Online-Journals.org

iJIM International Journal of Interactive Mobile Technologies

iJIM | eISSN: 1865-7923 | Vol. 17 No. 19 (2023) | OPEN ACCESS

https://doi.org/10.3991/ijim.v17i19.42029

PAPER

Optimizing Clustering Approaches in Cloud Environments

Abdel-Rahman Al-Ghuwairi[1], Dimah Al-Fraihat[2](✉), Yousef Sharrab[3], Yazeed Kreishan[1], Ayoub Alsarhan[4], Hasan Idhaim[5], Ayman Qahmash[6]

[1]Department of Software Engineering, Faculty of Prince Al-Hussein Bin Abdallah II for Information Technology, The Hashemite University, Zarqa, Jordan

[2]Department of Software Engineering, Faculty of Information Technology, Isra University, Amman, Jordan

[3]Department of Data Science and Artificial Intelligence, Faculty of Information Technology, Isra University, Amman, Jordan

[4]Department of Information Technology, Faculty of Prince Al-Hussein Bin Abdallah II for Information Technology, The Hashemite University, Zarqa, Jordan

[5]Department of Information Systems, Faculty of Prince Al-Hussein Bin Abdallah II for Information Technology, The Hashemite University, Zarqa, Jordan

[6]Department of Information Systems, Computer Science College, King Khalid University, Abha, Saudi Arabia

d.fraihat@iu.edu.jo

ABSTRACT

This study focuses on the challenge of developing abstract models to differentiate various cloud resources. It explores the advancements in cloud products that offer specialized services to meet specific external needs. The study proposes a new approach to request processing in clusters, improving downtime, load distribution, and overall performance. A comparison of three clustering approaches is conducted: local single cluster, local multiple clusters, and multiple cloud clusters. Performance, scalability, fault tolerance, resource allocation, availability, and cost-effectiveness are evaluated through experiments with 50 requests. All three approaches achieve a 100% success rate, but processing times vary. The local single cluster has the longest duration, while the local multiple clusters and multiple cloud clusters perform better and offer faster processing, scalability, fault tolerance, and availability. From a cost perspective, the local single cluster and local multiple clusters incur capital and operational expenses, while the multiple cloud clusters follow a pay-as-you-go model. Overall, the local multiple clusters and multiple cloud clusters outperform the local single cluster in terms of performance, scalability, fault tolerance, resource allocation, availability, and cost-effectiveness. These findings provide valuable insights for selecting appropriate clustering strategies in cloud environments.

KEYWORDS

cloud computing, load distribution, clustering approaches, performance analysis, multiple cloud clusters, Node.js

1 INTRODUCTION

The advancements made in cloud computing have altered the way users access distant resources easier than ever before. Due to this shift in how things are done, Cloud Service Providers' (CSPs) role is becoming more vital with time as they continue providing an increasing number of innovative solutions for customers worldwide [1]. The total value of cloud computing is predicted to increase significantly, from $141 billion to $495 billion, by 2022 [2]. This significant growth has prompted studies investigating

Al-Ghuwairi, A.R., Al-Fraihat, D., Sharrab, Y., Kreishan, Y., Alsarhan, A., Idhaim, H., Qahmash, A. (2023). Optimizing Clustering Approaches in Cloud Environments. *International Journal of Interactive Mobile Technologies (iJIM)*, 17(19), pp. 70–94. https://doi.org/10.3991/ijim.v17i19.42029

Article submitted 2023-06-05. Revision uploaded 2023-07-23. Final acceptance 2023-07-25.

its effectiveness and anticipating increased integration in the future [3]. Cloud computing acts as a solution offering off-premises computing power to users wishing to access tools and retain data, all facilitated by CSPs from a location in the cloud and accessible through the internet [4]. Relatedly, CSPs provide these resources using an "X as a Service" (XaaS) model including Software (SaaS), Infrastructure (IaaS), or Platform (PaaS) [5]. Given these options, cloud services are available via public, private, or hybrid cloud models depending on user requirements. Currently, CSPs range from industry tycoons such as Amazon, Google, IBM, and Microsoft, providing numerous services tailored towards specific clients based on their needs [6].

Cloud computing has emerged as a transformative technology, enabling the utility model of computing to serve clients worldwide. IT and business resources such as servers, storage, networks, and applications can be dynamically delivered to clients across the globe. The adoption of clusters in the cloud market has significantly increased, providing an opportunity to enhance clusters' capabilities to accommodate growing client requirements on the nodes [7].

A multi-cluster system refers to a distributed architecture that comprises multiple clusters, with each cluster consisting of several nodes capable of independent computation and data storage. These systems find application in various domains such as high-performance computing, data analytics, and cloud computing [8]. Compared to traditional single-cluster systems, multi-cluster systems offer notable advantages, including enhanced scalability, reliability, and fault tolerance. Consequently, they have gained significant attention in research and development, with the potential to revolutionize large-scale computing tasks [9].

However, the adoption of clusters in the cloud environment poses several challenges. These challenges include load balancing within a single cluster, managing fluctuating service demands, evaluating the trustworthiness of new clients, and addressing scalability and performance issues [10]. Scalability, defined as the ability to allocate appropriate computing resources to clients dynamically, is crucial in cloud environments. Failing to provide flexible services and scale computing resources can result in the deficient performance of the server [11]. To effectively manage resources in the cloud market, cloud service providers must handle a growing number of client requests while ensuring optimal performance [12].

Efficient management of server requests is essential in maintaining optimal functioning, building customer trust, and seamless service delivery. In addressing this aspect, this study emphasizes enhancing server capacity as a significant factor in timely response to user requests. Therefore, it suggests adopting a novel approach by implementing server clustering as an effective means of workload distribution. Each cluster has a designated head with oversight responsibilities for its operation.

Our research endeavors are aimed at improving cluster performance by enhancing its ability to meet growing client demands. Proposed as part of this effort, is an innovative architectural environment which divides a single cluster into smaller individual clusters for optimal request processing efficiencies in nodes, and minimal delays or downtime resulting from the process itself.

The proposed approach has been successfully implemented through an experimental setup utilizing Node.js programming language. Before conducting research experiments, performance metrics were compared for which results indicated that cloud computing can facilitate clustering, thereby offering scalability and flexibility crucial for optimizing services provided to clients.

With the aim of boosting efficiency and saving time when processing servers, the study proposes dividing one large cluster into several smaller ones, each complete with its own back up-server. Consequently, improving server processing times became achievable and may lead researchers to focus their investigation on this precise aspect.

The experimental findings were meticulously reviewed to gauge how effective the proposed approach is. To do so, performance metrics were compared pre- and post-application implementation under specific consideration of time wasted during request processing. Our anticipated result is an enhancement in server scalability and efficiency achieved via cluster division, which should translate into superior overall cluster output.

To optimize resource usage while delivering excellent service quality, we conducted this research with specific objectives, which include streamlining request processing and enhancing cluster robustness through an innovative architectural approach that divides clusters strategically. We anticipate that reducing time wastage during node processing coupled with improving server performance should yield the desired results and meet the expanding demands of customers effectively. Overall, these findings should assist us in meeting clients' evolving expectations as well as maintaining superior service delivery standards.

2 RESEARCH BACKGROUND

This section presents an overview of cluster computing, single core clusters, and multi core clusters which have delivered significant enhancements in performance levels. Achieving optimal results requires a clear appreciation for application behavior patterns and trends. This section outlines distinctions between traditional single-core clusters compared with those containing multiple cores. Our research identifies specific challenges when processing requests in nodes across a multi cluster core, whilst promoting its benefits.

2.1 Cluster computing

When several computers are connected to work together seamlessly as one unit, this is referred to as cluster computing [13]. IBM introduced this concept in the 1960s as a viable alternative for interconnecting massive mainframe computers with cost-effectiveness as its core value proposition [14]. Recently, cluster computing has garnered exceptional attention following advancements such as efficient microprocessors, high-speed networks, and pivotal tools capable of improving distributed computing performance available since the '80s onwards [15].

Recent technological advancements have paved the way for cost-effective parallelization solutions such as clusters, making them increasingly popular options across all sectors of computing, including high-performance applications where high-throughput and reliability are critical [16]. A computer cluster utilizes collections of interconnected computers combined collaboratively to achieve better computational processing efficiency than traditional single device platforms [17]. A cluster is defined as a collection of nodes, each having independent control over stand-alone workloads while collaborating in real-time with others via high-speed local-area networks [18]. Clusters allow executing heavy-duty tasks that are impractical on single machines [19]. The nodes can vary in number, and they incorporate memory units alongside comprehensive operating systems dependent on what specifications best suit their intended use [20]. The main system components include individual machines interlinked via fast interconnects, in addition to software features geared towards enabling maximized performance throughput during parallel execution operations, ensuring prompt task delivery [21].

Cluster computing is an effective approach of achieving improved availability rates in addition to better performance while keeping expenses minimized,

compared with individual computers [22]. Nevertheless, several significant drawbacks need to be factored in [23]. For example, establishing a cluster entails complexities requiring multidisciplinary knowledge. Other complications include scalability limits and communication overheads which together present significant synchronization challenges that disrupt efficient operation levels, significantly compromising overall system effectiveness. Further, factors such as fault tolerance coupled with reliability must be managed carefully when configuring the cluster. Moreover, the overall purchase costs, infrastructure prerequisites, and maintenance charges are all important aspects to consider [24]. Hence, organizations need to consider these challenges against their own specific performance priorities before deciding whether to invest in cluster computing services [25].

2.2 Single-core clusters

A single-core cluster includes nodes, each equipped with a single processor and a single core. Standardization is a key characteristic of such clusters, ensuring that nodes are similar in terms of memory, cache, and server connection [26]. Further, caches, which are storage locations for active data, play a crucial role in reducing latency, improving access times, and enhancing overall efficiency [27].

In the context of a common algorithm like the Message Passing Interface (MPI), a cluster composed of single-processor nodes can execute the algorithm independently. The performance of a single-core cluster relies on the processor's frequency [28]. Theoretically, incorporating multiple single-core processors onto a single chip could double the performance. However, the average speed of each core is slower than the fastest single-core processor due to communication delays at various levels of the cluster's communication link [29].

2.3 Multi-core clusters

The development of cloud computing systems, also known as multicore clusters, has been driven by the convergence of high-performance computing technology and high-speed connections [30]. In the past, clusters utilized multiple single-core processors. However, the industry has now introduced chips with multiple processors, or multi-cores, to address the limitations of single-core clusters [31]. A comprehensive understanding of multi-core chips necessitates acknowledging that individual processor cores operate at slower speeds in comparison to single-core processors [32]. However, when multiple cores collaborate on a single chip, higher data processing rates can be achieved [33]. With each new chip generation, we can anticipate an increase in the number of cores accompanied by reduced processing time. Notably, multi-core clusters exhibit a hierarchical storage structure where cache memory is shared among processor cores [34]. This implies that processors within the same node share main memory, while those from different nodes do not. To achieve optimal efficiency, parallel programming is recommended, with task allocation based on application communication patterns and system characteristics carefully considered [35].

The demand for multi core clusters has grown significantly due to their various benefits. The numerous advantages offered by multi core clusters have contributed to their growing popularity in recent years. Firstly, multi-threaded software enables the utilization of multi core technology, which can execute many tasks simultaneously and enhance overall system performance and efficiency by freeing up resources that were

previously tied up in managing multiple processors [36]. Secondly, multi-core clusters offer simple scalability as an essential feature. Organizations can add more cores or nodes to these clusters easily for enhanced computing power requirements when needed [37]. This flexibility provides a reliable way to handle computational demands while ensuring efficient operations at all levels of usage over time with minimum room for error from additional overhead tasks such as cooling equipment required in large data centers; hence less heat generation and lower energy consumption [38].

2.4 Node.js clustering

A feature embedded within the Node.js runtime environment is its unique clustering function. Through its utilization of child processes, it streamlines incoming requests management and enables task distribution within applications. This strategy can effectively use the capabilities of numerous CPU cores in multi-core systems enhancing scalability and boosting performance for Node.js applications [39].

Node.js operates using a cluster system which involves assigning a master process to manage multiple worker processes. Every slave runs its instance of the Node.js event loop to undertake operations. While distributing incoming connections or tasks, a load-balancing algorithm is implemented by the master process. This guarantees that each CPU core shares an equal workload across all slaves and smoothly completes assigned tasks [40].

Node.js clustering offers numerous primary benefits, including better performance and increased throughput for applications. Multiple cores are utilized, effectively allowing for greater processing power and speedy handling of concurrent requests while making good use of all available system resources efficiently. Additionally, this technique enhances resilience, so that even if a slave process fails or crashes within clustered deployment setup, it will not affect overall availability negatively due to some inbuilt error management mechanisms and security features that are factored into its design and triggered to action immediately, maintaining a stable environment at unprecedented scales [41].

Complexities that arise from attempting to build scalable applications in Node.js involve managing multiple processes and load balancing. Node.js clustering provides a simple solution by removing these hurdles altogether. This functionality proves to be particularly valuable when developing high performance web servers or real time applications required to manage an extensive number of concurrent connections [42].

Typically, Node.js operates on one thread that employs just one CPU core, irrespective of whether numerous cores exist in the system or not. Nevertheless, overcoming this predicament while improving performance calls for transitioning operations towards a multi core strategy utilizing clustering instead. The clustering method involves creating multiple Node.js procedures, collectively known as worker nodes designed to work simultaneously over an identical server port caliber through Inter Process Communication (IPC). With this setup, automatic workload balancing can be achieved such that, whenever any process manages resource-centric tasks, other secondary processors proffer an additional request processing power using spare CPUs [43].

Node.js has gained popularity among developers who strive to cluster tasks efficiently due to its use of JavaScript language, which opens up possibilities for optimization [44]. This ability can be advantageously employed in intrusion detection or protection schemes for cloud markets that depend heavily on traffic data from the marketplace's various nodes. Adequate interpretation and analysis enable one to evaluate a more extensive range of judgments regarding trends in activity levels

that influence the larger picture around investments in cloud markets. Designing an effective intrusion detection scheme typically entails several phases, which include initial sensing strategies; allowing to assure the cloud environment remains secure with suitable arrangements; intrusion prevention itself by incorporating proper policies like access controls into an IT system's architecture; performing behavior analyses on network systems that allow effective prediction; and responsibility management plan that maps out how well security incidents are managed where necessary roles are evidently defined, leveraging resources effectively [45].

3 RELATED WORK

Numerous studies have delved into enhancing techniques specifically tailored for cloud environments. In this section, we provide a review of research conducted in this domain emphasizing their contributions.

A pioneering study in this domain was conducted by [46]. They put forth an approach called the single cluster methodology to handle requests in cloud environments. The primary objective of their research revolved around enhancing downtime load distribution and overall system performance. Although their local single cluster approach demonstrated a success rate of 100% in request processing, it was observed to have limitations in terms of processing time. The authors acknowledged the necessity for methodologies that could provide expedited processing times.

To overcome the limitations of the cluster method using local single cluster approach, the researchers of [47] introduced the use of multiple clusters method. Their research focused on improving performance, scalability, fault tolerance, resource allocation and availability, in cloud environments. They conducted experiments involving 50 requests and discovered that the local multiple clusters approach exhibited processing time, scalability, fault tolerance and availability compared to the local single cluster approach. However, one aspect that was not explicitly considered during the study was the cost effectiveness of the proposed method. This gap in understanding prevents us from comprehending the implications associated with the suggested approach.

There has been a growing interest in utilizing cloud clusters to enhance clustering methods. The authors of [48] conducted a comparison of clustering approaches including employing a locally single cluster, locally multiple clusters, and harnessing multiple cloud clusters. They conducted experiments involving 50 requests to evaluate performance, scalability, fault tolerance, resource allocation, availability, and cost effectiveness. The outcomes revealed that both the use of clusters locally and multiple cloud clusters surpassed the local cluster approach in terms of several metrics. Notably, the multiple cloud clusters approach offered benefits such as processing, scalability, fault tolerance, resource allocation, availability and cost effectiveness due to its pay-as-you-go model. However, the study did not delve into analyzing the limitations or weaknesses of the proposed approaches, leaving an opportunity for further exploration.

Expanding on previous research, [49] proposed a clustering approach that blends local multiple clusters with multiple cloud clusters. Their objective was to enhance performance fault tolerance and cost effectiveness while considering the effectiveness. By conducting experiments and simulations, they successfully showcased that the hybrid approach outperformed clustering methods. This hybrid approach effectively leveraged the resources of clusters and the scalability of cloud clusters leading to performance fault tolerance and cost effectiveness. However, it is worth noting that the study primarily focused on performance and cost effectiveness without delving into the impact on metrics, like resource allocation or availability.

Another study of [50] investigated the application of machine learning techniques to optimize clustering approaches in cloud environments. The researchers proposed a predictive model that utilizes historical data to dynamically allocate resources in clusters based on workload patterns. Their study showed promising results in terms of performance improvement and resource utilization optimization. However, the study primarily focused on the performance aspect and did not comprehensively evaluate other factors such as fault tolerance or cost-effectiveness.

The study presented by [51] introduced an algorithm focused on load balancing for clustering in cloud environments. Their objective was to optimize resource utilization and minimize response time by distributing workloads among clusters based on their capacities and current utilization levels. The experimental findings indicated that their algorithm successfully achieved workload balance, resulting in improved performance and reduced response time.

The authors of [52] tackled the issue of handling faults in methods in cloud environments. Their proposed algorithm aimed to enhance system reliability and availability by identifying and recovering from node failures. To achieve this, they incorporated nodes and deployed fault detection mechanisms to maintain uninterrupted operation. The experimental evaluations showcased that their fault tolerant clustering algorithm remarkably improved system reliability while minimizing downtime. Nevertheless, the study did not explore its effects on measures such as performance.

The research conducted by [53] examined the impact of methods on energy efficiency within cloud environments. The study aimed to reduce energy consumption while ensuring performance levels. The researchers introduced a clustering algorithm that incorporated energy awareness, dynamically adapting resource allocation according to workload patterns and system conditions. The experimental findings indicated energy savings without compromising performance.

Considering the cost-effectiveness aspect, [54] conducted a study that analyzed the trade-offs between performance and cost in different clustering approaches. They proposed a cost-performance model that considers factors such as processing time, scalability, and resource allocation efficiency. Through experiments and simulations, they evaluated the cost-performance trade-offs of local single clusters, local multiple clusters, and multiple cloud clusters. The findings showed that selecting the clustering method relies on the unique features of the workload and the limitations imposed by cost considerations. Nonetheless, the research did not extensively delve into factors, like fault tolerance or availability.

The cost effectiveness aspect was explored in a study conducted by [55]. They investigated the balance between performance and cost in approaches. In their research, they proposed a model that examined factors such as processing time, scalability, and resource allocation efficiency, aiming to assess the trade-offs in cost performance. Through experiments and simulations, they evaluated the cost performance trade-offs of clusters, local multiple clusters, and multiple cloud clusters. The results indicated that the choice of clustering approach should consider workload characteristics and cost limitations.

Prior research has extensively examined methods within cloud environments. These methods encompass clusters, local multiple clusters, and multiple cloud clusters. While these studies have made strides in terms of enhancing performance, scalability, fault tolerance, resource allocation, availability, and cost effectiveness, there remain drawbacks that necessitate attention. The local single cluster approach exhibits its limitations pertaining to processing time. On the hand, the local multiple clusters approach lacks an analysis of cost effectiveness. Moreover, further investigation is required to uncover weaknesses in the multiple cloud clusters approach [56, 57].

Recent research endeavors have proposed approaches that incorporate machine learning techniques to tackle these challenges [58]. However, further exploration is needed to evaluate their impact across metrics and address the limitations inherent in existing clustering approaches [59, 60].

The current research aims to improve cluster performance to meet the growing demands of clients. The motivation behind this endeavor is to enhance the efficiency of request processing in clusters while minimizing delays and downtime. The major contribution of this research lies in improving cluster performance to meet growing client demands. It proposes an innovative architectural environment that divides a single cluster into smaller ones, optimizing request processing efficiencies and minimizing delays. By dividing clusters into ones equipped with backup servers, the study effectively improves server processing times. The experimental findings successfully validate the proposed approach leading to enhanced server scalability and efficiency. Furthermore, the research prioritizes streamlining request processing, reinforcing cluster resilience and optimizing resource utilization to ensure service quality and meet the expanding demands of customers. In summary, it offers insights into optimizing cluster performance and meeting the evolving expectations of clients.

4 RESEARCH METHODS

This research aims to investigate the behavior of a cluster under a specific test scenario where the number of requests has increased. A comparison is made with the multi-cluster method to evaluate performance improvements, high availability, load balancing, and reduction in response time, allowing the system to handle a higher volume of requests.

To conduct the experiment, the researchers utilized k6, a developer-focused, open-source load-testing tool known for its productivity in performance testing. The implementation of k6 allowed for the anticipation of performance degradation and the prompt identification of problems, enabling a proactive approach in the development of resilient systems and robust applications. The user-friendly nature of k6, as well as its utilization of JavaScript, proved to be valuable for the effective implementation of tests in this study.

The conventional approach in Node.js for managing incoming client requests involves queuing them in a single thread through its Event Queue system. However, our research aimed to explore alternative methods that could potentially enhance the efficiency of adopting an event-driven architecture. By conducting extensive testing and experimentation, we investigated the feasibility of utilizing the Event Loop not only for event listening but also as an infinite loop for data processing, thereby opening up new possibilities and potential improvements in the handling of requests within the Node.js.

This study adopted an inventive technique to expedite request processing within Node.js software by eliminating I/O blocking. Our results showed that such an overhaul greatly ameliorates time efficiency in the system. To verify its efficacy, we performed comparisons between two distinct groups—one with a single thread while another utilizing worker threads based on CPU capacities—and analyzed their relative outputs thoroughly.

In this research, we utilized a system that incorporated eight cores by creating eight Node.js instances, each designed with its independent event loop. We configured the program to operate effectively on one port (PORT 3002). Our implementation necessitated that we create several worker processes; hence we relied on an intelligent strategy deployed by our master process to handle connecting incoming traffic and distributing incoming ones among our various workers evenly. We utilized a

powerful module named the Worker Threads in Node.js because of its proven ability to carry out CPU-intensive JavaScript tasks.

To assess the efficacy and efficiency of distinct clustering scenarios, our method entails carrying out two experiments. Specifically, experiment one will involve employing single clustering while experiment two will employ multi clustering. Experiment one tackles the typical means of utilizing a solo clustering configuration through clustered nodes running on only one server or computing device. Researchers endeavor to evaluate this method's efficiency and limitations by analyzing various performance metrics such as response time, throughput, and resource utilization. Table 1 summarizes the metrics used for comparison and their definitions.

Table 1. The metrics used for comparison between the two experiments

#	Metrics	Definition
1	HTTP req connecting	This refers to the process of establishing a connection between the client (usually a web browser or an application) and the server that hosts the requested resource. It involves establishing a TCP connection.
2	HTTP req duration	This is the time it takes for an HTTP request to complete, starting from the moment the request is sent to the server until the response is received.
3	Expected response true	This indicates that you are expecting a successful response from the server. In the context of load testing or automated testing.
4	HTTP req failed	This means that the HTTP request was not successful. It could be due to various reasons, such as a server error, network issue, or an invalid request.
5	HTTP req receiving	This refers to the process of the client receiving the response from the server after sending an HTTP request. It involves receiving and reading the data sent by the server.
6	HTTP req sending	This is the process of the client sending an HTTP request to the server.
7	HTTP req handshaking	Handshaking typically refers to the initial communication between the client and server to establish the parameters of the connection, such as the supported protocols and encryption methods. In the context of HTTP, it can refer to the establishment of a TCP connection.
8	HTTP req waiting	This refers to the time spent by the client waiting for a response from the server after sending an HTTP request. It could be due to various factors, including network latency, server processing time, or server-side delays.
9	Iterations	This refers to the number of times a specific action or task is repeated. In load testing, it typically represents the number of iterations or cycles of sending requests and receiving responses.
10	Iteration duration	In the context of load testing, an iteration refers to a complete cycle of sending an HTTP request and receiving the corresponding response. The iteration duration is the time it takes to complete one iteration.
11	Vus	Vus stands for "virtual users". In the context of load testing, a virtual user simulates a single user interacting with the system under test. The number of virtual users represents the concurrency or simultaneous user load applied during the test.

Using a multi-cluster strategy is the focus of our second experiment. The purpose is to determine its advantages in optimizing application performance when numerous copies are deployed on multiple computing devices or servers that form a cluster through efficient load balancing capabilities. Our goal is to evaluate whether this design results in scalable operations and improves fault tolerance when assessing performance metric outcomes in comparison with experiment one's outputs.

Our research method adheres to standardized procedures to ensure the validity and reliability of our findings regarding the behavior of clustering scenarios under various parameters. To achieve this, we establish controlled experimental environments for each scenario, where we deploy applications and simulate diverse workloads while collecting relevant data. To assess the performance of these scenarios effectively, we employ robust techniques for measuring relevant metrics. Through statistical analysis of these measurements, we can effectively compare the efficiencies, scalability, and

overall performances of different clustering scenarios. This approach enables us to generate meaningful insights and draw reliable conclusions from our research.

By conducting these experiments, valuable insights can be gained regarding the strengths and limitations of single clustering, the benefits of multi clustering, and the advantages of leveraging cloud-based multi-cluster architecture. The findings will contribute to the understanding of cluster computing and assist in making informed decisions when choosing the most suitable clustering approach for specific application requirements.

5 EXPERIMENTS AND RESULTS

5.1 Load balancing with Bluster mode

Improving operational efficiency is essential to ensure the smooth running of Node.js applications by running an optimized workflow that can be achieved through load balancing with Bluster mode using Process Manager2 (PM2). PM2 has dramatically simplified the process by offering core features such as process management and batching functions to horizontal load balancing capabilities and non-stop reload allowing for easy and seamless application control. The use of PM2's user-friendly interface ensures hassle-free starts, stops, and restarts of the system while providing a centralized management platform for users to constantly monitor resource consumption levels.

Automatic reloading during application updates or deployment with near-zero downtimes that feature in PM2's Bluster Mode allows you to maintain continuity without interruptions whilst ensuring maximum efficiency achieved through optimization efforts like status monitoring. PM2 offers various key performance indicators monitored uniformly and displayed effectively, allowing quick identification and resolution of performance-related issues such as CPU usage rate, memory consumption rates, request throughput based on how many requests are processed per second, worker status in clustered apps amongst other vital KPIs, which helps optimize your systems further. Figure 1 shows detailed metrics resulting from adopting PM2 during operations.

Furthermore, PM2 along with bluster-mode offer development teams a fast-track route towards operational excellence by improving system uptime through simplified workflows encouraging productivity growth.

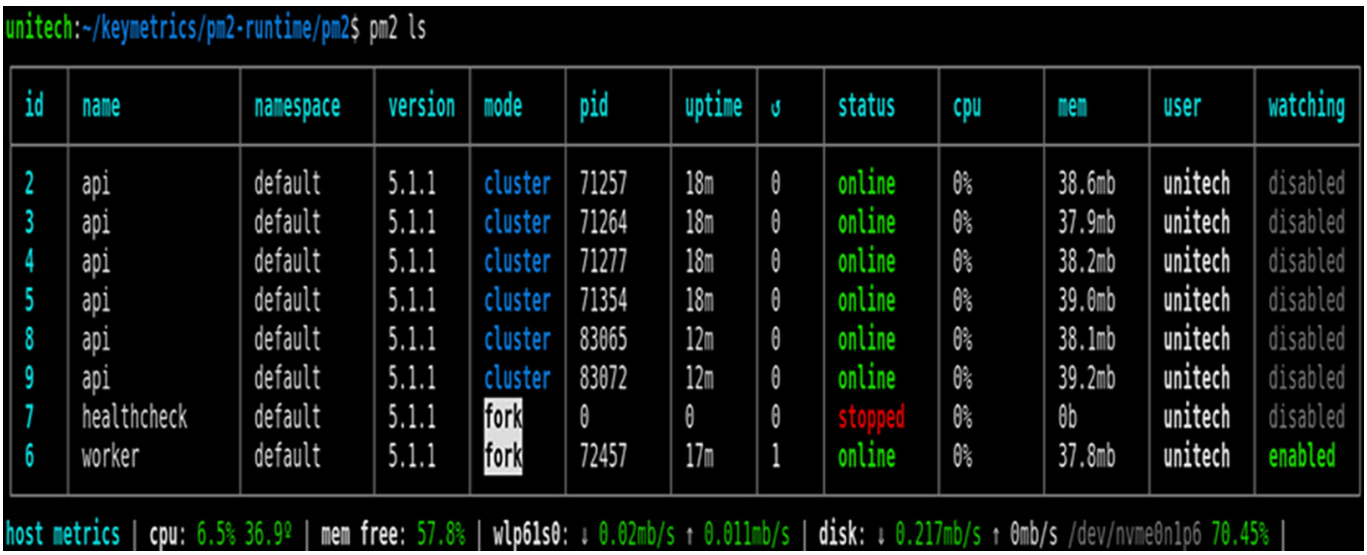

Fig. 1. Key metrics in PM2

5.2 Cluster mode: Node.js load balancing and zero downtime reload

In optimizing Node.js applications for peak performance and high availability, cluster mode is an essential feature worth exploring for efficient resource utilization. The master process manages several worker processes that balance incoming requests evenly, leading to better overall app performance. Cluster mode optimizes resource utilization, allowing applications to handle a larger volume of client requests effectively by spreading out operations evenly across processes, ensuring maximum efficiency. Another significant advantage is the zero downtime reload capability, whereby updating or reloading an application does not disrupt incoming requests due to master process management skills.

By taking advantage of the benefits of multiprocessing such as faster rebalancing of sockets if errors are not handled appropriately, developers ensure that robust applications are built with increased scalability and improved overall performance.

In cluster mode, two configurations exist: single- and multi-node configurations. Running on a solo machine per core might cause some limitations in terms of scalability due to backup constraints but running several instances on multiple machines called nodes provides better load balancing capabilities, enabling effort distribution among various tasks and leading to parallel processing advantage in multi nodes setups for maximum efficiency optimization.

For better scalability, increased performance rates along with optimal fault tolerance levels and utilizing multiple computing abilities made possible via Node.js's Cluster Mode becomes inevitable. By tactfully distributing workloads among different nodes, more power is given to the application to execute tasks seamlessly, thereby effectively handling a higher number of concurrent requests. Load balancing algorithms are then brought into play here to spread requests evenly among nodes—the aim being to optimize resource utilization while preventing any node from being overloaded. With the cluster mode already in place, recovery from any unhandled error or reloading of resources is swift and easy through faster socket rebalancing.

With the clustering arrangement, downtime rarely occurs—with other highly functional nodes continuing performance as usual in cases where there are hardware failures or resource mismanagement on a particular node. In summary, parallel processing, an ideal load balancing system, and quenched downtimes are enabled utilizing Node.js's cluster mode.

The experimental setup and procedure for the two conducted experiments are outlined as follows: Experiment 1 using single clustering and Experiment 2 using multi clustering.

5.3 Experiment 1: Local single clustering

In this scenario, the following steps were performed to evaluate the performance of single clustering on-premise:

1. Installation of Influx DB on Windows: Influx DB was installed on a Windows machine to facilitate data storage and management.
2. Installation of Grafana on Windows: Grafana was installed on the same Windows machine to provide visualization and analysis capabilities.
3. Running Grafana: The Grafana interface was accessed by opening a web browser and navigating to the designated port (http://localhost:8086/).

4. Creation of a Grafana Dashboard: A dashboard was created in Grafana to present the performance metrics obtained from the single clustering experiment.
5. Addition of Influx DB as a data source: Influx DB was configured as a data source in Grafana, establishing a connection to access the performance data. The database was set to “myk6db”.
6. Configuration of Influx DB details: Relevant details from Influx DB, such as the query “SELECT * FROM “_internal” LIMIT 10”, were set in Grafana to retrieve the required data.
7. Running Influx DB: The Influx DB service was initiated from the specified installation directory (C:\Program Files\Influx Data\influx dB).
8. Creation of the “single-clustering.js” file: A JavaScript file named “single-clustering.js” was created to implement the single clustering experiment.
9. Adding code to “single-clustering.js”: The necessary code for the single clustering experiment was added to the “single-clustering.js” file, as shown in Figure 2.

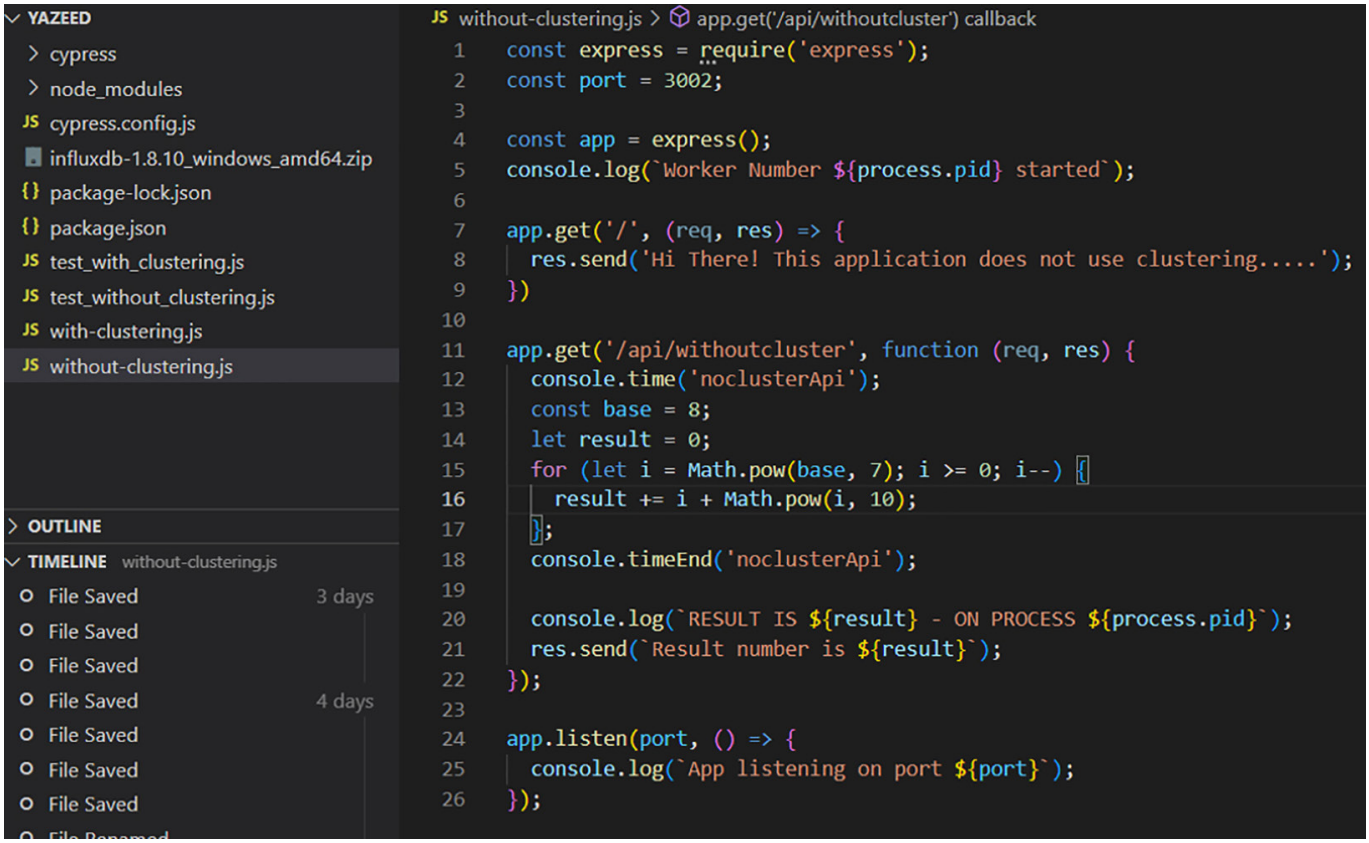

Fig. 2. Single clustering code

10. Running the Node server: The Node server was executed in the terminal, where the single clustering experiment was being performed.
11. Sending 50 requests to the single clustering setup: A total of fifty requests were sent to the single clustering configuration to evaluate its performance and measure relevant metrics. The results are shown in Figure 3.

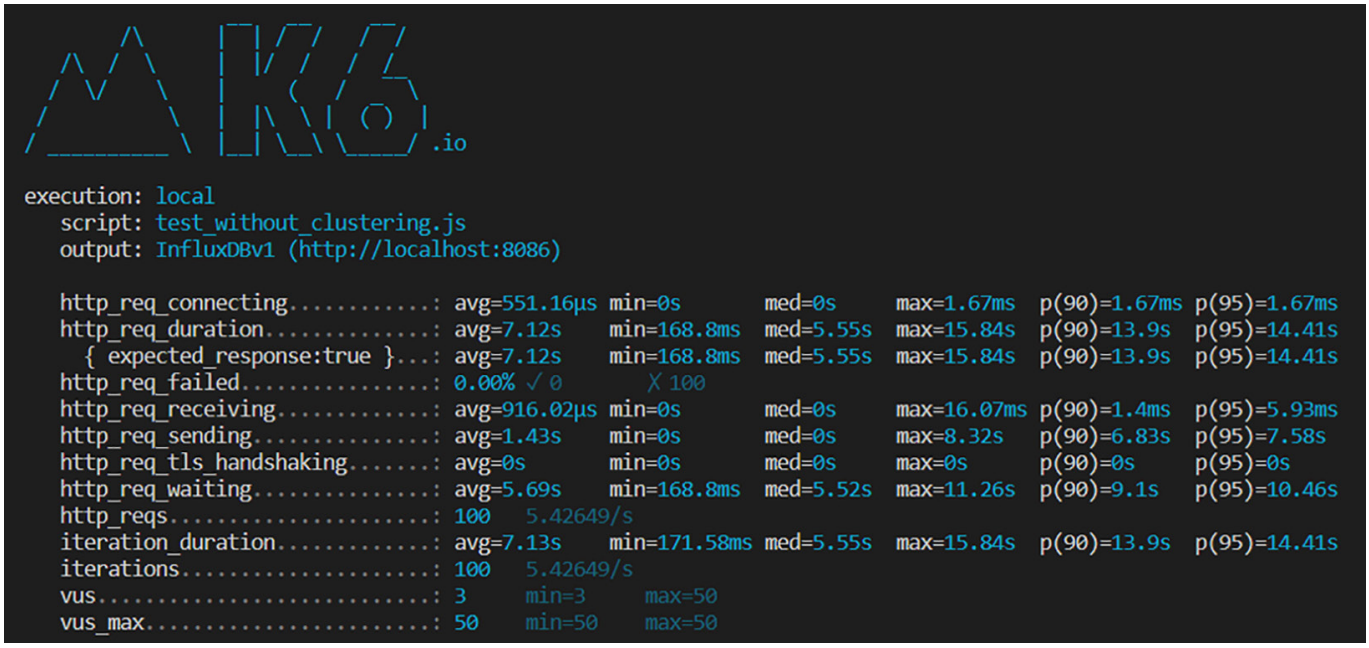

Fig. 3. The result of sending fifty requests with single clustering

12. Dashboard visualization: the dashboard provides a comprehensive overview of the single clustering process, displaying detailed information and metrics. Figure 4 shows the dashboard for the single clustering process.

Fig. 4. Dashboard for the local single clustering process

13. Analysis of the results: the outcome of sending fifty requests to the single clustering setup was recorded and analyzed to assess its performance characteristics. Table 2 below presents the outcome of sending fifty requests using the single cluster configuration:

Table 2. The results of sending fifty requests with single clusters

	Avg	Min	Mid	Max	p (90)	p (95)
HTTP req connecting	551.16 µs	0 s	0 s	1.67 ms	1.67 ms	1.67 ms
HTTP req duration	7.12 s	168.8 ms	5.55 s	15.84 s	13.9 s	14.41 s
Expected response: true	7.12 s	168.8 ms	5.55 s	15.84 s	13.9 s	14.41 s
HTTP req failed	0.00% ✓ 0	✗ 100				
HTTP req receiving	916.02 µs	0 s	0 s	16.07 ms	1.4 ms	5.93 ms
HTTP req sending	1.43 s	0 s	0 s	8.32 s	6.83 s	7.58 s
HTTP req handshaking	0 s	0 s	0 s	0 s	0 s	0 s
HTTP req waiting	5.69 s	168.8 ms	5.52 s	11.26 s	9.1 s	10.46 s
HTTP req	100	5.42649/s				
Iteration duration	7.13 s	171.58 ms	5.55 s	15.84 s	13.9 s	14.41 s
Iterations	100	5.42649/s				
Vus	3	3		50		
Vue's max	50	50		50		

5.4 Experiment 2: Local multi clustering

To evaluate the performance of multi clustering on-premise, the following steps were followed:

1. Influx DB Installation on Windows: Influx DB was installed on a Windows machine to enable efficient data storage and management.
2. Grafana Installation on Windows: Grafana was installed on the same Windows machine to provide advanced visualization and analysis features.
3. Accessing Grafana: The Grafana interface was accessed by launching a web browser and navigating to the designated port (http://localhost:8086/).
4. Dashboard Creation in Grafana: A new dashboard was created within Grafana to display relevant metrics and statistics.
5. Adding Influx DB as a Data Source: A new data source named "Influx DB" was added in Grafana to establish a connection with the Influx DB database.
6. Setting Database Configuration: The database configuration was set to "myk6db" within Grafana to ensure proper data retrieval.
7. Influx DB Details Retrieval: Details from Influx DB were obtained using the query: "SELECT * FROM "_internal". Database" LIMIT 10.
8. Running Influx DB: Influx DB was executed from the specified directory (C:\Program Files\Influx Data\influx dB).
9. Creation of "Multi clustering.js" File: A file named "Multi clustering.js" was created to contain the code for the multi clustering process.
10. Implementation of Multi Clustering Code: The necessary code for multi clustering was added to the "Multi clustering.js" file, as shown in Figure 5.

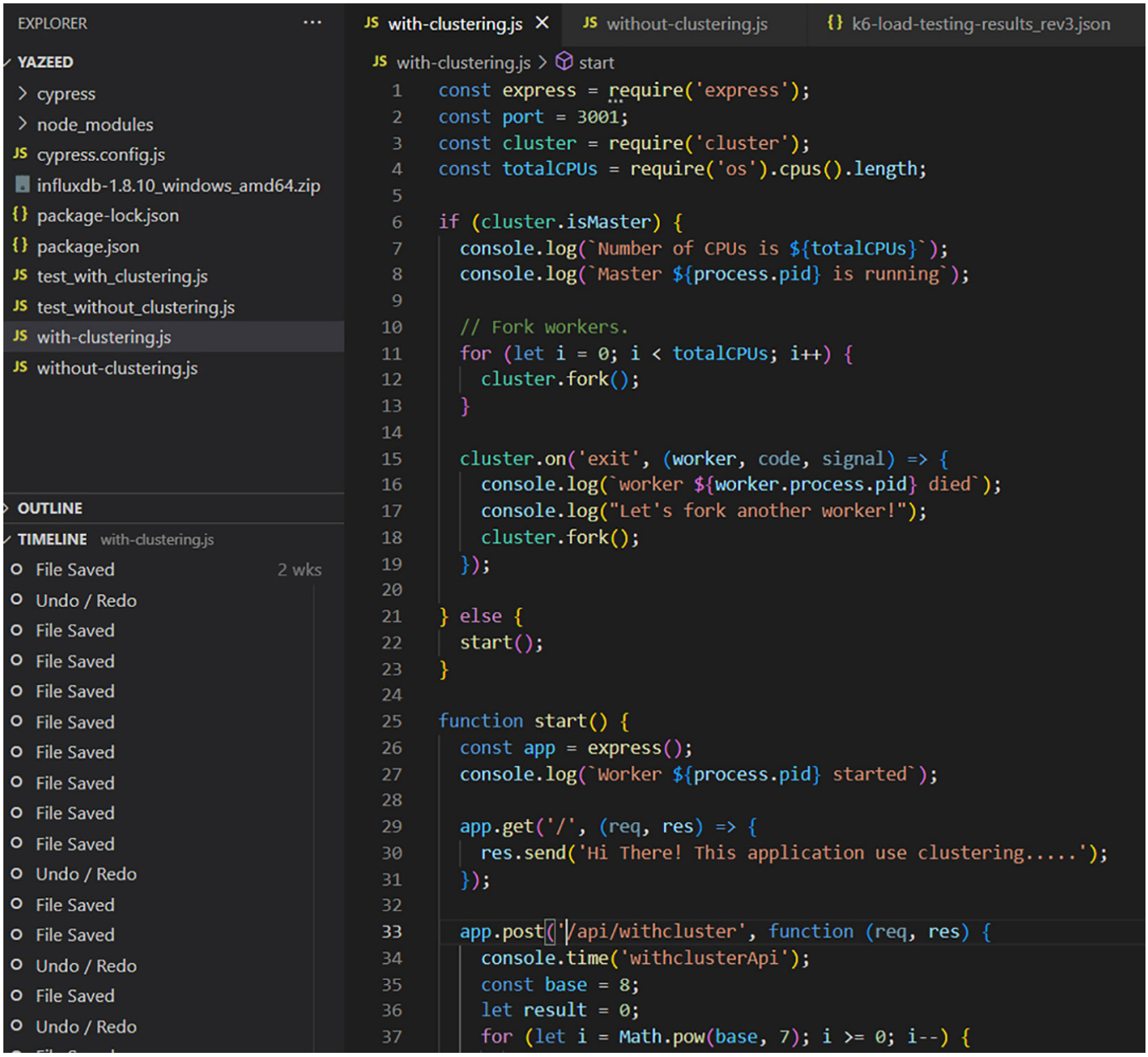

Fig. 5. Multi clustering code

14. Accessing the Node Server Terminal: The terminal where the Node server was running was accessed for further actions.
15. Sending 50 Requests to Multi Clustering: Fifty requests were sent to the multi clustering system to assess its performance. The result of sending fifty requests with multi clustering are shown in Figure 6.

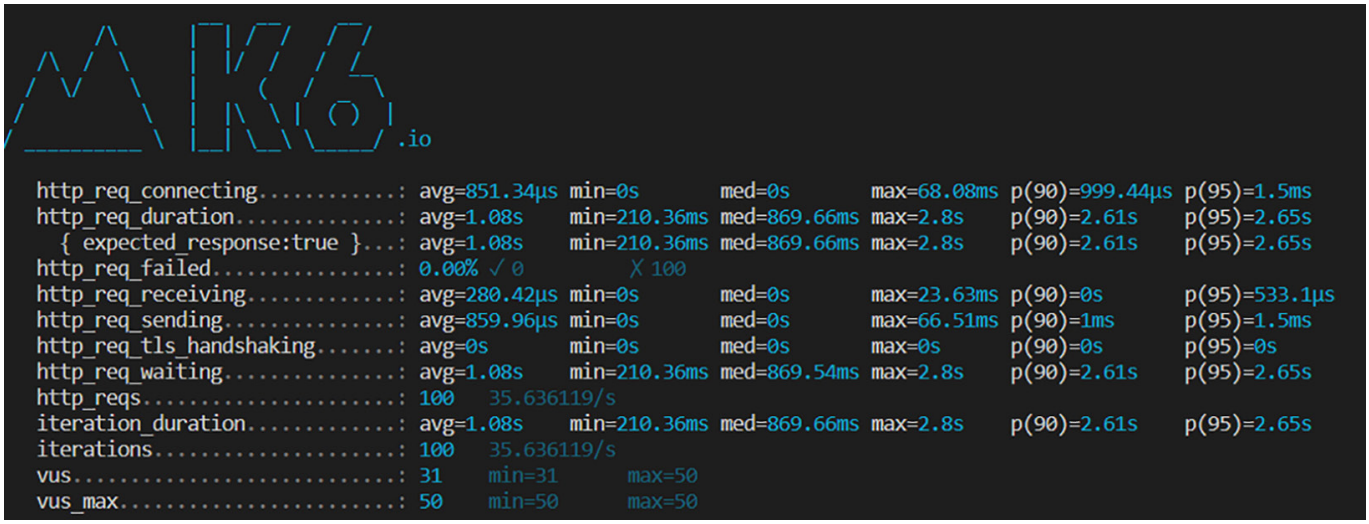

Fig. 6. The result of sending fifty requests with local multi clustering

16. Dashboard Visualization: The dashboard within Grafana displayed detailed information and visualizations related to the multi clustering process. Figure 7 shows the dashboard for the multi clustering process.

Fig. 7. Dashboard for local multi clustering process

17. Analysis of the results: the results obtained from sending fifty requests with multi clustering were analyzed to evaluate system efficiency. Table 3 illustrates the results obtained from sending fifty requests with multiple clusters.

Table 3. The results of sending fifty requests with local multiple clusters

	Avg	Min	Mid	Max	p (90)	p (95)
HTTP req blocked	.39 ms	0 s	0 s	68.08 ms	3.5 ms	.5 ms
HTTP req connecting	551.16 µs	0 s	0 s	68.08 ms	3.5 ms	1.5 ms
HTTP req duration	1.08 s	10.36 ms	869.66 ms	.8 s	.61 s	.65 s
Expected response: true	1.08 s	10.36 ms	869.66 ms	.8 s	.61 s	.65 s
HTTP req failed	0.00%					
HTTP req receiving	80	0 s	0 s	3.63 ms	0 s	533.1 µs
HTTP req sending	859.96 µs	0 s	0 s	66.51 ms	1 ms	1.5 ms
HTTP req handshaking	0 s	0 s	0 s	0 s	0 s	0 s
HTTP req waiting	1.08 s	10.36 ms	869.54 ms	.8 s	.61 s	.65 s
HTTP req	100					
Iteration duration	1.08 s	10.36 ms	869.66 ms	.8 s	.61 s	.65 s
Iterations	100					
Vus	31	31		50		
Vue's max	50	50		50		

As can be noticed from Table 3, the utilization of a multi-cluster system offers several notable advantages over the single cluster as outlined below:

1. Improved scalability: the ability to operate each cluster independently enables the addition or removal of clusters as required to accommodate specific workloads. This facilitates dynamic scaling of the system without necessitating a complete overhaul or reconfiguration.
2. Enhanced reliability: by distributing workloads and data across multiple clusters, a higher level of fault tolerance can be achieved. Even if one or more clusters experience failures, the system can continue to function, minimizing disruptions.
3. Enhanced performance: the parallelization and distribution of tasks across different clusters enable faster completion of tasks, resulting in improved overall performance.

In summary, multi-cluster systems offer significant benefits in terms of scalability, reliability, and performance. These systems find application in various domains, including high-performance computing, data analytics, and cloud computing, where their advantages are increasingly recognized and leveraged.

In addition to single and multi-clustering, we conducted a third experiment with multi-clustering on the cloud. The method of the third experiment is as follows.

5.5 Experiment 3: Multi clustering on cloud

To evaluate the performance of multi clustering system on the cloud, the following steps were followed:

1. Signing up for an Azure account on the Microsoft Azure website.
2. Creating a new Virtual Machine (VM) within a newly created resource group, providing the necessary details for the VM following the step-by-step instructions.

3. Starting the VM and downloading the Remote Desktop Protocol (RDP) or Secure Shell (SSH) version to establish remote control.
4. Executing the Influx DB on the VM.
5. Sending 50 requests to the multi clustering system to assess its performance and analyze the obtained results. Table 4 illustrates the results obtained from sending fifty requests with multiple clusters on cloud.

Table 4. The results of sending fifty requests with multiple clusters on cloud

	Avg	Min	Mid	Max	p (90)	p (95)
HTTP req connecting	370.4 µs	0 s	0 s	2.98 ms	0.98 ms	1.24 ms
HTTP req duration	420.1 µs	0 s	28.07 ms	50.08 ms	25.57 ms	35.98 ms
Expected response: true	65.1 ms	0.17 ms	25.56 ms	58.12 ms	25.18 ms	38.19 ms
HTTP req failed	0.00% ✓ 0	✗ 100				
HTTP req receiving	389.56 µs	0 s	0 s	2.07 ms	0.8 ms	1.91 ms
HTTP req sending	389.56 µs	0 s	0 s	1.58 ms	0.72 ms	1.19 ms
HTTP req handshaking	0 s	0 s	0 s	0 s	0 s	0 s
HTTP req waiting	65.1 ms	0 s	25.56 ms	58.12 ms	25.18 ms	38.19 ms
HTTP req	100					
Iteration duration	65.1 ms	0.17 ms	25.56 ms	58.12 ms	25.18 ms	38.19 ms
Iterations	100					
Vus	1	1		50		
Vue's max	50	50		50		

The performance of the three experiments; using one local cluster, multiple local clusters, and multiple cloud clusters, specifically in managing and processing a set of 50 requests was assessed. The comparative analysis is conducted using various essential metrics, including the number of successful requests, failed requests, completed requests, warned requests, as well as the time taken for request sending, request completion, and request reading per second. Table 5 shows the results of the comparison.

Table 5. Comparative analysis of performance and characteristics of three clustering approaches in handling 50 requests

	Local Single Cluster	Local Multiple Clusters	Multiple Cloud Clusters
Number of Successful Requests	50	50	50
Number of Failed Requests	0	0	0
Number of Completed Requests	50	50	50
Number of Warned Requests	0	0	0
Time Taken to Send Request (per second)	1.43 s	859.96 µs	389.56 µs
Time Taken to Complete Request (per second)	7.12 s	1.08 s	65.1 ms
Time Taken to Read Request (per second)	5.69 s	1.08 s	65.1 ms
Scalability	Limited	Higher scalability	Highly scalable
Fault Tolerance	Single point of failure	Higher fault tolerance	Higher fault tolerance

(Continued)

Table 5. Comparative analysis of performance and characteristics of three clustering approaches in handling 50 requests *(Continued)*

	Local Single Cluster	Local Multiple Clusters	Multiple Cloud Clusters
Resource Allocation	Shared resources	Dedicated resources	Flexible allocation
Performance	Limited capacity	Improved capacity	High capacity
Network Latency	Dependent on local network	Dependent on network	Dependent on network
Load Balancing	Manual	Load balancing	Automated load balancing
Availability	Dependent on local setup	Improved availability	High availability
Scalability Management	Requires hardware expansion	Cluster expansion	Automated scalability
Maintenance and Management	In-house responsibility	In-house responsibility	Managed by provider
Cost	Capital and operational	Capital and operational	Pay-as-you-go

The results from the three experiments comparing the performance of the local single cluster, local multiple clusters, and multiple cloud clusters provide valuable insights into their effectiveness in handling 50 requests. Firstly, all three approaches demonstrated 100% success in processing the requests, with no failures or warnings. However, notable differences were observed in the time taken to send, complete, and read requests. The local single cluster had the longest processing times, whereas the local multiple clusters and multiple cloud clusters exhibited significantly faster performance. This highlights the advantage of distributed setups and cloud infrastructure in improving processing speed.

Scalability and fault tolerance also varied among the approaches. The local single cluster showed limited scalability and a single point of failure, whereas both the local multiple clusters and multiple cloud clusters demonstrated higher scalability and fault tolerance. The ability to allocate dedicated resources and implement flexible resource allocation further contributed to the improved performance of the local multiple clusters and multiple cloud clusters.

Additionally, the availability and maintenance aspects differed between the approaches. The local single cluster was dependent on the local setup, while both the local multiple clusters and multiple cloud clusters offered improved availability and required in-house responsibility for maintenance. However, the multiple cloud clusters had the added advantage of being managed by the cloud provider, reducing the maintenance burden on the organization.

Cost-wise, the local single cluster and local multiple clusters incurred capital and operational expenses, whereas the multiple cloud clusters followed a pay-as-you-go model. This demonstrates the potential cost-effectiveness of cloud-based solutions, allowing organizations to optimize costs based on usage.

Overall, the results indicate that both the local multiple clusters and multiple cloud clusters outperformed the local single cluster in terms of performance, scalability, fault tolerance, resource allocation, and availability. The multiple cloud clusters, in particular, exhibited superior scalability, fault tolerance, and cost-effectiveness. However, organizations should consider their specific requirements and constraints when selecting the most suitable approach for handling requests.

The outcomes of this research offer valuable insights into the efficiency and effectiveness of each clustering approach, thus assisting decision-making processes regarding the selection and implementation of appropriate clustering strategies for diverse applications.

6 CONCLUSION

The research aimed at analyzing how different clustering techniques affect reliability along with efficiency with a keen interest in Node.js applications. Within this context, we analyzed three scenarios—local single clustering, local multi clustering, and multi clustering on cloud for the purpose of undertaking several experiments and evaluations that helped us derive relevant conclusions. We carried out tests within the "single cluster" environment after having established all necessary components such as dashboards databases, sending requests directed towards one cluster. These procedures provided in-depth insights into performance measures and potentialities achievable through singular approaches.

In addition, we further explored the feasibility of "multi clustering" comprising installations along with appropriate configurations while handling program workloads that can be distributed across multiple setups. Our findings showed numerous scalability benefits achieved via such methodology suitable primarily for programming-based network architecture.

Overall, our analytical results reveal significant insights surrounding the use of different clustering methodologies while implementing Node.js applications. The single cluster approach can provide straightforward baselines suitable for routine tasks but upgrading towards scalable optimizations inclusive on diverse networking infrastructures (i.e. cloud servers) might be more beneficial concerning developing better future-proof platforms ensuring reliability and flexibility even during peak usage periods.

7 FUTURE WORK

To advance our understanding further in clustered architectures' applications across varied industries, we must adopt an approach that focuses on investigating configuration optimizations alongside load balancing algorithms while striving for improved scalability along with performance enhancements in related system designs.

Continuing along these lines, there is a need for significantly more efficient work distribution methods that can use resources effectively while reducing content inertia across multiple groups. Moreover, given the increasing levels of cloud computing adoption today than ever before, improving the effectiveness of cloud-based environments cannot be overemphasized. Therefore, research must advance the discovery of innovative technologies and new technologies that improve resource allocation while enhancing energy efficiency in multi-cluster systems. Achieving success here will go a long way in enhancing our collective knowledge and understanding of the application of cluster environments in several different areas.

8 REFERENCES

[1] Y. Jadeja and K. Modi, "Cloud computing-concepts, architecture and challenges," in *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, IEEE, March 2012, pp. 877–880. https://doi.org/10.1109/ICCEET.2012.6203873

[2] M. Ramachandran and V. Chang, "Towards performance evaluation of cloud service providers for cloud data security," *International Journal of Information Management*, vol. 36, no. 4, pp. 618–625, 2016. https://doi.org/10.1016/j.ijinfomgt.2016.03.005

[3] S. Bharany, S. Sharma, O. I. Khalaf, G. M. Abdulsahib, A. S. Al Humaimeedy, T. H. Aldhyani, and H. Alkahtani, "A systematic survey on energy-efficient techniques in sustainable cloud computing," *Sustainability*, vol. 14, no. 10, p. 6256, 2022. https://doi.org/10.3390/su14106256

[4] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *2009 Fifth International Joint Conference on INC, IMS and IDC*, IEEE, August 2009, pp. 44-51. https://doi.org/10.1109/NCM.2009.218

[5] D. Contractor and D. R. Patel, "Accountability in cloud computing by means of chain of trust," *Int. J. Netw. Secur*., vol. 19, no. 2, pp. 251–259, 2017.

[6] W. Martorelli, L. Herbert, and T. M. B. Benefit, *Understanding The Cloud Services Provider Landscape*. Forrester, 2015.

[7] A. Sunyaev and A. Sunyaev, "Cloud computing," in *Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies,* 2020, pp. 195–236. https://doi.org/10.1007/978-3-030-34957-8_7

[8] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen, "G-Hadoop: MapReduce across distributed data centers for data-intensive computing," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 739–750, 2013. https://doi.org/10.1016/j.future.2012.09.001

[9] M. Ayyub, A. Oracevic, R. Hussain, A. A. Khan, and Z. Zhang, "A comprehensive survey on clustering in vehicular networks: Current solutions and future challenges," *Ad Hoc Networks*, vol. 124, p. 102729, 2022. https://doi.org/10.1016/j.adhoc.2021.102729

[10] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (SDN): A survey," *Security and Communication Networks*, vol. 9, no. 18, pp. 5803–5833, 2016. https://doi.org/10.1002/sec.1737

[11] B. Wilder, *Cloud Architecture Patterns: Using Microsoft Azure.* O'Reilly Media, Inc., 2012.

[12] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Algorithms and Architectures for Parallel Processing: 10th International Conference,* ICA3PP 2010, Busan, Korea, May 21–23, 2010. Proceedings. Part I 10 (pp. 13–31). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-13119-6_2

[13] S. Manvi and G. Shyam, *Cloud Computing: Concepts and Technologies*. CRC Press, 2021. https://doi.org/10.1201/9781003093671

[14] M. S. Jawed and M. Sajid, "A comprehensive survey on cloud computing: Architecture, tools, technologies, and open issues," *International Journal of Cloud Applications and Computing (IJCAC)*, vol. 12, no. 1, pp. 1–33, 2022. https://doi.org/10.4018/IJCAC.308277

[15] N. A. Angel, D. Ravindran, P. D. R. Vincent, K. Srinivasan, and Y. C. Hu, "Recent advances in evolving computing paradigms: Cloud, edge, and fog technologies," *Sensors*, vol. 22, no. 1, p. 196, 2021. https://doi.org/10.3390/s22010196

[16] Y. O. Sharrab, I. Alsmadi, and N. J. Sarhan, "Towards the availability of video communication in artificial intelligence-based computer vision systems utilizing a multi-objective function," *Cluster Computing*, vol. 25, no. 1, pp. 231–247, 2022. https://doi.org/10.1007/s10586-021-03391-4

[17] K. A. Kumari, G. S. Sadasivam, D. Dharani, and M. Niranjanamurthy, *Edge Computing: Fundamentals, Advances and Applications*, 2021. https://doi.org/10.1201/9781003230946

[18] X. Wei, L. Ma, H. Zhang, and Y. Liu, "Multi-core-, multi-thread-based optimization algorithm for large-scale traveling salesman problem," *Alexandria Engineering Journal*, vol. 60, no. 1, pp. 189–197, 2021. https://doi.org/10.1016/j.aej.2020.06.055

[19] A. Merzky, M. Turilli, M. Titov, A. Al-Saadi, and S. Jha, "Design and performance characterization of radical-pilot on leadership-class platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 818–829, 2021. https://doi.org/10.1109/TPDS.2021.3105994

[20] M. Hildebrand, J. Khan, S. Trika, J. Lowe-Power, and V. Akella, "Autotm: Automatic tensor movement in heterogeneous memory systems using integer linear programming," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2020, pp. 875–890. https://doi.org/10.1145/3373376.3378465

[21] C. Barakat, S. Fritsch, M. Riedel, and S. Brynjólfsson, "An HPC-driven data science platform to speed-up time series data analysis of patients with the acute respiratory distress syndrome," in *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO),* IEEE, September 2021, pp. 311–316. https://doi.org/10.23919/MIPRO52101.2021.9596840

[22] S. B. Goyal, P. Bedi, A. S. Rajawat, R. N. Shaw, and A. Ghosh, "Multi-objective fuzzy-swarm optimizer for data partitioning," in *Advanced Computing and Intelligent Technologies: Proceedings of ICACIT 2021*, Singapore: Springer, 2022, pp. 307–318. https://doi.org/10.1007/978-981-16-2164-2_25

[23] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19586–19597, 2020.

[24] N. Mungoli, "Scalable, distributed AI frameworks: Leveraging cloud computing for enhanced deep learning performance and efficiency," arXiv preprint arXiv:2304.13738, 2023.

[25] M. Tarawneh, F. AlZyoud, Y. Sharrab, and H. Kanaker, "Secure e-health framework in cloud-based environment," in *2022 International Arab Conference on Information Technology (ACIT)*, IEEE, November 2022, pp. 1–5. https://doi.org/10.1109/ACIT57182.2022.9994164

[26] E. Lee, H. Oh, and D. Park, "Big data processing on single board computer clusters: Exploring challenges and possibilities," *IEEE Access*, vol. 9, pp. 142551–142565, 2021. https://doi.org/10.1109/ACCESS.2021.3120660

[27] M. A. Naeem, Y. B. Zikria, R. Ali, U. Tariq, Y. Meng, and A. K. Bashir, "Cache in fog computing design, concepts, contributions, and security issues in machine learning prospective," *Digital Communications and Networks*, 2022. https://doi.org/10.1016/j.dcan.2022.08.004

[28] A. Poshtkohi and M. B. Ghaznavi-Ghoushchi, *Implementing Parallel and Distributed Systems*. CRC Press, 2023. https://doi.org/10.1201/9781003379041

[29] A. Krishnakumar, S. E. Arda, A. A. Goksoy, S. K. Mandal, U. Y. Ogras, A. L. Sartor, and R. Marculescu, "Runtime task scheduling using imitation learning for heterogeneous many-core systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4064–4077, 2020. https://doi.org/10.1109/TCAD.2020.3012861

[30] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann, 2013.

[31] M. Alsmirat, Y. Sharrab, M. Tarawneh, S. A. Al-shboul, and N. Sarhan, "Video coding deep learning-based modeling for long life video streaming over next network generation," *Cluster Computing*, pp. 1–9, 2023. https://doi.org/10.1007/s10586-022-03948-x

[32] E. R. Rodrigues, F. L. Madruga, P. O. Navaux, and J. Panetta, "Multi-core aware process mapping and its impact on communication overhead of parallel applications," in *2009 IEEE Symposium on Computers and Communications*, IEEE, July 2009, pp. 811–817. https://doi.org/10.1109/ISCC.2009.5202271

[33] B. A. Nayfeh and K. Olukotun, "A single-chip multiprocessor," *Computer*, vol. 30, no. 9, pp. 79–85, 1997. https://doi.org/10.1109/2.612253

[34] D. Wentzlaff and A. Agarwal, "Factored operating systems (fos) the case for a scalable operating system for multicores," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 2, pp. 76–85, 2009. https://doi.org/10.1145/1531793.1531805

[35] H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, and B. Chapman, "High performance computing using MPI and OpenMP on multi-core parallel systems," *Parallel Computing*, vol. 37, no. 9, pp. 562–575, 2011. https://doi.org/10.1016/j.parco.2011.02.002

[36] T. T. Vu and B. Derbel, "Parallel Branch-and-Bound in multi-core multi-CPU multi-GPU heterogeneous environments," *Future Generation Computer Systems*, vol. 56, pp. 95–109, 2016. https://doi.org/10.1016/j.future.2015.10.009

[37] Y. Ma, H. Wu, L. Wang, B. Huang, R. Ranjan, A. Zomaya, and W. Jie, "Remote sensing big data computing: Challenges and opportunities," *Future Generation Computer Systems*, vol. 51, pp. 47–60, 2015. https://doi.org/10.1016/j.future.2014.10.029

[38] Y. Sharrab, N. T. Almutiri, M. Tarawneh, F. Alzyoud, A. R. F. Al-Ghuwairi, and D. Al-Fraihat, "Toward smart and immersive classroom based on AI, VR, and 6G," *International Journal of Emerging Technologies in Learning*, vol. 18, no. 2, pp. 4–16, 2023. https://doi.org/10.3991/ijet.v18i02.35997

[39] Y. O. Sharrab and N. J. Sarhan, "Aggregate power consumption modeling of live video streaming systems," in *Proceedings of the 4th ACM Multimedia Systems Conference*, February 2013, pp. 60–71. https://doi.org/10.1145/2483977.2483983

[40] J. Zhu, P. Patros, K. B. Kent, and M. Dawson, "Node.js scalability investigation in the cloud," in *CASCON 2018,* ACM, 2018, pp. 201–212.

[41] A. R. Al-Ghuwairi, Y. Sharrab, D. Al-Fraihat, M. AlElaimat, A. Alsarhan, and A. Algarni, "Intrusion detection in cloud computing based on time series anomalies utilizing machine learning," *Journal of Cloud Computing*, vol. 12, no. 1, p. 127, 2023. https://doi.org/10.1186/s13677-023-00491-x

[42] G. McGrath, J. Short, S. Ennis, B. Judson, and P. Brenner, "Cloud event programming paradigms: Applications and analysis," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD),* IEEE, June 2016, pp. 400–406, https://doi.org/10.1109/CLOUD.2016.0060

[43] M. Kovatsch, M. Lanter, and Z. Shelby, "Californium: Scalable cloud services for the Internet of Things with coap," in *2014 International Conference on the Internet of Things (IOT),* IEEE, October 2014, pp. 1–6. https://doi.org/10.1109/IOT.2014.7030106

[44] I. K. Chaniotis, K. I. D. Kyriakou, and N. D. Tselikas, "Is Node.js a viable option for building modern web applications? A performance evaluation study," *Computing*, vol. 97, pp. 1023–1044, 2015. https://doi.org/10.1007/s00607-014-0394-9

[45] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2017. https://doi.org/10.1109/ACCESS.2017.2778504

[46] M. M. Golchi, S. Saraeian, and M. Heydari, "A hybrid of firefly and improved particle swarm optimization algorithms for load balancing in cloud environments: Performance evaluation," *Computer Networks*, vol. 162, p. 106860, 2019. https://doi.org/10.1016/j.comnet.2019.106860

[47] G. M. Diouf, H. Elbiaze, and W. Jaafar, "On Byzantine fault tolerance in multi-master Kubernetes clusters," *Future Generation Computer Systems*, vol. 109, pp. 407-419, 2020. https://doi.org/10.1016/j.future.2020.03.060

[48] N. Hashemipour, P. C. del Granado, and J. Aghaei, "Dynamic allocation of peer-to-peer clusters in virtual local electricity markets: A marketplace for EV flexibility," *Energy*, vol. 236, p. 121428, 2021. https://doi.org/10.1016/j.energy.2021.121428

[49] K. Aravindhan and C. S. G. Dhas, "Destination-aware context-based routing protocol with hybrid soft computing cluster algorithm for VANET," *Soft Computing*, vol. 23, no. 8, pp. 2499–2507, 2019. https://doi.org/10.1007/s00500-018-03685-7

[50] M. Gollapalli, M. A. AlMetrik, B. S. AlNajrani, A. A. AlOmari, S. H. AlDawoud, Y. Z. AlMunsour, and K. M. Aloup, "Task failure prediction using machine learning techniques in the google cluster trace cloud computing environment," *Mathematical Modelling of Engineering Problems*, vol. 9, no. 2, 2022. https://doi.org/10.18280/mmep.090234

[51] B. Kruekaew and W. Kimpan, "Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning," *IEEE Access*, vol. 10, pp. 17803–17818, 2022. https://doi.org/10.1109/ACCESS.2022.3149955

[52] D. Saxena, I. Gupta, A. K. Singh, and C. N. Lee, "A fault tolerant elastic resource management framework toward high availability of cloud services," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 3048–3061, 2022. https://doi.org/10.1109/TNSM.2022.3170379

[53] A. Javadpour, A. Nafei, F. Ja'fari, P. Pinto,W. Zhang, and A. K. Sangaiah, "An intelligent energy-efficient approach for managing IoE tasks in cloud platforms," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 4, pp. 3963–3979, 2023. https://doi.org/10.1007/s12652-022-04464-x

[54] O. Adeleke, S. A. Akinlabi, T. C. Jen, and I. Dunmade, "Prediction of municipal solid waste generation: An investigation of the effect of clustering techniques and parameters on ANFIS model performance," *Environmental Technology*, vol. 43, no. 11, pp. 1634–1647, 2022. https://doi.org/10.1080/09593330.2020.1845819

[55] D. Al-Fraihat, M. Alzaidi, and M. Joy, "Why do consumers adopt smart voice assistants for shopping purposes? A perspective from complexity theory," *Intelligent Systems with Applications*, vol. 18, p. 200230, 2023. https://doi.org/10.1016/j.iswa.2023.200230

[56] K. Ali, M. Alzaidi, D. Al-Fraihat, and A. M. Elamir, "Artificial Intelligence: Benefits, Application, Ethical Issues, and Organizational Responses," in *Intelligent Sustainable Systems: Selected Papers of WorldS4 2022*. Singapore: Springer Nature Singapore, 2023, Volume 1, pp. 685–702. https://doi.org/10.1007/978-981-19-7660-5_62

[57] M. G. Al-Obeidallah, D. G. Al-Fraihat, A. M. Khasawneh, A. M. Saleh, and H. Addous, "Empirical investigation of the impact of the adapter design pattern on software maintainability," in *2021 International Conference on Information Technology (ICIT)*, IEEE, July 2021, pp. 206–211. https://doi.org/10.1109/ICIT52682.2021.9491719

[58] M. El-Shebli, Y. Sharrab, and D. Al-Fraihat, "Prediction and modeling of water quality using deep neural networks," *Environment, Development and Sustainability*, pp. 1–34, 2023. https://doi.org/10.1007/s10668-023-03335-5

[59] M. Al-Okaily, D. Al-Fraihat, M. M. Al-Debei, and A. Al-Okaily, "Factors influencing the decision to utilize eTax systems during the covid-19 pandemic: The moderating role of anxiety of covid-19 infection," *International Journal of Electronic Government Research (IJEGR)*, vol. 18, no. 1, pp. 1–24, 2022. https://doi.org/10.4018/IJEGR.313635

[60] A. Alarabiat, O. Hujran, D. Al-Fraihat, and A. Aljaafreh, "Understanding students' resistance to continue using online learning," *Education and Information Technologies*, pp. 1–26, 2023. https://doi.org/10.1007/s10639-023-12030-x

9 AUTHORS

Dr. Abdel-Rahman Al-Ghuwairi received his Ph.D. in Computer Science from New Mexico State University, USA, in 2013. He is currently an Associate Professor at the Software Engineering Department of Hashemite University, Jordan. His research interests encompass Software Engineering, Cloud Computing, Requirements Engineering, Information Retrieval, Big Data, and Database Systems.

Dr. Dimah Al-Fraihat received her Ph.D. in Computer Science – Software Engineering, from the University of Warwick, UK. Currently, she is an Assistant Professor at the Faculty of Information Technology, Isra University, Jordan. Her research interests include Software Engineering, Cloud Computing, Computer Based Applications, Technology Enhanced Learning, and Deep Learning.

Dr. Yousef Sharrab received his Ph.D. in Artificial Intelligence from Wayne State University, USA, in 2017. He currently holds the position of Assistant Professor at the Department of Computer Science, Isra University. His primary research interests encompass Deep Learning, Computer Vision, Speech Recognition, and IoT.

Yazeed Kreishan received his M.Sc. in Software Engineering from the Hashemite University, Jordan. His research interests encompass Software Engineering, Cloud Computing, and Requirements Engineering.

Dr. Ayoub Alsarhan earned his Ph.D. degree in electrical and computer engineering from Concordia University, Canada, in 2011. Currently, he is a Professor in the Department of Information Technology at the Hashemite University, Jordan. His research interests encompass Cognitive Networks, Parallel Processing, Cloud Computing, Machine Learning, and Real-time Multimedia Communication over the Internet.

Hasan Idhaim received his M.Sc. in Computer Science from New Mexico Highlands University in 2008, and B.Sc. in Computer Science from Yarmouk University in 1985. He is currently a Lecturer at the Computer Information System Department of Hashemite University, Zarqa, Jordan. His research interests encompass Database Design & Tuning, E-Commerce Applications, and Industrial Data Analysis.

Ayman Qahmash received his Ph.D. degree in Computer Science from the University of Warwick, UK in 2018. Currently, he is an Assistant Professor, the Head of the Computer Engineering Department, and the Vice Dean of the Computer Science College for academic affairs at King Khalid University. His research interests include educational data mining, artificial intelligence, and statistical modeling.