration was self-configuring and self-adapting with a higher level of QoS. Here, the data privacy metrics were not considered. Further, the model enables authorized doctors only to access patient's EHRs for decision-making.

Iqbal [26] presented a healthcare monitoring system utilizing orchestration architecture. This monitoring system executed an optimized scheduling process for effectively monitoring a patient's vital signs. The presented monitoring system comprised two phases: (i) an IoT based orchestration architecture was used for optimizing healthcare services, and (ii) performed an optimized scheduling process for task scheduling. Additionally, an e-Health tool-kit was used for monitoring the patient data, where the procured experimental outcomes revealed that this monitoring system superiorly reduced task failures in comparison to other healthcare monitoring systems. The interrelated environment of IoT devices increases substantial security issues. The model enhanced the computational overhead and access time which affect the performance.

Moreno-Vozmediano [27] introduced a nuanced orchestration model for automating the management and deployment of higher availability services in multi-cloud and multi-zone scenarios. In this study, the presented orchestration model deployed the affinity mechanisms (anti-affinity and affinity rules). The automation process of managing and deploying higher availability services encompassed two major concerns of an inadequate management of cross-cloud private networks, and the incorporation of fail-over and global load balancing processes. The developed model unable to find the optimal granularity levels in micro-services.

Okwuibe [28] developed a novel orchestration model for optimal resource allocation in Industrial IoT (IIoT) systems. Based on predetermined constraints, the resources were dynamically allocated by the orchestration model that ensured Service Level Agreement (SLA). The orchestration model used a software-defined scheme for centralized resource management; here, the resources represented edge-cloud resources, bandwidth, power, and memory. This resource management approach effectively accelerated resource orchestration through edge-cloud resource usage and dynamic workload balancing. The robustness of the presented orchestration model was investigated through its task success rate, network utilization, and Central Processing Unit (CPU) utilization. However, as number of edge devices continue increasing, the cloud and edge scenarios displayed substantial drop in tasks leading to lower success rate. Qin [29] developed a dynamic orchestration framework for resource management in Platform as a Service (PaaS) based on Auto-Regressive Integrated Moving Average (ARIMA) model. The simulation outcomes demonstrated that this orchestration framework significantly improved the memory usage and reduced the response time. However, the suggested model does not consider external factors and requires multiple iterations to achieve stationarity.

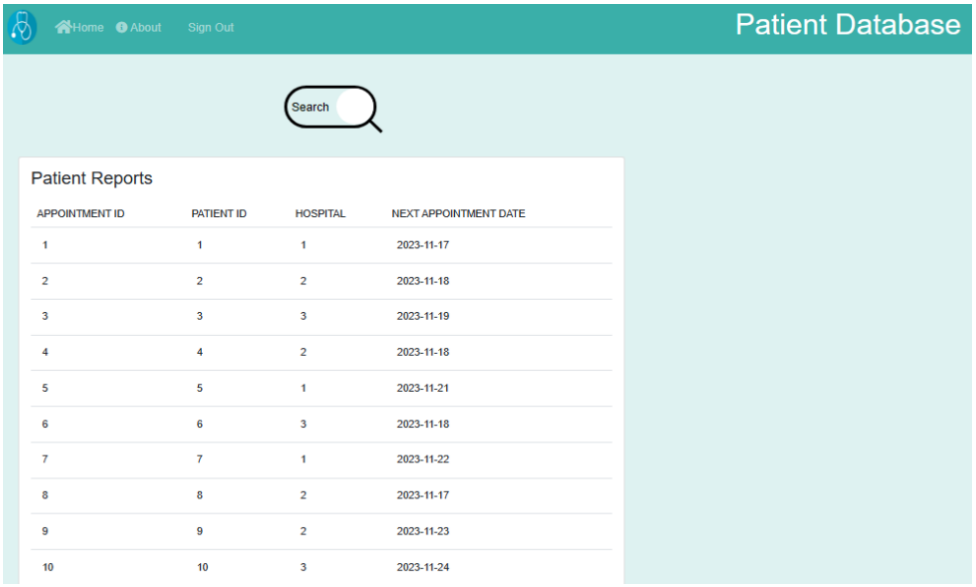
Kazim [30] employed a cloud-on-demand model for securing IoT services in multi-cloud

environments. In this study, effective protocols were implemented and developed within cloud paradigms for facilitating multi-cloud collaboration. This framework included three stages (service matchmaking, authentication, and SLA management), alongside allowing users to access IoT services within multi-clouds. Specifically, the SLA management ensured service execution in external clouds. The developed protocols were executed on two cloud platforms which are Amazon AWS and OpenStack. The high delay in different applications because of the extension of middleware using cloud infrastructure services.

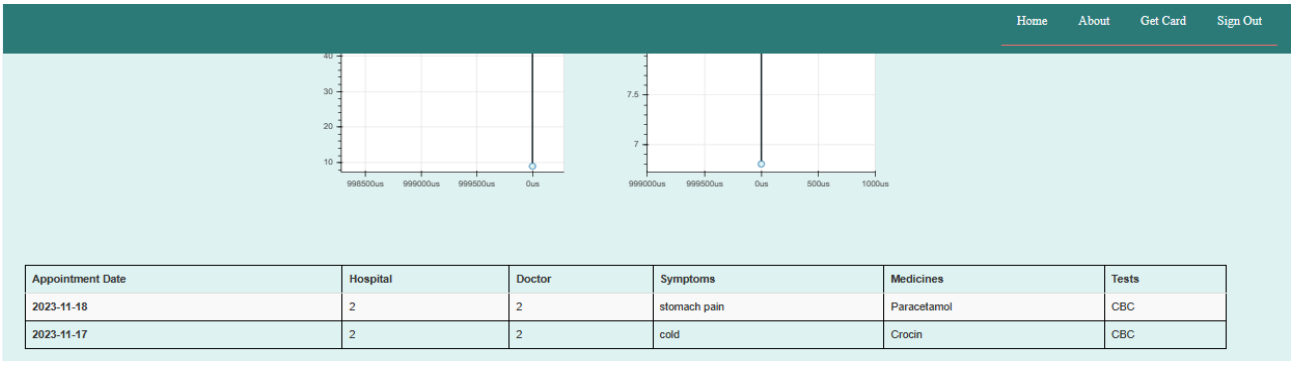
By reviewing the existing literatures, the below three problems are resolved in this present article: (i) high latency in various applications due to the extension of middleware using cloud infrastructure services, (ii) need to enhance the context-aware and learning ability of agents in the interoperability framework, and (iii) need to find the optimal granularity levels in micro-services. In this article, the suggested orchestration framework based on RDHT properly employs applications and services in cloud infrastructure services for solving the latency problems. In this framework, the multi-cloud models are deployed on a single platform for improving learning ability in the cloud. Moreover, the micro-services enable individual cloud services to be scaled independently in order to meet the demand for supporting the application features.

3. Methods

The suggested orchestration framework's performance is analysed on a web application related to hospital management systems. This web application uses My Structured Query Language (MySQL), Hypertext Preprocessor (PHP) database, and this application includes three main modules (admin, user, and doctor). Python was chosen for implementation due to its widespread use and availability of libraries. MySQL was selected for



(a)



(b)

Figure. 1: (a) View from doctor end (b) View from patient end

the database as it is a popular relational database management system suitable for storing structured EHR data. The RDHT method was developed to provide a secure and efficient way to manage EHR data across multiple cloud providers. The details about admin, user (patient), and doctor modules are presented as follows;

Admin module:

Dashboard: Admin view details about new queries, appointment history, logout and login time of doctors, patients and users, patient reports, and patient details like name and mobile number.

Patients: Admin view details about patients.

Doctors: Admin add and update the specialization of doctors.

Users: Admin view details about users and have authorization in deleting irrelevant users and adding relevant users.

User (patient) module:

Dashboard: Users view details about his/her appointment history, medical history, and this module assists users in booking appointments.

Doctor module:

Dashboard: Doctors view details about online appointments, patients' details (name and mobile number), and patients' medical history. In this module, it is possible for doctors to update and add patients. The sample screenshots of the web application related to the hospital management system are represented in Fig. 1.

3.1 Orchestration framework

The orchestration framework is a crucial part of modern software systems. It serves as a central point for controlling a multitude of software programs, data processing operations, and the interdependencies between them. The suggested orchestration framework includes four important processes: FLR, OFG, ER, and OFE. These four processes assist users in effective storage and access of EHRs. At first, FLR is carried-out for selecting optional functions in the web application of hospital management systems. Here, these optimal functions are modified or rewritten to enforce encryption mechanisms, access controls, and security policies for ensuring the integrity and confidentiality of EHRs [31,32].

In this orchestration framework, the FLR comprises two operations of separation operation and rewriting operation. In the separation operation, the selected optional functions are stored in the format of 'string values'. This process ensures that the empty codes in the optional functions are interchanged with other appropriate functions. Furthermore, in the rewriting operation, the custom codes are used for replacing the actual codes which have more lines, thereby making this web application more energy efficient. During the FLR, the functions are extended and rewritten based on the user requirements. This adaptability and flexibility enables healthcare organizations in customizing EHR systems based on specific requirements.

Next, the OFE mechanism eliminates the irrelevant EHRs in a web application related to hospital management systems. This mechanism eliminates four files which are the compiled files, local environmental files, test files present in the general library, and the information about directories in the general library present in the web application. The proper elimination of optional files improves data privacy and security as the additional unnecessary files contain sensitive information. The removal of unnecessary files maintains data protection, reduces memory usage, processing service CPU utilization and minimizes the risk of unauthorized access [33].

Furthermore, the serverless functions are determined in this web application using three approaches, namely Developer Defined Interfaces (DDI), Configuration File Analysis (CFA), and Source Code Analysis (SCA) through the ER process. Firstly, the name of the serverless function is identified via the DDI approach by providing external permits and interfaces to developers. In the introduced orchestration framework, the DDI approach acts as a backup option for identifying entry points of a serverless function. Secondly, the SCA approach determines the configurations of parameters by scanning serverless functions. Thirdly, the CFA approach is an important approach used in this orchestration framework for determining configuration files that are commonly used for resource allocation and permission settings. The CFA is an efficient and simple approach used to find the names of serverless functions in the web application. The effective identification of serverless functions namely, server scaling, maintenance, and provisioning reduce the operational overhead. This process allows administrators and developers to concentrate less on handling infrastructure and more on application development and logic [34]. It results in faster delivery of updates and features, further leading to effective development cycles.

Finally, the OFG is administered to further improve the flexibility and customization of the orchestration framework by supporting scripting and custom logic. This enables the framework to define complex workflows and actions. The OFG helps this framework to interact with different cloud resources

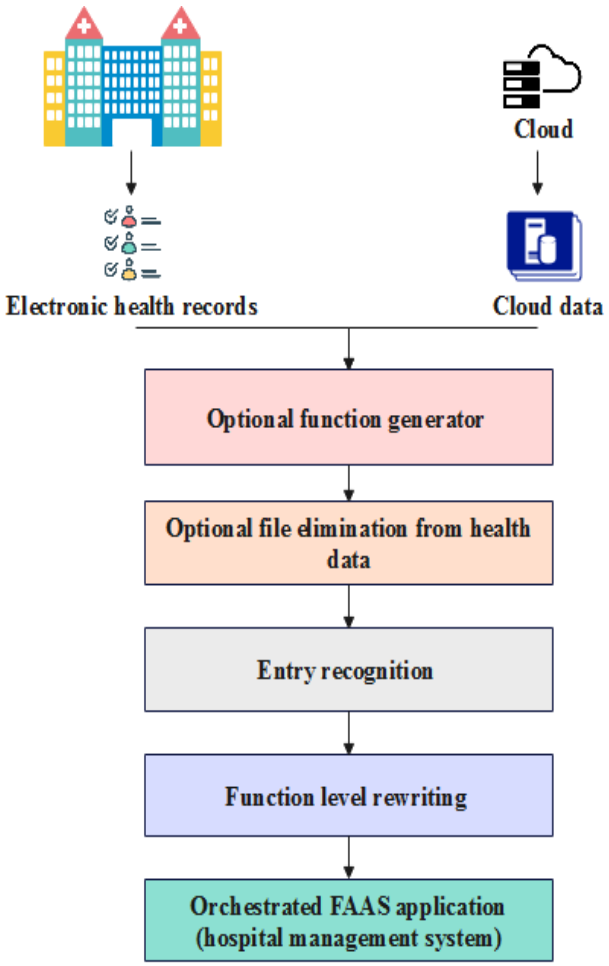


Figure. 2 Workflow of this orchestration framework

such as virtual machines, storage, databases, etc. As it is known, the use of various policies and rules helps users in effective management of resources.

Generally, the orchestration framework sets policies and rules for cost optimization and resource allocation. The OFG automates the policies and rules in the framework for managing a vast amount of events and actions without encountering bottlenecks. Moreover, the web application with OFG demands less training for users by focusing only on specific features. Furthermore, the OFG provides more flexibility that allows users in choosing or customizing particular functionalities according to their needs. This flexibility and adaptability improves the user-experience, and finally, is orchestrated in a FaaS application. The workflow of this orchestration framework is presented in Fig. 2, while its pseudocode is depicted below:

Pseudocode of this orchestration framework

```
Function handle EHR Request (user Credentials, patient ID, operation Type):
    If authenticate User (user Credentials):
        Cloud Provider = determine Cloud Provider (patient ID)
        Load Balanced Provider = perform Load Balancing (Cloud Provider)
        If check Access Control (user Credentials, patient ID, operation Type):
            Update Dynamic Hash Table (user Credentials, patient ID)
            // Revocable access
            Scale Infrastructure If Needed ()
            EHR Data = perform EHR Request (Load Balanced Provider, patient ID, operation Type)
            // Process the EHR data or update as needed
            Return EHR Data
        Else:
            Log Access Denied (user Credentials, patient ID, operation Type)
            Return "Access denied"
    Else:
        Log Authentication Failure (user Credentials)
        Return "Authentication failed"
Hash Table = new Revocable Dynamic Hash Table ()
# Add cloud providers to the hash table
Hash Table. Add Cloud Provider ("cloud1", {'credentials': 'abc123', 'revoked': False})
Hash Table. Add Cloud Provider ("cloud2", {'credentials': 'xyz456', 'revoked': False})
# Dynamically update access credentials for a cloud provider
Hash Table. Update Access Credentials ("cloud1", {'credentials': 'new Credentials', 'revoked': False})
# Dynamically revoke access for a cloud provider
Hash Table. Revoke Access ("cloud2")
# Check if access is revoked for a specific cloud provider
If hash Table. Is Access Revoked ("cloud1"):
    Print ("Access to cloud is revoked")
Else:
    Print ("Access to cloud is granted")
```

3.2 RDHT method

One method for dynamically adding and deleting data buckets on demand is dynamic hashing. In this hashing process, the hash function helps generate a large number of values. The RDHT is inspired from the concept of index hash table which is used to track the patient's EHRs for auditing in the orchestration framework. The RDHT is a two-dimensional data structure which is graphically illustrated in Fig. 4. The RDHT comprises two important elements, block elements and file elements. Every file element encompasses an index number NO_i of a given file F_i. The pointer and the file identifier ID_i represent the first block element and are stored in the format of an

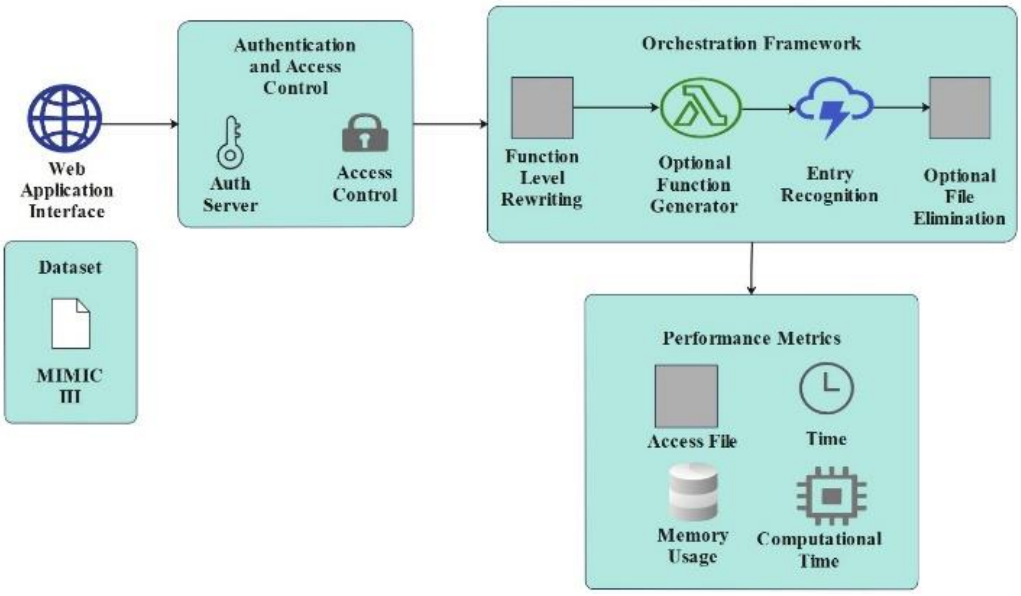


Figure. 3 Architecture of RDHT based Orchestration Framework

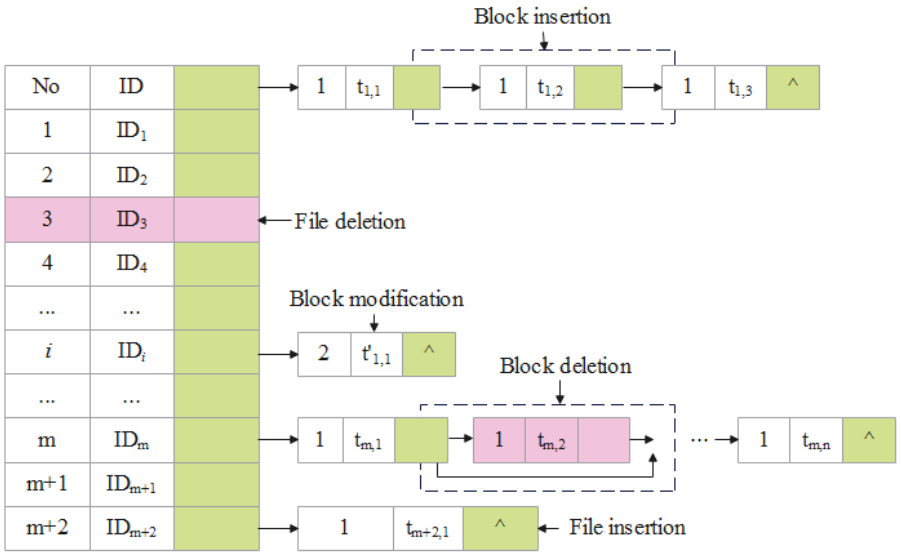


Figure. 4 Graphical presentations of the RDHT method

array-like structure. A linked list is utilized to organize every file with the respective file element as a header node. Every block element (for instance: j^{th} block of the i^{th} file $m_{i,j}$) contains a time stamp $t_{i,j}$, a pointer representing the next node, and a given block $v_{i,j}$. Additionally, the operations that are carried out in the RDHT are divided into two types, block operations and file operations. Both operations include modification, deletion, insertion, and search [35,36].

In the block operation, an appropriate block is searched for locating elements by visiting all nodes. Correspondingly, during the file operation, an appropriate file is searched for locating file elements based on their index values [37-39]. The file insertion step involves two processes: (i) insert file elements into a file array and (ii) construct a linked list. In the constructed linked list, the file modification option is used to update the related block elements and file elements, whereas the file deletion option is used to delete the given file and its file elements in the linked list. By taking the benefits of a linked list, the RDHT significantly outperforms the index hash table in the deletion and insertion of blocks and files. In comparison to the index hash table, the introduced RDHT has limited communication overhead in the updating process and limited computational costs of the cloud service providers [40,41]. The architecture of RDTH based Orchestration framework is presented in Fig. 3.

The pseudocode of the RDHT method is depicted below:

Pseudocode of the RDHT method

```
Class Revocable Dynamic Hash Table:
  Data = {} # Hash table to store cloud provider information
  Def add Cloud Provider (provider ID, provider Details):
    Data [provider ID] = provider Details
  Def remove Cloud Provider (provider ID):
    If provider ID in data:
      Del data [provider ID]
  Def update Access Credentials (provider ID, new Credentials):
    If provider ID in data:
      Data [provider ID] ['credentials'] = new Credentials
  Def revoke Access (provider ID):
    If provider ID in data:
      # Optionally, perform additional revocation actions (e.g., update access keys, tokens)
      Data [provider ID] ['revoked'] = True
  Def is Access Revoked (provider ID):
    Return provider ID in data and data [provider ID] ['revoked']
# Example Usage:
# Assume cloud providers are identified by unique IDs (e.g., "cloud1", "cloud2", etc.)
Hash Table = new Revocable Dynamic Hash Table ()
# Add cloud providers to the hash table
Hash Table. Add Cloud Provider ("cloud1", {'credentials': 'abc123', 'revoked': False})
Hash Table. Add Cloud Provider ("cloud2", {'credentials': 'xyz456', 'revoked': False})
# Dynamically update access credentials for a cloud provider
Hash Table. Update Access Credentials ("cloud1", {'credentials': 'new Credentials', 'revoked': False})
# Dynamically revoke access for a cloud provider
Hash Table. Revoke Access ("cloud2")
# Check if access is revoked for a specific cloud provider
If hash Table. Is Access Revoked ("cloud1"):
  Print ("Access to cloud is revoked")
Else:
  Print ("Access to cloud is granted")
```

The multi-cloud environment developed with the introduced orchestration framework based on the RDHT method ensures secure storage and access of EHRs. The orchestration layer plays a vital role to manage the whole process carried out in the multi-cloud environment. The orchestration layer authenticates the health professionals while he/she initiates a request for retrieving and updating patient records, where the verification of whether or not there is a match of credentials with the required permissions is involved. In addition, the orchestration layer routes the health professional's request based on the RDHT method.

The implemented RDHT method effectively distributes EHRs across different cloud providers. In this stage, the load balancing processes are also applied to analyse the load on every cloud provider. The orchestration layer enforces access control policies after finding the designated cloud provider. This action ensures that the health professional has proper and authorized permissions to carry out the requested operation. The revocable nature of the RDHT method assists this orchestration layer to generate reports on resource usage, continuously monitors system's health, and provides potential security incidents. This revocable nature ensures a secure, scalable, and resilient EHR system in a dynamic landscape of a multi-cloud environment. The graphical representation of the function handle EHR request is presented in Fig. 5. The empirical analysis of the orchestration framework with RDHT method is specified in section 4.

4. Results

In this research study, the orchestration framework based on RDHT is implemented utilizing Python 3.10 software tool with Pycharm 2022.1. This orchestration framework is executed on a personal computer equipped with 16GB Random Access Memory (RAM), Windows 10 (64-bit) operating system, and 4TB hard-drive. In this context, the minimum/maximum instance count ranges from 0 to 300, and 256MB of memory is allocated. The performance of the implemented orchestration framework with RDHT is analysed based on four evaluation metrics: memory usage, response time, computation overhead, and access time. The performance is evaluated by varying the size of patients, hospitals, and doctors. The evaluation metric, memory usage is defined as the amount of memory (RAM) consumed by the implemented orchestration framework for executing and managing tasks. Furthermore, response time is the time taken for providing meaningful results and processing a task or request. The response time is measured in terms of milliseconds (ms).

In this context, the computation overhead is defined as the extra computational resources (processing power) needed by the introduced orchestration framework for managing and executing tasks. The computation overhead is measured by means of Mega-Byte (MB). Additionally, the access time is the time taken by the introduced orchestration framework for accessing and retrieving resources or information from various components in the system.

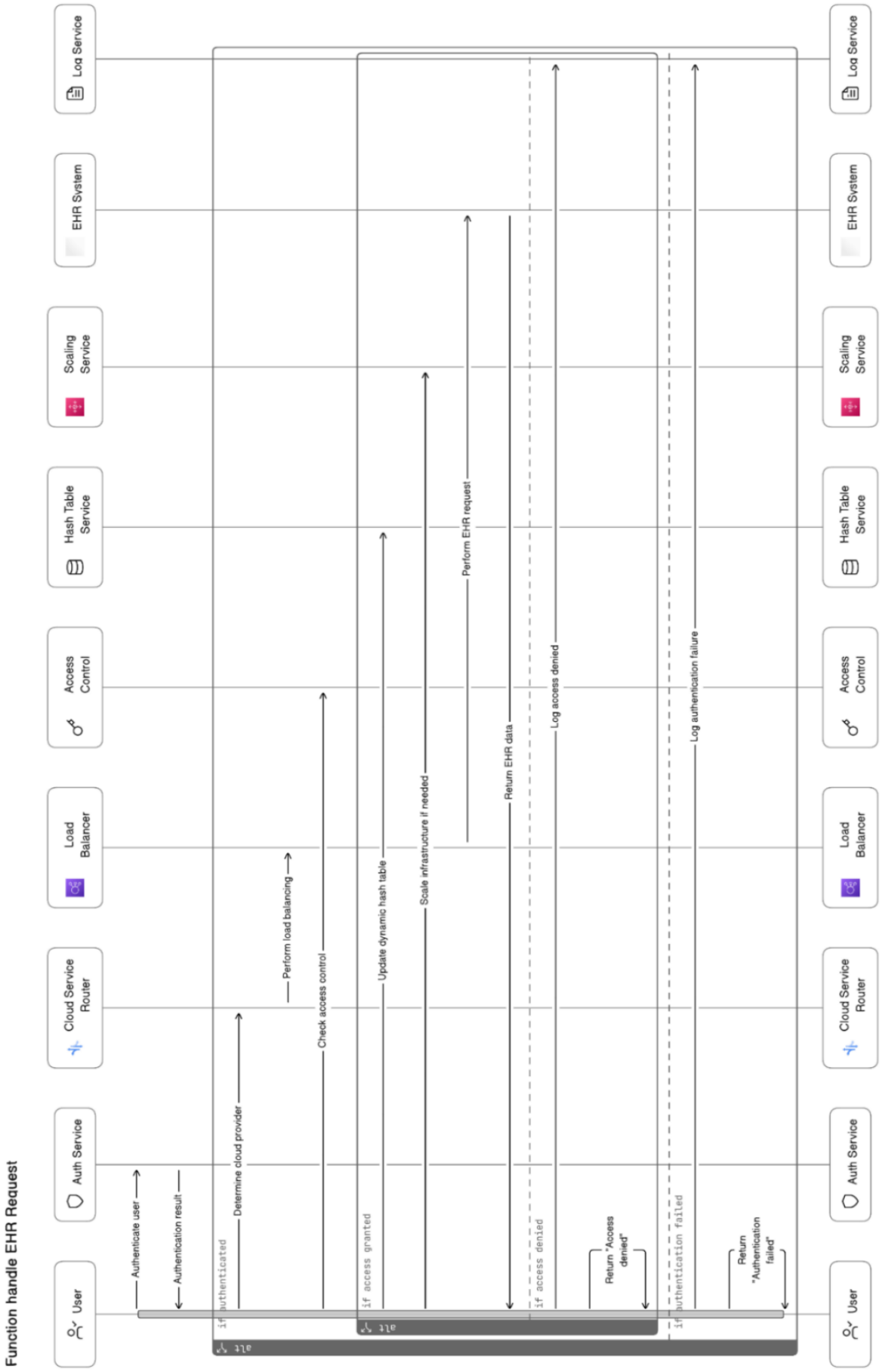


Figure. 5 Graphical representation of function handle EHR request

4.1 Description of dataset

In this article, the MIMIC-III dataset is utilized for validating the performance of the suggested orchestration framework. This dataset comprises records of 60,000 patients admitted in medical centers, specifically in critical care units during the period of 2001 to 2012. This MIMIC-III dataset is accessible through a web based data resource (Physio-Net) and contains several physiological records [39,40]. The MIMIC-III dataset contains several information of hospital admissions, medications, hospital mortality rate, measurements of vital signs, fluid intake records, laboratory tests, and patient demographics. In this particular context, this dataset is chosen because of its higher velocity, volume, and veracity.

MIMIC-III dataset: https://www.kaggle.com/datasets/asjad99/mimiciii

4.2 Quantitative analysis

As illustrated in Table 1, the efficacy of the proposed framework is validated by varying the patient size between 50, 100, 150, 200, 250, and 300, as it is analysed using four evaluation metrics: memory usage, response time, computation overhead, and access time. The paper does not provide a detailed functional evaluation of the proposed system in terms of practicality. However, it mentions that the orchestration framework's performance was analyzed using a web application related to hospital management systems, suggesting a practical implementation and evaluation. By investigating Fig. 6, it is seen that the orchestration framework based on RDHT consumes minimal memory usage and computation overhead, alongside requiring minimal response time and access time. In the orchestration layer, the RDHT effectively organizes data into key-value pairs for secure storage and access. This effective data organization optimizes memory usage and reduces redundant storage. Further, the health data is distributed across the hash table with the usage of an efficient hash function. This process reduces excessive collisions and results in better memory usage. The RDHT provides constant and fast data retrieval, especially in the context of healthcare management, where the patients' health information is quickly accessed, thereby contributing to responsive and significant healthcare services. As stated in Table 1, the suggested orchestration framework consumes minimal memory usage of 89.68MB, response time of 1070.52ms, computation overhead of 124.56MB, and access time of

Table 1. Achieved results of the proposed framework on varying the patient size

Patients	Memory usage (MB)	Response time (ms)	Computation overhead (MB)	Access time (ms)
50	110	1393.32	164.38	1408.25
100	118.01	1413.52	135.34	1456.54
150	148.43	1333.85	136.64	1353.18
200	169.12	1373.95	144.43	1318.94
250	178.22	1428.26	145.10	1279.86
300	89.68	1070.52	124.56	1187.12

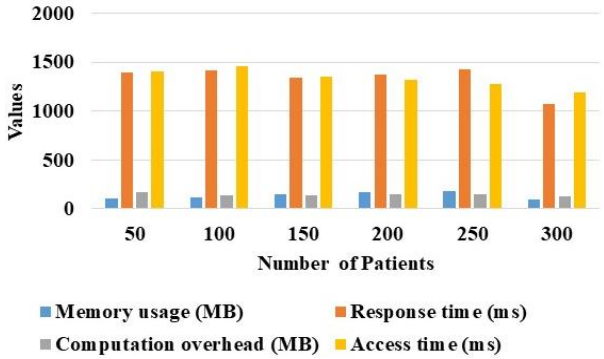


Figure. 6 Visual representation of the proposed framework results on varying the patient size

Table 2. Achieved results of the proposed framework on varying the hospital size

Hospitals	Memory usage (MB)	Response time (ms)	Computation overhead (MB)	Access time (ms)
1	138.78	1285.45	130.10	1294.14
2	114.38	1365.80	138.78	1324.33
3	125.34	1371.26	166.90	1494.24
4	140.10	1413.48	181.69	1510.74
5	151.60	1456.98	214.42	1559.81

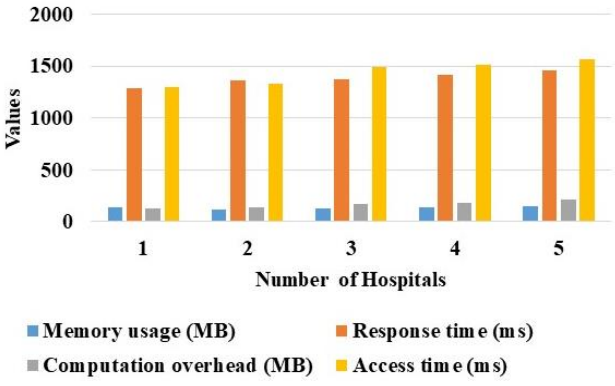


Figure. 7 Visual representation of the proposed framework results on varying the hospital size

1187.12ms, even after increasing the size of the patient to 300. The visual representation of the recommended framework's results when the patient sizes are varied are illustrated in Fig. 6.

The experimental results of the recommended orchestration framework when the hospital sizes are varied are represented in Table 2. By examining the obtained results, the recommended orchestration framework is seen to consume more memory usage and computation overhead, and takes maximal response time and access time in the scenario of increasing the hospital size. In this case, the RDHT effectively accommodates updates and alternations in patient health records and variations in data structures without any performance degradation. In a distributed healthcare system, this adaptability nature is more advantageous when the health information about patients is continuously evolving. Therefore, the RDHT is recommended for supporting the distribution of EHRs across several servers and nodes. This process facilitates the effective management of EHRs in distributed healthcare environments. The RDHT provides predictable, consistent, and secure data retrieval performance even when the size of the dataset is higher. In real-time healthcare applications, this consistent nature of RDHT maintains a higher-level of service. The visual representation in Fig. 7 depicts the suggested framework's outcomes when the hospital size is varied.

Correspondingly, the experimental results of the recommended orchestration framework when varying the doctor size is presented in Table 3, while its performance is validated utilizing four evaluation metrics. By examining the obtained results, it is seen that the memory usage, response time, computation overhead, and access time are significantly increased by increasing the size of doctors (5, 10, 15, 20, and 25). The visual representation of the outcomes of the presented framework on varying the doctor size is illustrated in Fig. 8. The response time and access time of the orchestration framework is significantly reduced because of the consistent time retrieval property of RDHT. In the orchestration layer, the RDHT is employed for managing the relationship among dissimilar healthcare entities like combining EHRs for the test results, prescriptions, and medical history. It additionally contributes in generating a comprehensive health profile for a patient.

In this healthcare framework, the presented RDHT method's performance is compared with four existing methods namely, Quick-FaaS, FHIR, ElGamal encryption, and DS-Chain for 300 patient records, where the obtained results are depicted in Table 4 and Fig. 9. In this orchestration framework,

Table 3. Achieved results of the proposed framework on varying the doctor size

Doctors	Memory usage (MB)	Response time (ms)	Computation overhead (MB)	Access time (ms)
5	123.50	1267.06	199.18	1283.24
10	139.48	1231.38	225.25	1322.21
15	212.41	1323.50	248.81	1394.33
20	248.88	1360.35	272.60	1428.12
25	298.72	2419.60	332.92	1427.09

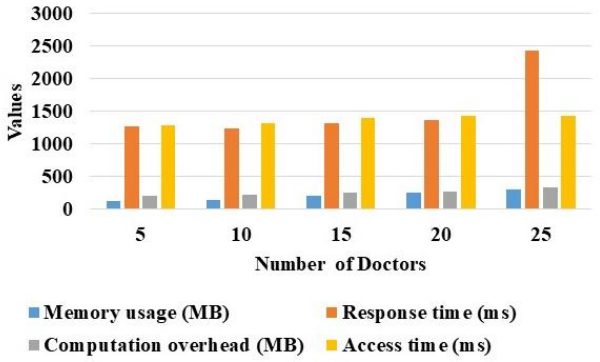


Figure. 8 Visual representation of the proposed framework results on varying the doctor siz

Table 4. Results of state-of-the-art methods

Methods	Memory usage (MB)	Response time (ms)	Computation overhead (MB)	Access time (ms)
Quick-FaaS	378	1458.78	1251.88	1638.77
FHIR	464	2423.98	1803.22	2407.83
ElGamal encryption	389	1257.83	1531.12	1468.03
DS-Chain	278	1435.84	1362.68	1335.91
RDHT	89.68	1070.52	124.56	1187.12

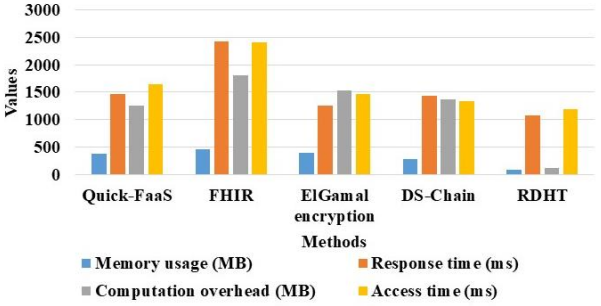


Figure. 9 Visual presentation of the comparative and proposed method's results

the presented RDHT method is recorded to exhibit a minimal memory usage of 89.68MB, response time of 1070.52ms, computation overhead of 124.56MB, and access time of 1187.12ms. On the contrary, the existing methods: Quick-FaaS, FHIR, ElGamal encryption, and DS-Chain, respectively exhibit a memory usage of 378MB, 464MB, 389MB and 278MB, response time of 1458.78ms, 2423.98ms, 1257.83ms and 1435.84ms, computation overhead of 1251.88MB, 1803.22MB, 1531.12MB and 1362.68MB, and access time of 1638.77ms, 2407.83ms, 1468.03ms and 1335.91ms. These outcomes clearly represent that the presented RDHT method exhibits minimal memory usage, response time, computation overhead and access time in comparison to the existing methods (Quick-FaaS, FHIR, ElGamal encryption, and DS-Chain).

4.3 Comparative analysis with discussion

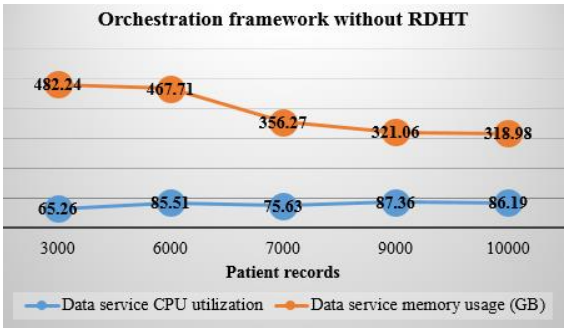
In this analysis, the efficacy of the presented orchestration framework with RDHT is contrasted by comparing its performance with the health monitoring workflow developed by El-Kassabi [25]. As discussed in the literature section, El-Kassabi [25] developed a workflow adaptation, monitoring, and orchestration model for performing automatic reconfiguration and for detecting performance degradation in order to ensure the workflow's QoS. In this technique, the adaptation and monitoring approaches were utilized for repairing and detecting errors, aside from triggering various adaptation actions such as resource scaling, migration, and workflow reconfiguration. Additionally, a model checker was used in the study for validating this presented workflow's outcomes by means of safety properties, liveness, and reachability. The extensive experimental evaluation carried out on the MIMIC-III dataset confirms the superiority of this health monitoring workflow. This workflow deploys on a Docker-swarm cluster and is examined by means of data service (CPU utilization and memory usage) and processing service (CPU utilization and memory usage).

As specified in tables 5 and 6, and Figs. 10 and 11, the developed orchestration framework with RDHT demands low CPU utilization and memory usage for both processing and data services related to the existing health monitoring workflow on the MIMIC-III dataset. In tables 5 and 6, the outcomes of the developed orchestration framework are validated with and without using RDHT.

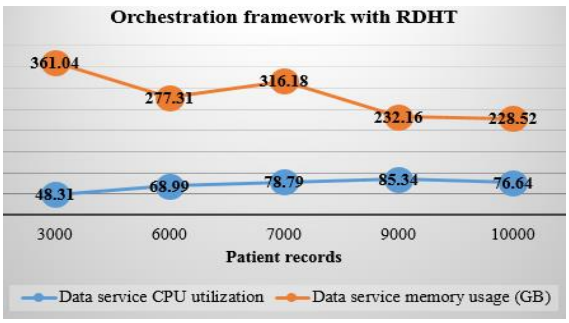
Tables 5 and 6 clearly demonstrate the effectiveness of using RDHT in the orchestration framework for secure storage and access of EHRs. The RDHT provides a fine-grained access control in the orchestration framework that allows different entities and users for accessing specific patient's EHRs. This is vital in the healthcare management system because in few cases, the sensitive information about the patient needs to be limited as per the permissions and roles. While the increasing number of EHRs, the RDHT significantly manages data distribution and accesses user's requests in a scalable way that guarantees the orchestration framework's responsiveness and performance.

Table 5. Obtained results of the proposed framework by means of data service CPU utilization and memory usage

Patient records	Orchestration framework without RDHT		Orchestration framework with RDHT	
	Data service CPU utilization	Data service memory usage (GB)	Data service CPU utilization	Data service memory usage (GB)
3000	65.26	482.24	48.31	361.04
6000	85.51	467.71	68.99	277.31
7000	75.63	356.27	78.79	316.18
9000	87.36	321.06	85.34	232.16
10000	86.19	318.98	76.64	228.52



(a)

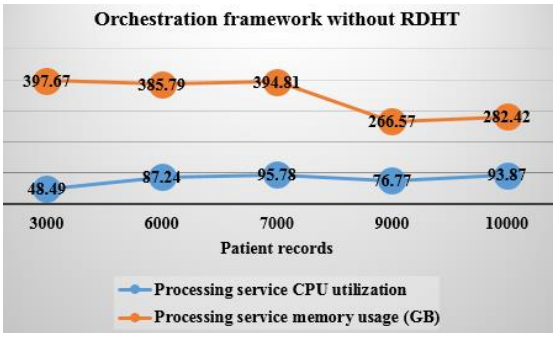


(b)

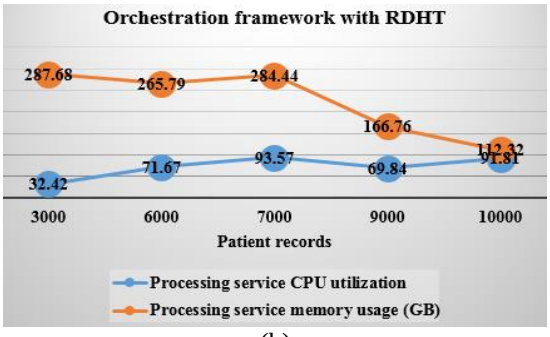
Figure. 10 Data service (CPU utilization and memory usage): (a) Orchestration framework without RDHT and (b) Orchestration framework with RDHT

Table 6. Obtained results of the proposed framework by means of processing service CPU utilization and memory usage

Patient records	Orchestration framework without RDHT		Orchestration framework with RDHT	
	Processing service CPU utilization	Processing service memory usage (GB)	Processing service CPU utilization	Processing service memory usage (GB)
3000	48.49	397.67	32.42	287.68
6000	87.24	385.79	71.67	265.79
7000	95.78	394.81	93.57	284.44
9000	76.77	266.57	69.84	166.76
10000	93.87	282.42	91.81	112.32



(a)



(b)

Figure. 11 Processing service (CPU utilization and memory usage): (a) Orchestration framework without RDHT and (b) Orchestration framework with RDHT

5. Conclusion

In this research study, a novel orchestration framework is developed based on the RDHT method for secure data storage and access in a multi-cloud environment. The presented orchestration framework comprises four important processes namely, FLR, OFG, ER, and OFE, as these processes generate a simplified user interface with reduced complexity and clutter. These four processes avoid unnecessary overhead and create a user-friendly experience. The performance of the suggested orchestration framework is validated using EHRs which are acquired from the web application related to hospital management system. In addition, a RDHT method is incorporated with the recommended orchestration framework for secure data storage and access by effectively distributing EHRs across different cloud providers. In this context, the efficacy of the RDHT method is investigated based on four evaluation metrics namely, memory usage, response time, computation overhead, and access time. The empirical analysis evidences that the RDHT method consumes a limited access time of 1187.12ms, response time of 1070.52ms, memory usage of 89.68MB, and computation overhead of 124.56MB than the existing methods like Quick-FaaS, FHIR, ElGamal encryption, and DS-Chain on the MIMIC-III dataset for 300 patient records. Additionally, the future work concentrates on enhancing the compatibility of cross-platform for ensuring that the recommended orchestration framework operates in different environments such as edge devices, on-premises infrastructures, and with various cloud providers.

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

Conceptualization, ZAA and DLN; methodology, ZAA; software, DLN; validation, DLN; formal analysis, DLN; investigation, ZAA; resources, DLN; data curation, ZAA; writing—original draft preparation, ZAA; writing—review and editing, DLN; visualization, ZAA; supervision, DLN; project administration, DLN

References

[1] C. Ramalingam, and P. Mohan, “Addressing Semantics Standards for Cloud Portability and Interoperability in Multi Cloud Environment”, *Symmetry*, Vol. 13, No. 2, p. 317, 2021.

[2] O. Tomarchio, D. Calcaterra, and G. D. Modica, “Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks”, *Journal of Cloud Computing: Advances, Systems and Applications*, Vol. 9, p. 49, 2020.

[3] K. Benhssayen, and A. Ettalbi, “Semantic interoperability framework for IAAS resources in multi-cloud environment”, *International*

Journal of Computer Science & Network Security, Vol. 21, No. 2, 2021.

[4] K. Benhssayen, and A. Ettalbi, "An Extended Framework for Semantic Interoperability in PaaS and IaaS Multi-cloud", In: *Proc. of International Conf on Digital Technologies and Applications*, pp. 415-424, 2022.

[5] T. Kaur, and K. Kaur, "TensorFlow-based semantic techniques for multi-cloud application portability and interoperability", In: *Proc. of G. Ranganathan, J. Chen, A. Rocha, (Eds.) Inventive Communication and Computational Technologies: Proceedings of ICICCT. 2019*, pp. 13-21, 2020.

[6] G. Cordasco, M. D. Auria, A. Negro, V. Scarano, and C. Spagnuolo, "Toward a domain-specific language for scientific workflow-based applications on multicloud system", *Concurrency and Computation: Practice and Experience*, Vol. 33, No. 18, p. e5802, 2021.

[7] G. Fatouros, Y. Poulakis, A. Polyviou, S. Tsarsitalidis, G. Makridis, J. Soldatos, G. Kousiouris, M. Filippakis, and D. Kyriazis, "Knowledge Graphs and interoperability techniques for hybrid-cloud deployment of FaaS applications", In: *Proc. of 2022 IEEE International Conf on Cloud Computing Technology and Science. (CloudCom)*, pp. 91-96, 2022.

[8] C. K. Dehury, P. Jakovits, S. N. Srirama, G. Giotis, and G. Garg, "TOSCAdata: Modeling data pipeline applications in TOSCA", *Journal of Systems and Software*, Vol. 186, p. 111164, 2022.

[9] A. Mujezinović, and V. Ljubović, "Serverless architecture for workflow scheduling with unconstrained execution environment", In: *Proc. of 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, pp. 242-246, 2019.

[10] N. Ravi, and M. Thangarathinam, "Emergence of Middleware to Mitigate the Challenges of Multi-Cloud Solutions onto Mobile Devices", *International Journal of Cooperative Information Systems*, Vol. 28, No. 04, p. 1950012, 2019.

[11] M. A. Serhani, H. T. E. Kassabi, K. Shuaib, A. N. Navaz, B. Benatallah, and A. Beheshti, "Self-adapting cloud services orchestration for fulfilling intensive sensory data-driven IoT workflows", *Future Generation Computer Systems*, Vol. 108, pp. 583-597, 2020.

[12] B. Shirazi, "Super-process interoperability optimization architecture in healthcare ultra-large-scale systems: A graph-based multi-objective approach", *Concurrency and Computation: Practice and Experience*, Vol. 34, No. 3, p. e6595, 2022.

[13] J. Han, S. Park, and J. Kim, "Dynamic OverCloud: Realizing Microservices-Based IoT-Cloud Service Composition over Multiple Clouds", *Electronics*, Vol. 9, p. 969, 2020.

[14] E. Zeydan, J. Baranda, and J. M. Bafalluy, "Post-Quantum Blockchain-Based Secure Service Orchestration in Multi-Cloud Networks", *IEEE Access*, Vol. 10, p. 129520-129530, 2022.

[15] D. Kim, H. Muhammad, E. Kim, S. Helal, and C. Lee, "TOSCA-Based and Federation-Aware Cloud Orchestration for Kubernetes Container Platform", *Applied Sciences*, Vol. 9, p. 191, 2019.

[16] D. Calcaterra, and O. Tomarchio, "Multi-faceted cloud portability with a TOSCA-based orchestrator", In: *Proc. of 2021 8th International Conf on Future Internet of Things and Cloud (FiCloud)*, Rome, Italy, pp. 326-333, 2021.

[17] J. Li, Y. Deng, W. Sun, W. Li, R. Li, Q. Li, and Z. Liu, "Resource orchestration of cloud-edge–based smart grid fault detection", *ACM Transactions on Sensor Networks (TOSN)*, Vol. 18, No. 3, p. 46, 2022.

[18] Q. Qi, J. Wang, Y. Cao, J. Wang, H. Sun, and J. Liao, "Cluster-PSO based resource orchestration for multi-task applications in vehicular cloud", *Wireless Personal Communications*, Vol. 102, No. 3, pp. 2133-2155, 2018.

[19] J. Shen, P. Zeng, K. K. R. Choo, and C. Li, "A Certificateless Provable Data Possession Scheme for Cloud-Based EHRs", *IEEE Transactions on Information Forensics and Security*, Vol. 18, pp. 1156-1168, 2023.

[20] G. P. Kanna, and V. Vasudevan, "An improved privacy aware secure multi-cloud model with proliferate ElGamal encryption for big data storage", *International Journal of Information and Computer Security*, Vol. 17, No. 1-2, pp. 1-20, 2022.

[21] R. Mishra, D. Ramesh, D. R. Edla, and L. Qi, "DS-Chain: A secure and auditable multi-cloud assisted EHR storage model on efficient deletable blockchain", *Journal of Industrial Information Integration*, Vol. 26, p. 100315, 2022.

[22] A. N. Gohar, S. A. Abdelmawgoud, and M. S. Farhan, "A Patient-Centric Healthcare Framework Reference Architecture for Better

Semantic Interoperability Based on Blockchain, Cloud, and IoT", *IEEE Access*, Vol. 10, pp. 92137-92157, 2022.

[23] L. Ouchaou, H. Nacer, and C. Labba, "Towards a distributed saas management system in a multi-cloud environment", *Cluster Computing*, Vol. 25, No. 6, pp. 4051-4071, 2022.

[24] P. Rodrigues, F. Freitas, and J. Simão, "QuickFaaS: Providing Portability and Interoperability between FaaS Platforms", *Future Internet*, Vol. 14, p. 360, 2022.

[25] H. T. E. Kassabi, M. A. Serhani, R. Dssouli, and A. N. Navaz, "Trust enforcement through self-adapting cloud workflow orchestration", *Future Generation Computer Systems*, Vol. 97, pp. 462-481, 2019.

[26] N. Iqbal, S. I. Ahmad, R. Ahmad, and D. H. Kim, "A Scheduling Mechanism Based on Optimization Using IoT-Tasks Orchestration for Efficient Patient Health Monitoring", *Sensors*, Vol. 21, p. 5430, 2021.

[27] R. M. Vozmediano, R. S. Montero, E. Huedo, and I. M. Llorente, "Orchestrating the deployment of high availability services on multi-zone and multi-cloud scenarios", *Journal of Grid Computing*, Vol. 16, pp. 39-53, 2018.

[28] J. Okwuibe, J. Haavisto, I. Kovacevic, E. Harjula, I. Ahmad, J. Islam, and M. Ylianttila, "SDN-enabled resource orchestration for industrial iot in collaborative edge-cloud networks", *IEEE Access*, Vol. 9, pp. 115839-115854, 2021.

[29] H. Qin, M. Yu, Y. Lai, Z. Liu, and J. Liu, "Cloud resource orchestration optimisation based on ARIMA", *International Journal of Simulation and Process Modelling*, Vol. 14, No. 5, pp. 420-430, 2019.

[30] M. Kazim, L. Liu, and S. Y. Zhu, "A framework for orchestrating secure and dynamic access of IoT services in multi-cloud environments", *IEEE Access*, Vol. 6, pp. 58619-58633, 2018.

[31] R. Mishra, I. Kaur, S. Sahu, S. Saxena, N. Malsa, and M. Narwaria, "Establishing three layer architecture to improve interoperability in Medicare using smart and strategic API led integration", *SoftwareX*, Vol. 22, p. 101376, 2023.

[32] H. Ghayvat, S. Pandya, P. Bhattacharya, M. Zuhair, M. Rashid, S. Hakak, and K. Dev, "CP-BDHCA: Blockchain-based Confidentiality-Privacy preserving Big Data scheme for healthcare clouds and applications", *IEEE Journal of Biomedical and Health Informatics*, Vol. 26, No. 5, pp. 1937-1948, 2022.

[33] S. Bebortta, S. S. Tripathy, S. Basheer, and C. L. Chowdhary, "FedEHR: A Federated Learning Approach towards the Prediction of Heart Diseases in IoT-Based Electronic Health Records", *Diagnostics*, Vol. 13, p. 3166, 2023.

[34] R. Oruche, E. Milman, M. L. Alarcon, X. Cheng, A. Pandey, S. Wang, P. Calyam, and K. Kee, "Science gateway adoption using plug-in middleware for evidence-based healthcare data management", *Concurrency and Computation: Practice and Experience*, Vol. 35, No. 18, p. e7195, 2023.

[35] R. Hossen, M. Whaiduzzaman, M. N. Uddin, M. J. Islam, N. Faruqui, A. Barros, M. Sookhak, and M. J. N. Mahi, "BDPS: An Efficient Spark-Based Big Data Processing Scheme for Cloud Fog-IoT Orchestration", *Information*, Vol. 12, p. 517, 2021.

[36] E. Zeljković, T. D. Schepper, P. Bosch, I. Vermeulen, J. Haxhibeqiri, J. Hoebeke, J. Famaey, and S. Latré, "ORCHESTRA: Virtualized and programmable orchestration of heterogeneous WLANs", In: *Proc. of 2017 13th International Conf on Network and Service Management (CNSM)*, Tokyo, Japan, pp. 1-9, 2018.

[37] P. Ren, L. Liu, X. Qiao, and J. Chen, "Distributed Edge System Orchestration for Web-Based Mobile Augmented Reality Services", *IEEE Transactions on Services Computing*, Vol. 16, No. 3, pp. 1778-1792, 2023.

[38] H. Tian, Y. Chen, C. C. Chang, H. Jiang, Y. Huang, Y. Chen, and J. Liu, "Dynamic-hash-table based public auditing for secure cloud storage", *IEEE Transactions on Services Computing*, Vol. 10, No. 5, pp. 701-714, 2017.

[39] J. Wang, D. Liu, X. Fu, F. Xiao, and C. Tian, "DHash: Dynamic Hash Tables with Non-blocking Regular Operations", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 33, No. 12, pp. 3274-3290, 2022.

[40] X. Wang, and L. Liu, "Image encryption based on hash table scrambling and DNA substitution", *IEEE Access*, Vol. 8, pp. 68533-68547, 2020.

[41] P. Kumar, M. Rahman, S. Namasudra, and N.R. Moparthi, "Enhancing security of medical images using deep learning, chaotic map, and hash table", *Mobile Networks and Applications*, 2023.

[42] M. Scherpf, F. Gräßer, H. Malberg, and S. Zaunseder, "Predicting sepsis with a recurrent neural network using the MIMIC III database", *Computers in Biology and Medicine*, Vol. 113, p. 103395, 2019.

[43] S.R. Khope, and S. Elias, "Critical correlation of predictors for an efficient risk prediction framework of ICU patient using correlation and transformation of MIMIC-III dataset", *Data Science and Engineering*, Vol. 7, No. 1, pp. 71-86, 2022.

[44] Y. Zhu, J. Zhang, G. Wang, R. Yao, C. Ren, G. Chen, X. Jin, J. Guo, S. Liu, H. Zheng, and Y. Chen, "Machine learning prediction models for mechanically ventilated patients: analyses of the MIMIC-III database", *Frontiers in Medicine*, Vol. 8, p. 662340, 2021.

[45] A. Budrionis, M. Miara, P. Miara, S. Wilk, and J.G. Bellika, "Benchmarking pysyft federated learning framework on mimic-iii dataset", *IEEE Access*, Vol. 9, pp. 116869-116878, 2021.