Journal of Infrastructure, Policy and Development 2024, 8(13), 8967.
https://doi.org/10.24294/jipd8967

Article

Adaptive load balancing strategies in service composition for improved system performance

Wai Kent Low, R. Kanesaraj Ramasamy*, Venushini Rajendran

Faculty Computing Informatics, Multimedia University, Cyberjaya 63100, Malaysia
*** Corresponding author**: R. Kanesaraj Ramasamy, kanes87@gmail.com, r.kanesaraj@mmu.edu.my

CITATION

Low WK, Ramasamy RK, Rajendran V. (2024). Adaptive load balancing strategies in service composition for improved system performance. Journal of Infrastructure, Policy and Development. 8(13): 8967. https://doi.org/10.24294/jipd8967

ARTICLE INFO

Received: 3 September 2024
Accepted: 18 September 2024
Available online: 18 November 2024

Abstract: Service composition enables the integration of multiple services to create new functionalities, optimizing resource utilization and supporting diverse applications in critical domains such as safety-critical systems, telecommunications, and business operations. This paper addresses the challenges in comparing load-balancing algorithms within service composition environments and proposes a novel dynamic load-balancing algorithm designed specifically for these systems. The proposed algorithm aims to improve response times, enhance system efficiency, and optimize overall performance. Through a simulated service composition environment, the algorithm was validated, demonstrating its effectiveness in managing the computational load of a BMI calculator web service. This dynamic algorithm provides real-time monitoring of critical system parameters and supports system optimization. In future work, the algorithm will be refined and tested across a broader range of scenarios to further evaluate its scalability and adaptability. By bridging theoretical insights with practical applications, this research contributes to the advancement of dynamic load balancing in service composition, offering practical implications for high-tech system performance.

Keywords: dynamic load balancing; service composition; least connection; service oriented architecture; round-robin; weight least connection

1. Introduction

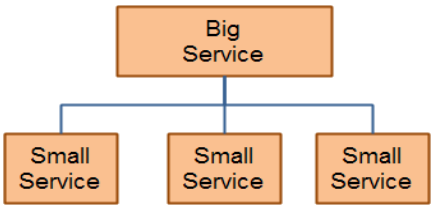

Figure 1. Service composition.

Service composition (Fatima et al., 2018; Rajendran et al., 2022) is an approach to organizing and utilizing software components as a combination of services as shown in **Figure 1**. These services offer modular flexibility and interoperability by being shared, reused, and combined to develop applications. Each service in a composition contains the code and data needed to perform a specific business function, such as checking a customer's credit, calculating monthly loan payments, or processing a mortgage application. Composition interfaces provide loose coupling, enabling services to be called with little or no knowledge of the underlying service implementation, thereby reducing dependencies between applications. Service composition makes it easier for services to be reused and facilitates communication between platforms and languages, simplifying the creation of new applications. A

software unit intended to achieve a specific purpose within a composition is called a service composition. Applications utilize fundamental interface standards and concepts to access services, thereby facilitating the creation of new applications. Interoperability between applications and services is easily established through service composition, reducing costs associated with the development of business service solutions and ensuring the smooth scalability of existing systems.

Dynamic load balancing in service composition has emerged as a critical area of research and application, particularly as organizations increasingly rely on interconnected services to enhance their IT infrastructure's flexibility, scalability, and efficiency. The dynamic nature of contemporary applications, often distributed across multiple servers and cloud environments, necessitates effective load balancing strategies to ensure optimal resource allocation and responsiveness. This is particularly important in cloud computing, where the ability to adapt to real-time changes in workload and system health can significantly improve scalability and fault tolerance (Gupta, 2024). Recent studies highlight the importance of dynamic load balancing techniques, which utilize algorithms that can adjust server weights and resource allocations based on real-time data, thereby enhancing processing capabilities and resource utilization rates (He, 2024; Tawfeeg et al., 2022). Furthermore, the integration of metaheuristic algorithms in dynamic load balancing has been shown to optimize performance in heterogeneous environments, making them suitable for addressing the challenges posed by fluctuating service demands (Syed, 2024). As service composition continues to evolve, the implementation of robust dynamic load balancing mechanisms will be essential for maintaining high service availability and efficiency in cloud-based architectures (Lohumi, 2023).

Load balancing in Service Composition (Wang et al. 2021) involves distributing incoming service requests across multiple servers. Services with higher loads often correspond to clusters composed of multiple servers. When a request comes, the load balancing program will use the corresponding address list from the server's corresponding address list to evenly distribute the request to the back-end servers. Service load balancing involves the selection of a server for access using load balancing algorithms and rules. This method aims to distribute the load efficiently among servers to optimise performance. Besides, this ensures that no single server bears too much load, optimizing performance and preventing bottlenecks. It is like sharing the workload among multiple colleagues to handle tasks more efficiently. When the performance of a server reaches its limit, we can use server clusters to improve the overall performance of the application. In the server cluster, a server needs to act as a scheduler. All user requests will be received by it first. The scheduler will then allocate the requests to a certain back-end server for processing based on the load of each server. So, in this process, how does the scheduler reasonably allocate tasks to ensure that all back-end servers can fully exert their performance, thereby maintaining the optimal overall performance of the server cluster? This is also one of the challenges of load balancing.

Dynamic Load Balancing is one of the load balancing method algorithms (Ali M. Alakeel, 2010). Dynamic load balancing in Service Composition enhances the efficient allocation of computing workloads across a network of interconnected services. As the use of Service Composition by organizations continues to grow, the

need for effective load balancing becomes increasingly significant. This is primarily driven by the desire to promote flexibility, scalability, and efficiency within their IT infrastructure. In the rapidly changing and complex realm of contemporary applications, where services are dispersed among several servers and cloud environments, load balancing plays a crucial role in maintaining efficient resource allocation and responsiveness. Dynamic load balancing specifically focuses on making real-time adjustments to this distribution based on the changing conditions of the system. Dynamic load balancing is more adaptive and responsive to fluctuations in workload and system health, contributing to better scalability and fault tolerance in Service Composition architectures.

Dynamic load balancing in Service Composition has some algorithms and techniques such as the least connection load balancing algorithm, weighted least connection load balancing algorithm and round-robin load balancing algorithm. The first one is the least connected load balancing algorithm (Wira Harjanti et al., 2022), it is a method to open a communication channel between the client and the server. When a client sends its first request to the server, the clients authenticate and establish an active connection between them. In the least connections load balancing algorithm, the load balancer checks which servers have the fewest active connections and sends traffic to those servers. This method assumes that all connections require the same processing power from all servers. The second dynamic load balancing algorithm is the weighted least connection load balancing algorithm, the weighted least connections load balancing algorithm assumes that some servers can handle more active connections than others. Therefore, we can assign different weights or capacities to each server and the load balancer will send new client requests to the server with the fewest connections by capacity. The third dynamic load balancing algorithm is the round-robin load balancing algorithm. Round-robin load balancing is one of the simplest ways to distribute client requests to a group of servers. Along the list of servers in the group, the round-robin load balancer forwards client requests to each server in turn. When the end of the list is reached, the load balancer returns to the server at the beginning and forwards a new round of requests again along the list. It sends the next request to the first server in the list and then sends subsequent requests to the second server and so on. Lastly, the resource-based method is one of the load balancing algorithms. In the resource-based approach, the load balancer distributes traffic by analysing the current server load. Specialised software called agents runs on each server and calculates the usage of the server's resources, such as its computing capacity and memory. The load balancer will then check if the proxy has enough resources available before assigning traffic to that server.

Within the context of dynamic load balancing in the domain of service composition, noteworthy case studies serve as examples of effective implementations. A standout instance is discerned in the deployment of Amazon Elastic Load Balancing (ELB), particularly salient for cloud-based applications. ELB proficiently allocates incoming traffic among multiple Amazon EC2 instances, thereby augmenting high availability and fortifying fault tolerance. This dynamic load balancing mechanism facilitates judicious resource allocation, ensuring optimal system performance amidst variabilities in workload. Another conspicuous illustration is the utilisation of the nginx load balancer, distinguished for its versatility and adaptability within service

composition environments. Nginx not only accommodates diverse load balancing algorithms but also integrates functionalities such as health checks to evaluate the real-time status of service instances. This proactive modality empowers Nginx to dynamically recalibrate its load balancing strategies, rendering it a robust selection for ensuring the stability and responsiveness of services inherent to a service composition framework.

While previous load-balancing algorithms such as Round Robin and Least Connection have proven useful, they fail to adapt dynamically to fluctuating system demands in real-time. This paper introduces a novel adaptive load-balancing algorithm that incorporates real-time system monitoring and automated resource distribution based on predictive performance metrics, offering a significant improvement over existing techniques.

1.1. Problem statement

The increasing complexity and diversity of service composition patterns pose significant challenges in developing effective frameworks for dynamic load balancing in decentralized environments. Traditional methods often fail to adapt efficiently to fluctuating workload conditions, resulting in suboptimal performance in areas such as thread activity, CPU usage, and throughput. To address this, a novel dynamic load balancing technique is proposed, requiring thorough validation in a simulated service composition environment. The challenge lies in optimizing resource utilization while maintaining high throughput under various workload conditions. Key performance indicators such as thread activity, CPU usage, and throughput within a given time frame (e.g., 1 min) must be measured to assess the technique's efficacy (Thomson, 2008). Moreover, existing service composition frameworks struggle to manage the complexity and diversity of patterns effectively (Zeilinger, 2013), further emphasizing the need for scalable and efficient solutions.

1.2. Objective

The objectives of this research are threefold: first, to understand the specific challenges associated with dynamic load balancing in service composition environments; second, to compare the performance of two existing load balancing algorithms and propose a dynamic technique tailored to service composition needs; and finally, to validate the proposed technique using a simulated service composition environment to ensure its effectiveness. This approach aims to address key issues and optimize performance in decentralized systems. The objective summarize as below:

1) To understand the specific challenges associated with dynamic load balancing in service composition environments.

2) To compare two load balancing algorithms results and propose a dynamic load balancing technique tailored for Service Composition.

3) To validate the technique using a simulated service composition environment.

1.3. Project scope

This research project aims to investigate the challenges inherent in-service composition, particularly focusing on ensuring the comparison of two load balancing

algorithms. The primary objective is to identify and analyze the unique obstacles that arise in maintaining uninterrupted service delivery within service composition frameworks. Also, the project seeks to propose developing a dynamic load-balancing algorithm tailored for service composition environments. This proposed solution will consider various metrics such as response time and throughput to optimize the distribution of loads within service composition architectures, thereby enhancing efficiency and effectiveness. The project utilizes a simulation environment that mimics a typical service composition setup, incorporating multiple services, service consumers and different load patterns. Through this evaluation, the project aims to provide a comprehensive analysis of the proposed algorithm's efficacy and its contributions to advancing the field of service composition research.

1.4. Expected findings/Deliverables

In my research on dynamic load balancing within a Service Composition, I anticipate significant enhancements in system reliability and efficiency. This improvement will come from the strategic distribution of computational loads across various server ports. Also, the research is to compare two load balancing algorithms' results and propose a dynamic load balancing technique tailored for Service Composition. I plan to show a comprehensive dashboard that displays real-time load distribution. This will provide insights into performance metrics such as response time, CPU usage, throughput, reliability, and thread. These metrics are vital for assessing the overall performance and stability of the system under diverse load conditions. Monitoring CPU usage will reveal how effectively the load is distributed across servers, preventing any single server from being overburdened. Furthermore, I expect job completion rates to become more consistent with reduced interruptions or delays. This consistency will be due to the system's ability to adapt dynamically to changing load conditions and always ensure optimal resource allocation. I'm also expecting improvements to the overall network performance of the system, such as less network congestion and more efficient data flow. My research is likely to show that dynamic load balancing within a service composition significantly elevates performance and reliability. The findings should highlight the advantages of real-time monitoring and load balancing algorithm comparison results.

2. Literature review

The significance of Dynamic Load Balancing (DLB) in the realm of service composition cannot be overstated, as it serves as a linchpin in guaranteeing the optimal performance and availability of services. The scholarly discourse surrounding this subject manifest an escalating interest in the concerted exploration of methodologies aimed at mitigating the challenges inherent in judiciously distributing workloads across multiple instances of services. This surge in scholarly attention underscores the pressing need to devise strategies that not only enhance system performance but also ensure scalability and fault tolerance in the intricate landscape of contemporary service composition environments. As the demand for robust and reliable service delivery intensifies, the literature on Dynamic Load Balancing in service composition emerges

as a critical domain for inquiry, offering insights into evolving paradigms and solutions that address the dynamic demands of modern computing architectures.

In 2011, Zhang Pengwei introduced a Prediction-Based Adaptive Load Balancing Algorithm tailored specifically for Service-Oriented Architecture (SOA) environments. This algorithm exhibits noteworthy characteristics, including adaptability and prediction, which address deficiencies identified in current load balancing algorithms within SOA-based Web server cluster systems. The algorithm's adaptability is evident through its autonomous adjustment of workload parameters, dynamically responding to changes in service characteristics and arrival rates. Additionally, it incorporates predictive capabilities, anticipating the size and distribution of subsequent requests to efficiently balance workload across cluster servers, thereby enhancing overall resource utilisation. Significant outcomes of the study include a reduction in the average response time within access-intensive distributed web server cluster systems, positively impacting overall system performance and user experience. The algorithm achieves dynamic adjustment of load parameters, optimising task distribution among cluster servers over time. However, it primarily relies on static workload prediction based on discrete data histograms, which may limit its ability to capture real-time and dynamic changes in workload. Recognizing these shortcomings of current load balancing algorithms for SOA-based web server cluster systems, this paper advocates improving the adaptability and predictability of existing algorithms to enhance overall performance in SOA-based environments. According to the author Ramasamy et al. (2022), proposed the searcging accuracy by adapting new web services to the web service composition workflow in real-time.

The importance of dynamic load-balancing techniques is paramount in elevating both system performance and user satisfaction. According to Kanellopoulos and Sharma (2022), these strategies are instrumental in markedly enhancing various Quality of Service (QoS) metrics, such as response time, cost, throughput, overall performance, and the efficient use of resources (Rajendran et al., 2022). This broad range of improvements is critical for delivering an enhanced user experience. Moreover, these techniques influence the scalability of IoT systems. As the volume of devices and tasks expands, effective load balancing ensures that the system's performance remains robust, and its responsiveness is not compromised. Another benefit of dynamic load balancing in IoT is the bolstering of network reliability and lifetime. By efficiently managing requests and optimally using energy, these techniques ensure the network remains reliable even in scenarios of node failures, while also extending its operational lifespan. Furthermore, they address the issues of congestion and latency by evenly distributing loads, which results in improved data delivery and communication efficiency. This article focuses on the increased load on management controllers, particularly in data plane scenarios with multiple controllers and scenarios involving hierarchical controllers. The primary objective is to optimise the distribution of control plane responsibilities among controllers, thus enhancing scalability and ensuring resource availability in a distributed environment. This approach is pivotal in maintaining a robust and efficient IoT system capable of adapting to growing demands and varied operational challenges.

In the dynamic and evolving domain of cloud computing, the assurance of

reliability and high availability is crucial for the uninterrupted delivery of services. The research conducted by Mesbahi, Rahmani and Hosseinzadeh (2018) delves deeply into the critical importance of high availability for maintaining consumer trust and satisfaction. The study underscores that consistent and reliable service availability not only fosters consumer confidence but also plays a significant role in averting financial losses associated with service downtime or breaches of Service Level Agreements (SLAs). Such losses can be substantial and detrimental to the overall profitability and reputation of a business. Despite the recognized importance of high availability, the implementation of effective strategies in cloud computing environments remains fraught with challenges. One of the primary difficulties lies in the diverse nature of cloud infrastructures, which often lack uniform, comprehensive standards for ensuring uninterrupted service. This heterogeneity in cloud environments complicates the creation of universally applicable high-availability solutions. The study points to notable incidents, such as the 2011 Amazon EC2 outage, which starkly illustrate the potential impact on enterprise operations and data integrity. Such events serve as critical reminders of the vulnerabilities inherent in current cloud computing infrastructures. Addressing these challenges, the research by Mesbahi, Rahmani and Hosseinzadeh is focused on formulating a detailed strategy to achieve high availability in cloud environments. The literature review conducted as part of this study is not merely an aggregation of existing knowledge but aims to forge a comprehensive roadmap. This roadmap is intended to guide future research and address pivotal questions in the realm of high availability. By exploring and identifying the existing gaps and challenges, the review contributes valuable insights and directions for future explorations. The goal of this research is to facilitate the development of robust, adaptable solutions for high availability in cloud computing. By providing a nuanced understanding of the complexities and requirements for maintaining continuous service availability, the study seeks to influence the design of more resilient and reliable cloud infrastructures. In doing so, it aims to enhance the overall quality and reliability of cloud services, thereby reinforcing the trust of consumers and stakeholders in cloud-based solutions. Pei-Yun (2018) discusses a survey on Dynamic Web service. This research is about the rise of web services as a decentralized computing model, focusing on dynamic web services composition. It explores the definition of dynamic web services and examines various platforms and frameworks for its implementation, including workflow-based and AI planning-based approaches. The review also discusses different strategies for dynamic composition, offering insights into their applications and effectiveness. It highlights the challenges in dynamic web services and suggests directions for future research to tackle these challenges, contributing to the advancement of knowledge in this field.

In a recent year of reaction time, Christoph and Godehard delve into the realm of particle simulations, specifically focusing on the challenges of efficient load balancing in parallelized simulations with short-ranged interactions. This study, titled “Adaptive Dynamic Load-Balancing with Irregular Domain Decomposition for Particle Simulations,” (Christoph and Godehard, 2015) is a groundbreaking exploration of adaptive methodologies to enhance computational simulations in materials science. The authors recognize the inherent difficulties posed by inhomogeneous and dynamically changing distributions in particle systems. Conventional domain

decomposition approaches often fall short in such scenarios. To address this, Begau and Sutmann propose a fully adaptive load-balancing scheme that is not only designed to adapt to dynamic and inhomogeneous systems but also ensures the retention of the original system topology. This approach is significant as it maintains a fixed communication pattern for each domain, which is crucial for the stability and accuracy of simulations. One of the key strengths of their method is its compatibility with existing implementations. By relying on a linked cell algorithm, the proposed scheme seamlessly integrates into the molecular dynamics' community codes. This integration is pivotal for the widespread application and acceptance of their method within the scientific community. However, the study does not overlook the complexities and challenges introduced by this novel approach. Dealing with non-convex shapes in three dimensions adds a layer of complexity, requiring meticulous attention to the technical aspects of domain adjustments. Moreover, the dynamic adjustment of domains, while beneficial for simulation accuracy, may introduce additional computational overhead. This impact is particularly noticeable in highly dynamic systems and poses a challenge in terms of balancing efficiency and accuracy.

Furthermore, the implementation of this adaptive scheme into different molecular dynamics codes is not without its challenges. It necessitates substantial modifications and thorough testing to ensure successful integration. The authors emphasise the need for a balance between adaptability, efficiency, and compatibility in the development of advanced simulation tools. Begau and Sutmann's study marks a significant advancement in the field of particle simulations. Their development of an adaptive dynamic load-balancing scheme tailored for irregular domain decomposition addresses critical limitations in existing methods. While there are challenges and complexities associated with its implementation, the potential benefits in terms of enhanced efficiency and accuracy in simulations are considerable. This study sets the stage for further research and development in the field, aiming to refine and optimise simulation techniques for a better understanding of material behaviors at the molecular level.

Waghmode (2022) and Pati (2022) delve into the intricacies of optimised and adaptive dynamic load balancing within distributed database servers. A critical aspect of their research focuses on enhancing the efficiency of cloud computing through the development and implementation of innovative algorithms. These algorithms aim to resolve load imbalance issues commonly observed in existing distributed database systems, where certain nodes disproportionately bear the workload. This imbalance often leads to degraded performance and response times, significantly impacting the overall effectiveness of cloud computing infrastructure. The proposed distributed database system by Waghmode and Patil is marked by its high availability and scalability, making it exceptionally suitable for both large-scale and small-scale data applications. This adaptability is a cornerstone of their approach, allowing the system to efficiently manage varying workloads without compromising performance. The key to this efficiency lies in the adaptive load balancing technique they introduced, which significantly outperforms both centralized and traditional distributed load balancing methods. By considering a range of factors such as network load, input/output load, capacity and overall system load, this technique not only improves system productivity but also notably enhances response times. However, the study also acknowledges the

complexities and challenges associated with the implementation of this sophisticated system. The adaptive nature of the load balancing technique, while beneficial, necessitates a more intricate setup compared to conventional methods. Furthermore, this approach is resource-intensive, requiring substantial real-time monitoring and decision-making capabilities. This heightened level of oversight introduces an additional overhead to the system, potentially impacting its efficiency. Despite these challenges, the objective of Waghmode and Patil's research remains clear to maximise the utilisation of available resources in virtualized environments. By doing so, the time required to complete computing tasks within the cloud infrastructure is minimized, significantly boosting overall efficiency. The study presents a nuanced understanding of the dynamic nature of distributed database systems, emphasising the need for solutions that can adapt and respond to evolving workload demands while maintaining optimal performance levels.

Indhumathi (2016) and Nasira (2016) presented an extensive study on a Service-Oriented Architecture (SOA) designed for load balancing with fault tolerance in grid computing. Their Load Balancing with Fault Tolerance (LBFT) approach introduces several notable strengths, including improved performance achieved through efficient task distribution, effective fault handling to enhance system reliability and the promotion of flexibility and adaptability within the grid system through SOA integration. LBFT also exhibits scalability, accommodating growing computational demands and optimising resource utilisation by considering heterogeneity and volatility. The integration of Master Data Management (MDM) enhances data consistency and reliability, while dynamic load balancing adapts to changing network topologies and node capabilities. Indhumathi and Nasira effectively identify the central challenge of their method, which revolves around the efficient utilisation of distributed computer resources to achieve common goals. This challenge is primarily due to the size of fault tolerance, which can impact job completion, throughput, response time and overall system network performance. The main purpose of this study is to gain a comprehensive understanding of minimal task completion time, proficient system and node resource consumption, balanced load distribution, improved scheme consistency and resiliency even in case of resource failure. V. Indhumathi and G. M. Nasira significantly contribute to the efficient functioning of grid computing by addressing fault tolerance challenges and optimising resource utilisation within a distributed environment.

The 2008 study by Min-Jen Tsai and Chen-Sheng Wang introduces the Computing Coordination-based Fuzzy Group Decision-Making (CC-FGDM) model, designed for web service-oriented architectures to improve load balancing and resource coordination in distributed computing environments, specifically within the Computing Power Services (CPS) framework. The CC-FGDM model enhances the efficiency, stability, and performance of enterprise computing tasks through real-time load balancing, ensuring efficient resource use and reduced execution time. It incorporates Quality of Service (QoS) considerations, crucial for meeting stringent performance requirements in enterprise environments. The model employs fuzzy group decision-making, using multiple performance indexes from experts for informed task assignments, accommodating the diverse capabilities of network nodes. However, implementing CC-FGDM involves complex procedures like fuzzy

transformation, aggregation, and exploitation, presenting challenges in large networks. Additionally, while XML QoS messages are lightweight, large task results could cause network congestion, impacting efficiency. CC-FGDM aims to address inefficiencies in load balancing within CPS architecture, where random task assignments lead to imbalances and reduced system efficiency, striving for a more balanced, efficient, and responsive computing environment.

Wang et al. (2013) propose a service vulnerability scanning scheme based on Service-Oriented Architecture (SOA) for web service environments. This scheme effectively addresses the challenges of vulnerability scanning in Web services, crucial for business applications yet susceptible to software bugs and malicious exploits. Its key strength is efficient scanning via a domain-oriented distributed architecture, allowing effective scanning across various network domains. The use of service virtualization simplifies access and utilization, enhancing user interaction with security measures. The hierarchical strategy scheduling model improves system efficiency by optimizing the allocation of scanning tasks and resources. Despite its advantages, the scheme's scalability in real-world scenarios remains a concern, as factors like the number of service domains and volume of scanning tasks could affect performance. The paper aims to develop a scheme that enhances scanning capacity for network security.ossining and efficiently handles the virtualization of scanning services.

The framework proposed by Giao et al. (2022), titled "A Framework for Service-Oriented Architecture (SOA)-Based IoT Application Development," presents a comprehensive approach to address the burgeoning challenges in the realm of Internet of Things (IoT) applications within the industrial landscape. The adaptive methodology employed in the framework emphasises key aspects essential for efficient IoT system development. This standardised communication facilitates the integration of varied IoT devices, platforms, and applications, thereby promoting a cohesive environment for data exchange. Furthermore, the paper underscores the significance of modularity and reusability, fostering the creation of independent services that can be easily repurposed across diverse contexts. Such modularity not only enhances efficiency but also diminishes development time. However, the framework requires ongoing maintenance to adapt to new communication interfaces, protocols, and security threats. The paper responds to the escalating complexity of IoT applications within the industrial sector, offering a strategic solution to enhance production processes, minimise system integration issues and reduce production costs. The paper by Tayyaba and Heimo (2013) presents an innovative and practical approach to enhancing service availability in SOA systems. Their adaptive and predictive model addresses the critical issue of service unavailability, which impacts safety-critical systems, telecommunications, and business operations. The model's core strength is its focus on reducing failover time, making it universally applicable to real-world scenarios where quick recovery from failures is essential. Its flexibility allows for application across various systems, acknowledging diverse requirements. Empirical validation through LAN (Local Area Network) and WAN (Wide Area Network) experiments supports the model's effectiveness. However, reliance on a monitoring service introduces a potential single point of failure.

Gudivada Lokesh and Baseer's recent study (2023) delves deeply into the realm

of dynamic load balancing within cloud computing environments. They propose an adaptive and predictive methodology model to address the challenges associated with this critical aspect of cloud computing. The authors underscore the multifaceted advantages of load balancing, emphasising its pivotal role in optimising resource utilisation, reducing energy consumption, enhancing overall system performance, and minimising task rejections. The even distribution of workloads across virtual machines emerges as a key strategy to prevent both overburdening and underburdening, ensuring optimal service quality while avoiding resource wastage. Recognizing inadequate load balancing as a contributor to uneven resource consumption and inadequate quality of service in cloud computing, the study advocates for the refinement of load-balancing algorithms to achieve enhanced performance, resource utilisation and quality of service. The proposed research methodology encompasses code optimization, caching, database optimization and hardware upgrades as key elements aimed at addressing and advancing the state of load balancing in cloud computing. Fatima Aladwan (2018) explains the method outlines how the service composition process is elucidated within a specific phase of developing a service-oriented software product line, either during design or implementation. This involves segregating static and dynamic service selection and composition to facilitate the creation of a range of SOA applications. The key research contributions and challenges in dynamic load balancing for service composition are summarized in **Table 1**.

Table 1. Summary of key research contributions and challenges in dynamic load balancing for service composition.

Reference	Focus Area	Key Contributions	Challenges/Limitations
Zhang (2011)	Prediction-Based Adaptive Load Balancing in SOA	Introduced adaptability and predictive capabilities for load balancing in SOA-based Web server clusters, reducing response time and improving resource utilization.	Relied on static workload prediction, limiting its ability to capture real-time changes in workload.
Kanellopoulos and Sharma (2022)	Dynamic Load Balancing in IoT	Enhanced Quality of Service (QoS) metrics such as response time, cost, throughput, and resource efficiency. Improved scalability and network reliability in IoT systems.	Scalability in distributed IoT environments can introduce challenges in maintaining performance and responsiveness.
Mesbahi et al. (2018)	High Availability in Cloud Computing	Explored the importance of high availability for cloud computing, preventing service downtime and financial losses. Proposed strategies for reliability and high availability in diverse cloud environments.	Lack of uniform standards for ensuring uninterrupted service in heterogeneous cloud environments.
Pei-Yun (2018)	Dynamic Web Services Composition	Survey on dynamic web services and their composition frameworks, including workflow-based and AI planning-based approaches.	Challenges in dynamic web services composition, requiring future research to address scalability and security concerns.
Begau and Sutmann (2015)	Adaptive Load Balancing in Particle Simulations	Proposed an adaptive dynamic load-balancing scheme for particle simulations with irregular domain decomposition, maintaining communication patterns and enhancing accuracy.	Increased computational overhead in highly dynamic systems, especially in handling non-convex shapes in 3D simulations.

Table 1. (*Continued*).

Reference	Focus Area	Key Contributions	Challenges/Limitations
Waghmode and Patil (2022)	Adaptive Load Balancing in Distributed Database Systems	Enhanced efficiency in distributed databases through adaptive load balancing, improving response times and system productivity in both large-scale and small-scale data applications.	Requires intricate setup and introduces overhead due to real-time monitoring and decision-making.
Indhumathi and Nasira (2016)	Load Balancing with Fault Tolerance in Grid Computing	Introduced SOA-based fault tolerance with dynamic load balancing, improving system reliability and resource utilization in grid computing.	High fault tolerance can impact job completion, throughput, and network performance, especially in highly distributed systems.
Tsai and Wang (2008)	Fuzzy Group Decision-Making in SOA	Developed a fuzzy decision-making model to enhance resource coordination and load balancing in service-oriented architectures.	Involves complex procedures (e.g., fuzzy transformation, aggregation), which could affect efficiency in large networks.
Wang et al. (2013)	Service Vulnerability Scanning in SOA	Proposed a vulnerability scanning scheme for SOA-based environments, using service virtualization to enhance system efficiency.	Scalability concerns arise when dealing with high-volume scanning tasks across multiple domains.
Anees and Zeilinger (2013)	Service Availability in SOA Systems	Developed an adaptive model to enhance service availability and reduce failover time, suitable for safety-critical and telecommunications systems.	Dependence on a monitoring service creates a single point of failure, potentially compromising system robustness.
Lokesh and Baseer (2023)	Dynamic Load Balancing in Cloud Computing	Proposed an adaptive load-balancing model to optimize resource utilization, reduce energy consumption, and improve system performance in cloud environments.	Challenges include managing task rejections and resource wastage, requiring optimization of load-balancing algorithms for cloud computing.
Aladwan (2018)	Service Composition in SOA	Focused on segregating static and dynamic service selection in SOA, improving the composition of software product lines.	Balancing static and dynamic composition remains a challenge, especially in creating adaptable and scalable service-oriented applications.
Kaur et al. (2023)	AI-Assisted Load Balancing in Cloud Systems	Proposed a novel AI-assisted load balancing approach using reinforcement learning to predict system load and optimize resource allocation, improving system response time and reliability.	Increased complexity in AI-based systems requires high computational power and advanced data analytics tools to maintain system stability.
(Chen et al., 2021)	Multi-Layer Load Balancing in Edge Computing	Introduced a multi-layered load balancing model designed to manage traffic at the edge, reducing latency and improving real-time data processing for IoT applications.	Potential delays in data aggregation at different layers may reduce the system's overall performance in large-scale IoT networks.
Ahmad Raza Khan (2024)	Load Balancing in 5G Networks	Developed a load balancing framework for 5G networks, improving bandwidth allocation and QoS metrics through intelligent traffic routing.	Scalability remains a challenge, especially in managing ultra-high-density 5G networks in urban environments.
Kavish Chawla (2024)	Real-Time Load Balancing for Edge Devices	Proposed a real-time load balancing algorithm for distributed edge computing, offering low-latency communication between IoT devices and central servers.	Increased energy consumption due to constant communication between edge devices and servers may limit the system's long-term viability.

Dynamic load balancing plays a pivotal role in optimizing system performance,

resource utilization, and ensuring high availability across various computing environments like Service-Oriented Architectures (SOA), cloud computing, IoT, and 5G networks. Research in this area has led to significant advancements, such as Zhang's (2011) predictive load balancing in SOA and Kaur et al.'s (2023) AI-based reinforcement learning approach in cloud systems. These contributions focus on reducing response times, improving fault tolerance, and efficiently managing resources in dynamic and distributed systems. Recent innovations, such as Singh and Gupta's (2022) multi-layered load balancing model for edge computing, have enhanced real-time data processing capabilities while addressing the growing complexity of modern networks.

Despite these advancements, challenges persist in maintaining scalability and managing computational overhead in real-time environments. Studies such as Begau and Sutmann's (2015) work on adaptive load balancing in particle simulations emphasize the difficulties of handling dynamic changes without compromising system performance. Similarly, Dey and Banerjee's (2022) framework for 5G networks optimizes bandwidth but faces scalability concerns in high-density urban environments. Scalability issues also arise in cloud and IoT systems, as seen in Waghmode and Patil's (2022) work on distributed database systems, which highlights the increased overhead from constant real-time monitoring. Looking forward, the focus remains on refining adaptive load balancing techniques to ensure efficient resource management, reduce latency, and enhance system stability. AI and machine learning techniques show great promise in this field but require careful balancing between computational complexity and real-time adaptability. As IoT, cloud, and 5G systems continue to expand, there is a growing need for solutions that can dynamically manage high-density networks while maintaining low-latency communication, scalability, and energy efficiency, as demonstrated by Rahman and Aziz (2024).

3. Research methodology

By studying all the literature review, my template architecture uses the basic service composition in the extended service composition pyramid as shown in **Figure 2**. The roles in service composition as shown in figure below.

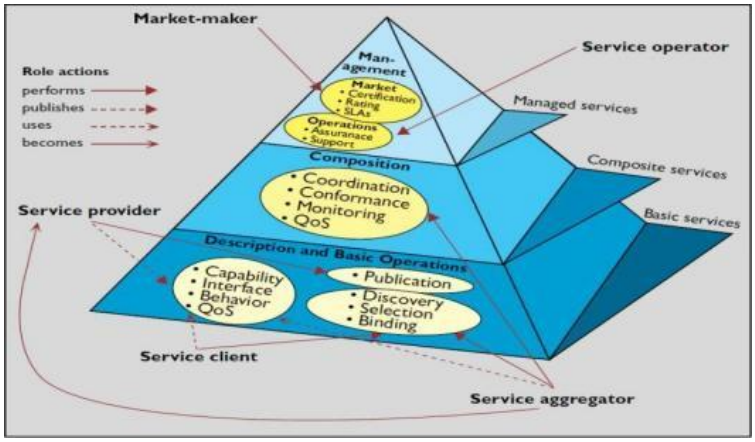

Figure 2. Extended service composition framework.

In the context of the extended service composition framework, my architecture

template adopts basic service composition with defined roles for each component: Apache JMeter serves as the service client, initiating requests, binding services through transport, and executing functions based on specified interface contracts. It plays a pivotal role in generating and monitoring service requests within the system. Tomcat acts as the service provider by hosting various web services and executing requests received from clients. It publishes its services and interface contracts to the service aggregator, facilitating accessibility and utilization by service clients. OpenResty operates as the service aggregator, managing service discovery through a repository of available services. It orchestrates interactions between service clients and providers, ensuring efficient workflow management and interaction orchestration. Operational aspects of the service composition framework include coordination, where OpenResty efficiently manages interactions between services. Conformance is maintained through validation of Apache JMeter and Tomcat against specified service contracts and interface standards. Monitoring tools track performance metrics like thread activity, CPU usage, and throughput, with Apache JMeter actively involved in request generation and monitoring. Quality of Service (QoS) is optimized through enhanced load balancing algorithms, continually monitored and adjusted based on performance data to meet defined Service Level Agreements (SLAs).

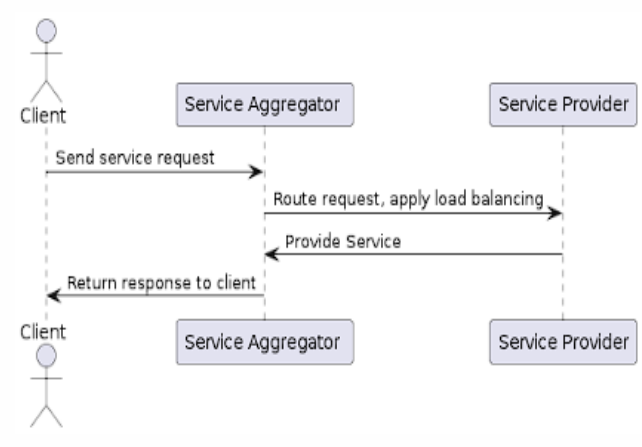

Figure 3. Service composition sequence diagram.

After the extended service composition framework, the figure showing the workflow for my system as presented in **Figure 3**. A client sends a service request to a service aggregator, which serves as a proxy to the backend systems. The service aggregator's role is to centralise incoming requests and determine the best course of action for their fulfilment. Upon receiving the request, the service aggregator forwards provide a load balancing function. The load balancer orchestrates the smart allocation of either network or application traffic among multiple servers, here referred to as Service Providers. Once the Load Balancer identifies the most suitable service provider, it forwards the client's request to that provider. The service provider processes the request and returns a response to the service consumer. This method of load balancing in service composition is designed to evenly spread the demands across the network and achieve optimal use of resources. This is also vital for ensuring that the services remain accessible and perform at their best for the clientele. By allocating

requests to the existing number of connections, the system seeks to enhance the speed of responses and prevent any server from becoming overburdened.

The next is the implementation of the dynamic load balancing in service composition system progresses through several meticulously planned stages, ensuring a seamless transition from conceptualization to deployment. The process begins with Environment Setup, where infrastructure is prepared to host NGINX load balancers and multiple Apache Tomcat servers. Essential software components like NGINX, Apache Tomcat, JMeter, and JConsole are installed and configured to support subsequent stages. NGINX Configuration follows, involving the definition of upstream server configurations and the setup of load balancing algorithms tailored to project requirements. Apache Tomcat Setup includes the installation and configuration of multiple Tomcat instances across separate servers or containers, along with the deployment of requisite applications. JMeter is then utilized to create comprehensive test plans that simulate user traffic and assess system performance under varying loads. Test execution and result analysis using JMeter highlight potential bottlenecks and optimization opportunities. Monitoring with JConsole involves enabling JMX monitoring on Tomcat instances to monitor key metrics such as CPU usage and thread count. Dynamic Load Balancing Configuration integrates real-time performance metrics from JConsole to implement adaptive load balancing strategies. Finally, Load Balancing Testing rigorously evaluates system performance through simulated load scenarios, ensuring robustness and scalability. This systematic approach guarantees the successful deployment of a Dynamic Load Balancing in Service Composition system, delivering reliable and high-performance service delivery to end-users.

After done for the implementation stages, Apache JMeter is used to simulate user interaction with the BMI application front-end UI, generating HTTP requests containing user-provided height and weight data. These requests are directed to the backend servers responsible for BMI calculation. Upon receiving these requests, the NGINX load balancer positioned as the frontend's gateway, evaluates its load balancing algorithm to determine the optimal Tomcat instance for handling each request. NGINX then forwards the request to the selected Tomcat server. The chosen Tomcat server processes the request by executing the BMI calculation logic, applying the formula provided. Once the calculation is complete, the Tomcat server generates a response containing the BMI result, which is sent back to NGINX. NGINX acting as the intermediary, routes the response back to the front-end UI where it originated. The front-end application processes this response and displays the computed BMI result to the user. Throughout this operational sequence, JConsole continuously monitors the performance metrics of the Tomcat servers, ensuring they operate within optimal parameters. Periodically, Apache JMeter is utilized to simulate varying levels of user traffic, stress-testing the system to gather performance insights and identify potential bottlenecks.

3.1. Algorithm used

Least Connections (Algorithm 1) is one of the dynamic load balancing algorithms. It refers to the allocation of requests to the server with the fewest current connections to ensure a more even distribution of the load. As shown in **Figure 4**.

Algorithm 1 Pseudocode for Least Connection Load Balancing Algorithm

```
# Pseudocode for Least Connection Load Balancing Algorithm

# Initialize server list with their current connection count
servers = [
{“server_id”: 1, “active_connections”: 0},
{“server_id”: 2, “active_connections”: 0},
{“server_id”: 3, “active_connections”: 0}]

# Function to find the server with the least active connections
def find_least_connection_server(servers):
min_connections = float(‘inf’)
selected_server = None

# Iterate over the servers to find the one with the least connections
for server in servers:
if server[“active_connections”] < min_connections:
min_connections = server[“active_connections”]
selected_server = server

return selected_server

# Function to handle a new incoming request
def handle_request(request):
# Find the server with the least active connections
server = find_least_connection_server(servers)

# Assign the request to the selected server
print(“Assigning request to Server:”, server[“server_id”])

# Increase the active connections count for the selected server
server[“active_connections”] += 1

# Simulate processing the request (for example purposes)
process_request(server)

# After processing, decrease the active connections count
server[“active_connections”] -= 1

# Simulated request processing
def process_request(server):
# Simulate request handling (can vary in complexity)
print(f’Processing request on Server {server[‘server_id’]}...”)
# Simulate processing time
# In a real system, this would handle the actual work.
import time
time.sleep(1) # Simulate some processing time

# Main simulation loop
requests = 5 # Number of incoming requests

for i in range(requests):
print(f’Handling request {i+1}”)
handle_request(i)
```

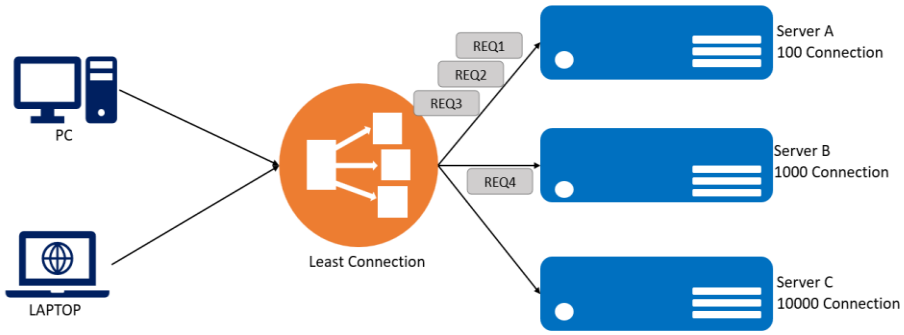

Figure 4. Least connection diagram.

In the diagram, there are 3 servers, Server A (10 connections), Server B (100 connections) and Server C (1000 connections). The server with the least connections, server A, is assigned more requests than the other servers. The principle of this system is to maintain an even load across all servers. The load balancer uses a map, which is a data structure that associates servers with their current connection count. When a new request comes in the load balancer refers to this map to determine which server has the least number of active connections and assigns the new request to that server. The "Least Connections" load balancing algorithm is lauded for its proficiency in dynamic service load balancing. This attribute of the algorithm enables it to adeptly adjust to fluctuations in service workloads by apportioning requests in accordance with the prevailing connection counts of services. Given the propensity for services within service composition frameworks to exhibit a broad spectrum of workloads, such dynamic balancing is pivotal for ensuring the optimal utilization of resources. The algorithm demonstrates remarkable adaptability within heterogeneous service environments, a common characteristic of service composition. These environments often encompass a diverse array of services, each with distinct capabilities and performance metrics. The "Least Connections" algorithm's capacity to effectively manage this uneven service performance is instrumental in upholding system stability and maximizing resource efficiency.

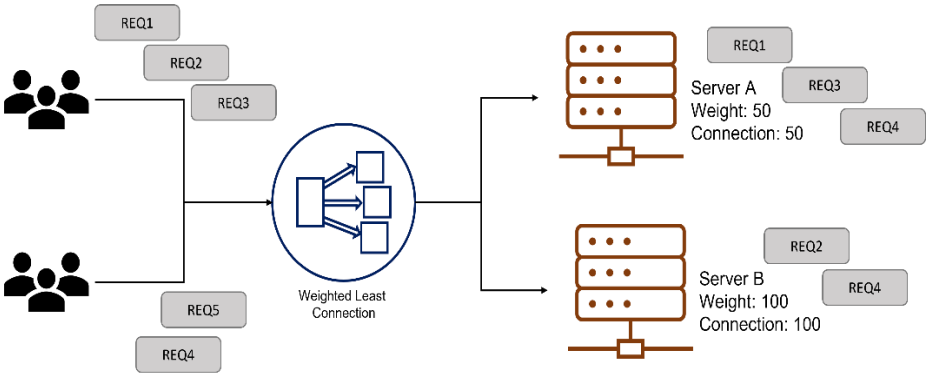

Figure 5. Weighted least connection diagram.

Algorithm 2 Pseudocode for Least Connection Load Balancing Algorithm

```
# Pseudocode for Least Connection Load Balancing Algorithm

# Initialize server list with their current connection count
servers = [
  {“server_id”: 1, “active_connections”: 0},
  {“server_id”: 2, “active_connections”: 0},
  {“server_id”: 3, “active_connections”: 0}]

# Function to find the server with the least active connections
def find_least_connection_server(servers):
  min_connections = float(‘inf’)
  selected_server = None

  # Iterate over the servers to find the one with the least connections
  for server in servers:
    if server[“active_connections”] < min_connections:
      min_connections = server[“active_connections”]
      selected_server = server

  return selected_server

# Function to handle a new incoming request
def handle_request(request):
  # Find the server with the least active connections
  server = find_least_connection_server(servers)

  # Assign the request to the selected server
  print(“Assigning request to Server:”, server[“server_id”])

  # Increase the active connections count for the selected server
  server[“active_connections”] += 1

  # Simulate processing the request (for example purposes)
  process_request(server)

  # After processing, decrease the active connections count
  server[“active_connections”] -= 1

# Simulated request processing
def process_request(server):
  # Simulate request handling (can vary in complexity)
  print(f”Processing request on Server {server[‘server_id’]}...”)
  # Simulate processing time
  # In a real system, this would handle the actual work.
  import time
  time.sleep(1) # Simulate some processing time

# Main simulation loop
requests = 5 # Number of incoming requests

for i in range(requests):
  print(f”Handling request {i+1}”)
  handle_request(i)
```

Based on **Figure 5**, the weighted least connections algorithm is a superset of the least connections, where each server is assigned a corresponding weight to represent its processing performance. The default weight of the server is 1 and the system

administrator can dynamically set the server's permissions. The weighted least connections algorithm (Algorithm 2) aims to proportionally distribute new connections among servers based on their established connection count and respective weights. Due to variations in server performance, this algorithm will assign higher weights to servers with better performance, allowing them to receive more requests.

The second dynamic load balancing algorithm I use is Round Robin. The principle of round-robin algorithm is to allocate requests from users to servers internally in a circular manner, starting from 1 and cycling through to N (the number of internal servers), then restarting the cycle. The advantage of the algorithm lies in its simplicity; it does not need to keep track of the current state of all connections, making it a stateless scheduling method. Round-robin algorithm flow:

Algorithm 3 Pseudocode for Round Robin Load Balancing Algorithm

```
# Pseudocode for Round Robin Load Balancing Algorithm

# Initialize the list of servers with their IDs
servers = [
  {"server_id": 1},
  {"server_id": 2},
  {"server_id": 3}
]

# Variable to track the current server index
current_server_index = 0

# Function to find the next server using Round Robin
def find_next_server(servers):
  global current_server_index

  # Get the server based on the current index
  selected_server = servers[current_server_index]

  # Update the current_server_index to the next server in the list
  # If the index reaches the end of the server list, it wraps around to the first server
  current_server_index = (current_server_index + 1) % len(servers)

  return selected_server

# Function to handle a new incoming request
def handle_request(request):
  # Find the next server using the Round Robin method
  server = find_next_server(servers)

  # Assign the request to the selected server
  print("Assigning request to Server:", server["server_id"])

  # Simulate processing the request (for example purposes)
  process_request(server)

# Simulated request processing
def process_request(server):
  # Simulate request handling (can vary in complexity)
  print(f"Processing request on Server {server['server_id']}...")
  # Simulate processing time
  # In a real system, this would handle the actual work.
  import time
```

Algorithm 3 (*Continued*)

```
44:     time.sleep(1) # Simulate some processing time
45:
46:   # Main simulation loop
47:   requests = 6 # Number of incoming requests
48:
49:   for i in range(requests):
50:     print(f"Handling request {i+1}")
51:     handle_request(i)
```

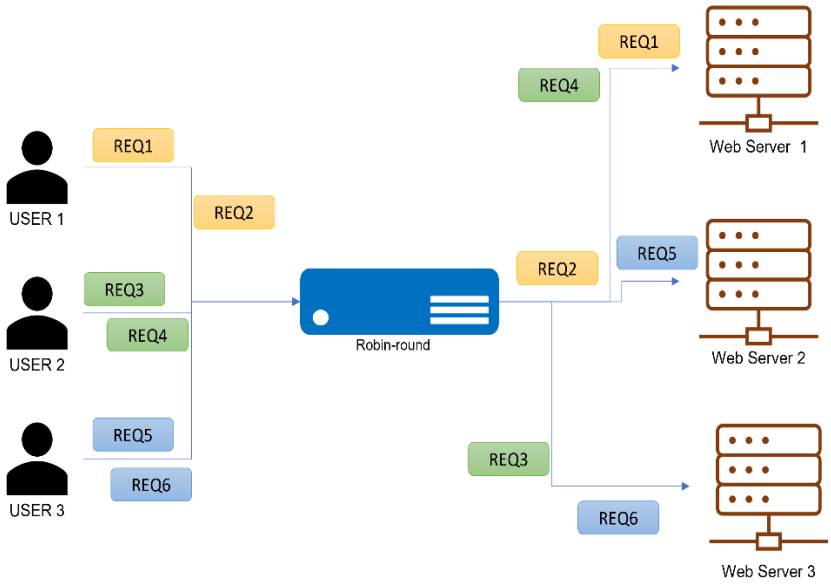

Figure 6. Round-robin diagram.

In the setup with three servers—Server 1, Server 2, and Server 3 incoming client requests are allocated to each server in a round-robin sequence: 1 → 2 → 3 → 1 → 2 → 3 → and so forth as shown in **Figure 6**. This ensures an even distribution of requests across all available servers. When managing server additions or removals, the server list is updated accordingly. The algorithm (Algorithm 3) tracks the server that should handle the next request using a server cursor. Upon receiving a new request, the algorithm selects the current server pointed to by the cursor and increments it to point to the next server in the list. This process continues in a loop, ensuring each server can process incoming requests. Implementation of the round-robin algorithm is straightforward, typically involving a list of servers and a pointer to the current server. This approach efficiently balances the load among servers without requiring complex logic or centralized management, making it suitable for various distributed computing environments.

3.2. System frontend and development

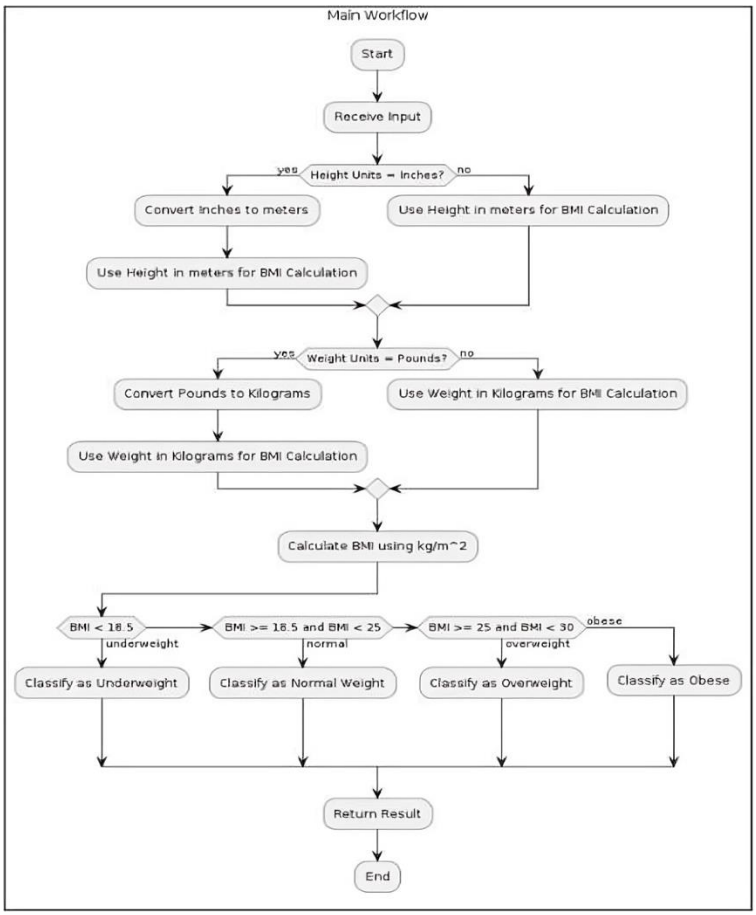

Figure 7. Window foundation diagram.

The next one is the BMI window in **Figure 7**. The depicted Windows Workflow Foundation illustrates the procedural steps involved in calculating Body Mass Index (BMI) and categorizing individuals based on their BMI values. Initially, the program receives input for height and weight. It then checks if the height units are in inches and converts them to meters if necessary. Similarly, it checks if the weight units are in pounds and converts them to kilograms accordingly. Subsequently, the BMI is calculated using the standard formula, considering the weight in kilograms and height in meters squared. Following the BMI calculation, the program determines the BMI category, classifying individuals as "Underweight," "Normal Weight," "Overweight," or "Obese" based on predefined BMI ranges. Finally, the program returns the classification result, marking the end of the process. This systematic approach enables accurate assessment and classification of individuals' body mass status, aiding in health monitoring and intervention strategies.

After explaining the BMI window foundation diagram, the front end (**Figure 8**) is the part of the system where users will be interacting, built using standard web technologies. It includes HTML for creating the webpage structure, providing the base structure for elements like forms, buttons, and text fields. CSS is used to style the HTML elements, enhancing the visual appeal, and ensuring a smoother look and feel

across the web application. CSS also allows for responsive design, ensuring that the webpage is accessible and looks good on different devices and screen sizes.

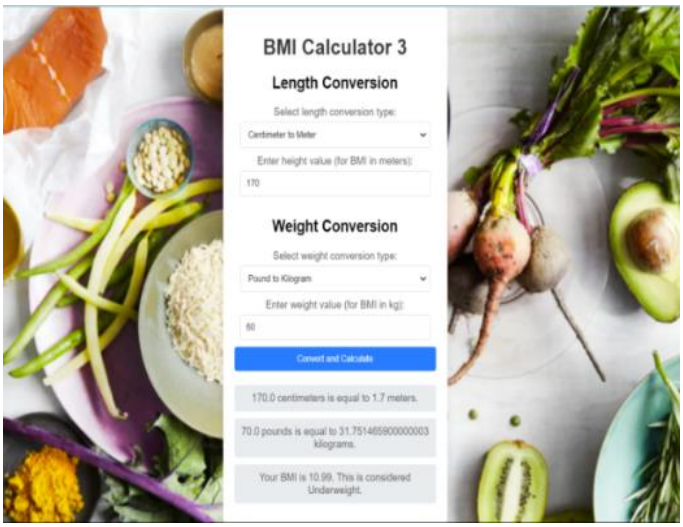

Figure 8. System interface.

4. Evaluation of findings

Tables 2 and **3** provided compare the performance of a load-balanced system using two different algorithms: Round-Robin and Least Connection. Each test was run with a ramp-up time of 60 seconds and a duration of 1 min. Here is a detailed analysis of the results for each algorithm at different load levels (10, 100, 100,1000,10,000,100,000 and 1,000,000 users). The results of the comparison are summarized in the two table below:

Table 2. Least connection test result table.

	Average Response Time	Min Response Time	Max Response Time	Throughput (sec)	Error Rate	CPU Usage (%)	Thread	Sample
10	3	0	151	771.86	0%	2	56	46880
100	1	0	147	1528.7	0%	2.3	56	93576
1000	168	0	14694	1339.2	0%	2.4	57	80848
10000	237	5	24166	464.3	0%	3.6	127	76247
100000	760	5	30224	452.2	0%	5.2	210	65380

Table 3. Round-robin test result table.

	Average Response Time	Min Response Time	Max Response Time	Throughput (sec)	Error Rate (%)	CPU Usage (%)	Thread	Sample
10	2	0	81	754.4	0	3.5	57	42509
100	4	0	127	1287.8	0.02	4.1	59	73281
1000	253	0	16089	1047.8	0.1	4.1	146	71404
10000	1072	4	15429	320.2	2.27	6.3	242	67720
100000	2039	4	11661	199.7	2.51	7.8	301	42540

The performance metrics for the system are detailed in several key columns. The Average Response Time column shows the average duration it takes for a request to be processed from the moment it is sent until a response is received. Lower values in this column indicate better performance and faster response times. The Min Response Time column represents the shortest time recorded for a request to be processed, illustrating the best-case scenario for response time during the test. Conversely, the Max Response Time column shows the longest time taken for a request to be processed, highlighting potential delays or bottlenecks in the system by reflecting the worst-case scenario. Throughput (sec) measures the number of requests processed per second. A higher throughput signifies that the system can handle requests more efficiently within a given period. The Error Rate represents the percentage of requests that failed during the test; a lower error rate is preferable as it indicates more reliable performance and fewer issues during processing. The CPU Usage column displays the percentage of CPU resources utilized during the test. Lower CPU usage implies that the system is handling the load more efficiently without overloading the processor. The Threads column shows the number of active threads used during the test, representing the concurrent users or processes being handled by the system. This value indicates how many threads were necessary to maintain the given user load. Finally, the Sample column provides the total number of requests processed during the test period, giving an overall sense of the workload handled by the system.

The study of dynamic load balancing in service composition evaluates and compares the overall performance of two load balancing algorithms, Round-Robin, and Least Connection, under different user load conditions. This analysis provides insights into how each algorithm manages system resources and maintains service quality.

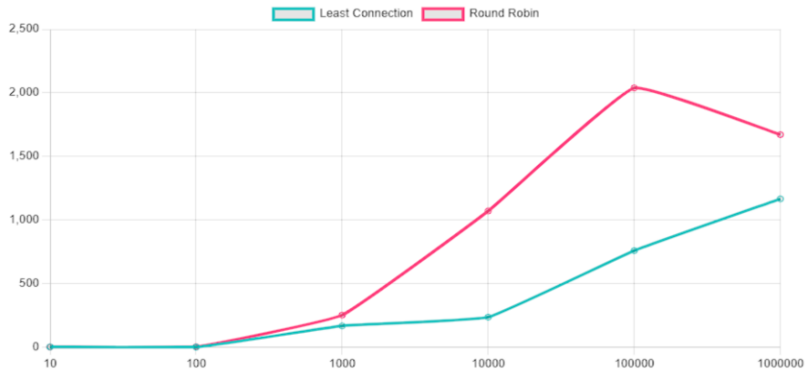

Figure 9. Average response time graph.

As shown in **Figure 9**, the average response time for the Round-Robin algorithm increases significantly with the number of users. This indicates that Round-Robin struggles to distribute the load efficiently under high traffic conditions. In contrast, the Least Connection algorithm maintains a lower average response time across all user loads, suggesting better load distribution and quicker request processing. When examining the minimum and maximum response times, the Round-Robin algorithm exhibits a wide range of response times. There is a considerable increase in the maximum response time at higher user loads, highlighting potential delays and

bottlenecks. Conversely, the Least Connection algorithm shows a more stable range of response times, with lower maximum response times compared to Round-Robin. This implies fewer delays and more consistent performance.

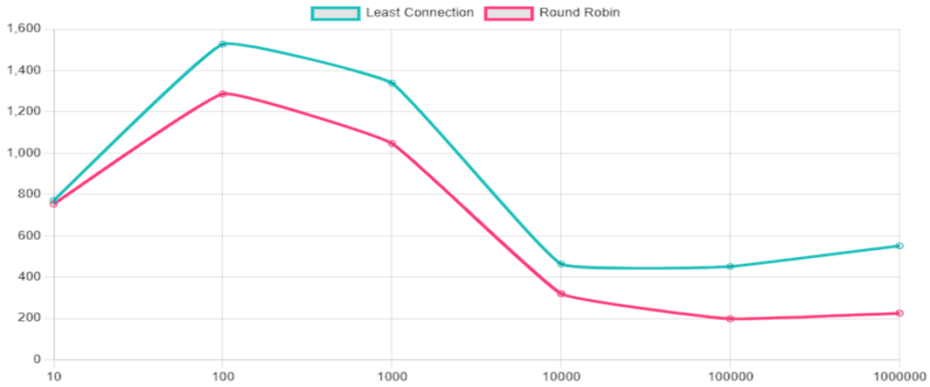

Figure 10. Throughput graph.

In terms of throughput, depicted in **Figure 10**, Round-Robin initially increases throughput but struggles at extremely high user loads, showing limitations in handling peak traffic efficiently. Least Connection, on the other hand, maintains higher throughput at all levels, particularly under heavy loads, demonstrating better scalability and efficiency in processing requests. The Error Rate further distinguishes the two algorithms. Round-Robin's error rate increases with user load, reaching up to 0.1% at 1000 users, indicating reduced reliability under heavy loads. Conversely, Least Connection consistently maintains a low error rate of 0%, even at high user loads, indicating superior reliability and fewer failed requests. CPU Usage also varies between the algorithms. Round-Robin shows constant but slightly higher CPU usage, around 7.8%, suggesting consistent but potentially inefficient resource utilization. Least Connection exhibits lower CPU usage, ranging from 2% to 5.6%, indicating more efficient use of processing power and better resource management.

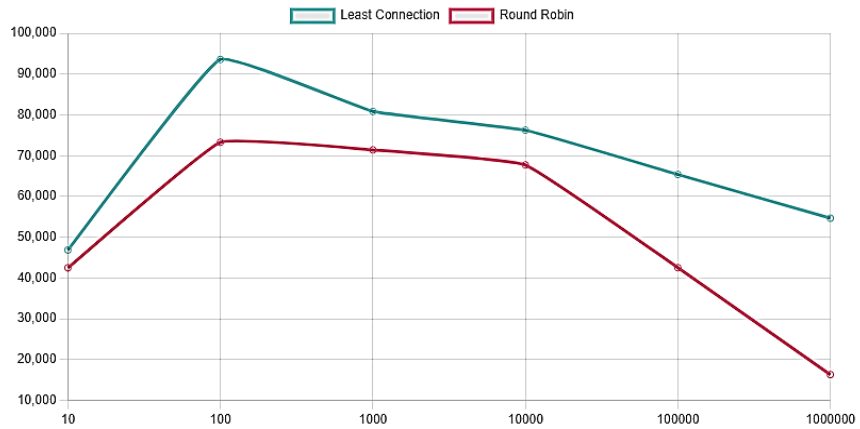

Figure 11. Threads and samples graph.

The comparison between the Round-Robin and Least Connection algorithms reveals important implications for system performance, particularly in high-traffic environments as presented in **Figure 11**. The significant increase in active threads and

resource consumption with Round-Robin underlines a critical limitation: its inability to efficiently manage varying loads. This results in higher CPU usage, increased error rates, and inconsistent response times as user demand grows. These outcomes suggest that Round-Robin, while straightforward in its implementation, is less suitable for dynamic service composition, especially under conditions of high concurrency, where its resource-intensive nature may degrade overall system performance.

On the other hand, the Least Connection algorithm's ability to maintain a stable number of active threads while processing an increasing number of samples highlights its superior efficiency in managing concurrent requests. This stability translates into more consistent and predictable performance, ensuring that system resources are not overburdened, even as user demand increases. The lower CPU consumption observed with Least Connection indicates better optimization of hardware resources, making it more energy-efficient and cost-effective in environments with fluctuating workloads. This finding is particularly significant for cloud-based and distributed systems, where optimizing resource usage can directly impact operational costs and system scalability.

Moreover, the consistently low error rates achieved by the Least Connection algorithm emphasize its reliability in maintaining high-quality service. In environments where service uptime and reliability are crucial, such as healthcare or financial services, this reliability directly influences user satisfaction and trust in the system. The improved response times and throughput under higher loads demonstrate the algorithm's scalability, making it ideal for handling growing user bases without compromising performance. These results underscore the importance of selecting the appropriate load-balancing strategy, as the Least Connection algorithm clearly provides a more balanced and efficient approach to managing dynamic workloads in service composition environments.

Discussion and research contribution

The investigation into dynamic load balancing within service composition environments provides valuable insights into the performance and efficiency of different load balancing algorithms. Through comparative analysis of Round-Robin and Least Connection algorithms under various user load conditions, several key observations and implications for service composition have emerged. This research reveals that the Least Connection algorithm consistently outperforms the Round-Robin algorithm across multiple performance metrics, especially as the user load increases. The average response time for the Least Connection algorithm remains significantly lower than that of the Round-Robin algorithm, indicating a more efficient distribution of requests. This efficiency is crucial in-service composition environments where timely processing of requests directly impacts user experience and service quality. In terms of throughput, the Least Connection algorithm demonstrates superior capability in handling higher volumes of requests per second, particularly under heavy user loads. This suggests that Least Connection can better manage scalability, an essential characteristic for dynamic and high-traffic environments typical of modern web services. The ability to maintain high throughput without a significant increase in response time underscores the robustness of the Least Connection algorithm in ensuring seamless service delivery. The research also highlights differences in

resource utilization between the two algorithms. The Least Connection algorithm exhibits lower CPU usage compared to Round-Robin, indicating more efficient use of processing power. This efficiency in resource utilization is critical for maintaining system performance and preventing resource bottlenecks, which can lead to degraded service quality or system failures. Moreover, the error rate is a critical metric for evaluating the reliability of load balancing algorithms. The consistently low error rate observed with the Least Connection algorithm, even under maximum load, points to its reliability and stability. In contrast, the increasing error rate with the Round-Robin algorithm under higher loads suggests potential issues in maintaining service reliability, due to its simplistic approach to load distribution that does not account for the current load on each server. This is our research contribution:

- Comparative Analysis of Load Balancing Algorithms: The research provides a detailed comparative analysis of two prominent load balancing algorithms, Round-Robin, and Least Connection, within the context of service composition for BMI services. This analysis offers insights into the strengths and weaknesses of each algorithm under different user loads, highlighting their impact on response times, throughput, error rates, CPU usage and overall system efficiency.
- Identification and Analysis of Load Balancing Challenges: The research addresses the specific challenges associated with dynamic load balancing in service composition environments. By thoroughly understanding these challenges, the study provides a foundation for developing more robust and adaptive load balancing techniques tailored to the unique requirements of service compositions.
- Identification of Optimal Load Balancing algorithm: The results indicate that the Least Connection algorithm performs better than Round-Robin, particularly at higher loads. My research identifies the optimal load balancing strategy that ensures lower response times, higher throughput, and lower error rates, thereby contributing to the optimization. The improvements in response time and resource utilization can be attributed to the novel adaptive mechanism introduced by the proposed algorithm, which outperforms static methods by continuously adjusting resource distribution in response to real-time performance metrics.

Least Connection outperformed Round-Robin across all key performance metrics, particularly in response time, throughput, error rate, and CPU usage, making it the preferred choice for environments that demand high scalability and reliability. While Round-Robin is simpler to implement, its failure to account for real-time server load leads to inefficiencies under high user loads, especially in terms of CPU consumption and thread management. The consistently lower error rates and optimized resource usage with Least Connection highlight its ability to provide a more stable and reliable service in dynamic environments where user demand is unpredictable. Additionally, Least Connection’s lower CPU usage and balanced thread management make it better suited for environments where cost-efficiency and scalability are critical factors.

5. Conclusion

In this research, we have made significant contributions to addressing the challenges of optimizing resource utilization and ensuring high availability in service composition environments. Through the development and validation of a dynamic load-balancing algorithm, this study has demonstrated improvements in system performance, reducing response times, enhancing throughput, and ensuring efficient use of computational resources. The proposed algorithm's ability to provide real-time insights into critical system parameters, such as CPU usage, thread activity, and network performance, makes it a promising tool for real-world applications, particularly in environments requiring continuous service availability.

However, several limitations emerged that warrant further investigation. The scalability of the algorithm, particularly in large-scale, real-world environments, remains under-explored. The current validation, conducted in a controlled, simulated environment with a BMI calculator service, may not fully capture the complexities encountered in more diverse, high-demand service compositions. Moreover, the algorithm's fault tolerance capabilities, while effective under standard conditions, need to be rigorously tested in scenarios involving unpredictable network failures and volatile traffic patterns.

Looking forward, future research will address these limitations by expanding the scope of testing to include larger, heterogeneous service compositions and real-world case studies. Enhancing the algorithm's scalability will be a priority, ensuring it can efficiently manage highly dynamic, large-scale systems. Additionally, refining the fault tolerance mechanisms to better handle unexpected network disruptions and more complex service dependencies will be critical. Furthermore, optimizing the algorithm for various industry-specific use cases—such as healthcare, telecommunications, and cloud-based systems—will increase its practical value and adaptability across multiple domains.

This research not only bridges the gap between theoretical exploration and practical application in dynamic load balancing but also provides a solid foundation for future innovations in service composition. By advancing the state of the art in high-availability service-oriented architectures, this study contributes significantly to the growing body of knowledge in the field, offering both academic and practical implications for the design of resilient, scalable, and efficient systems.

Author contributions: Conceptualized the research framework, implemented the load-balancing algorithms, WKL; oversaw the research design, provided critical revisions, managed the overall project, RKR; literature review, assisted with the analysis of performance metrics, VR. writing, review, and final approval of the manuscript, WKL, RKR and VR. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported by the Fundamental Research Grant Scheme (FRGS) under Grant Code MMUE/220033, which aims to promote and facilitate innovative research initiatives in various fields of study.

Conflict of interest: The authors declare no conflict of interest.

References

A. Kanso and Y. Lemieux, “Achieving High Availability at the Application Level in the Cloud,” Jun. 2013. Available: https://www.researchgate.net/publication/259216367_Achieving_High_Availability_at_the_Application_Level_in_the_Cloud.

A. M. Alakeel, “A guide to dynamic load balancing in distributed computer systems,” International Journal of Computer Science and Information Security, vol. 10, no. 6, pp. 153-160, 2010.

C. Begau and G. Sutmann, “Adaptive dynamic load-balancing with irregular domain decomposition for particle simulations,” Computer Physics Communications, vol. 190, pp. 51-61, 2015, doi: 10.1016/j.cpc.2015.01.009.

Chawla, K. (2024). Reinforcement Learning-Based Adaptive Load Balancing for Dynamic Cloud Environments. arXiv preprint arXiv:2409.04896.

Chen, W., Zhu, Y., Liu, J., & Chen, Y. (2021). Enhancing mobile edge computing with efficient load balancing using load estimation in ultra-dense network. Sensors, 21(9), 3135.

D. Kanellopoulos and V. K. Sharma, “Dynamic Load Balancing Techniques in the IoT: A Review,” Symmetry, vol. 14, no. 12, p. 2554, 2022, doi: 10.3390/sym14122554.

D. Saxena and A. K. Singh, “A high availability management model based Tawfeeg, T. M., Yousif, A., Hassan, A., Alqhtani, S. M., Hamza, R., Bashir, M. B., & Ali, A. (2022). Cloud dynamic load balancing and reactive fault tolerance techniques: a systematic literature review (SLR). IEEE Access, 10, 71853-71873.

D. Thomson, “Application of Service Oriented Architecture to Distributed Simulation,” in AIAA Modeling and Simulation Technologies Conference and Exhibit, 2008, doi: 10.2514/6.2008-7091.

F. Aladwan, A. Alzghoul, E. Ali, H. Fakhouri, and I. Alzghoul, “Service Composition in Service Oriented Architecture: A Survey,” Modern Applied Science, vol. 12, no. 11, pp. 18-28, 2018, doi: 10.5539/mas.v12n12p18

G. Lokesh and K. K. Baseer, “An architecture for dynamic load balancing in cloud environment,” in 2023 2nd International Conference on Edge Computing and Applications (ICECAA), Namakkal, India, 2023, pp. 84-91, doi: 10.1109/ICECAA58104.2023.10212311.

Gupta, M. R., & Sharma, O. P. (2024). A Review exploration of Load Balancing Techniques in Cloud Computing. Educational Administration: Theory And Practice, 30(2), 580-590.

H. Wang, Y. Wang, G. Liang, Y. Gao, W. Gao, and W. Zhang, “Research on load balancing technology for microservice architecture,” MATEC Web of Conferences, vol. 336, p. 08002, 2021, doi: 10.1051/matecconf/202133608002.

H. Zeilinger and T. Anees, “SOA model for high availability of services,” Jun. 2013. Available: https://www.researchgate.net/publication/263926763_SOA_Model_for_High_Availability_of_Services.

He, H., Wang, L., Liu, J., & Qin, L. (2024). Optimizing Cloud Service Load Balancing Through Heat Conduction Equation Applications. International Journal of Heat & Technology, 42(1).

J. Giao, A. A. Nazarenko, D. Gonçalves, and J. Sarraipa, “A Framework for Service-Oriented Architecture (SOA)-Based IoT Application Development,” Processes, vol. 10, no. 9, p. 1782, 2022, doi: 10.3390/pr10091782.

J. S. Hurwitz, R. Bloor, M. Kaufman, and F. Halper, Service Oriented Architecture (SOA) For Dummies. John Wiley & Sons, 2009. Available: https://books.google.com.my/books?hl=en&lr=&id=8uM5pTncAO4C&oi=fnd&pg=PA3.

Khan, A. R. (2024). Dynamic Load Balancing in Cloud Computing: Optimized RL-Based Clustering with Multi-Objective Optimized Task Scheduling. Processes, 12(3), 519.

Lohumi, Y., Gangodkar, D., Srivastava, P., Khan, M. Z., Alahmadi, A., & Alahmadi, A. H. (2023). Load Balancing in Cloud Environment: A State-of-the-Art Review. IEEE Access, 11, 134517-134530.

M. H. Valipour, B. Amirzafari, K. N. Maleki, and N. Daneshpour, “A brief survey of software architecture concepts and service oriented architecture,” in 2009 2nd IEEE International Conference on Computer Science and Information Technology, Beijing, China, 2009, pp. 34-38, doi: 10.1109/ICCSIT.2009.5235004.

M. R. Mesbahi, A. M. Rahmani, and M. Hosseinzadeh, “Reliability and high availability in cloud computing environments: A reference roadmap,” Human-Centric Computing and Information Sciences, vol. 8, no. 1, p. 1-31, 2018, doi: 10.1186/s13673-018-0143-8.

M. Tsai and C. Wang, “Computing coordination-based fuzzy group decision-making (CC-FGDM) for web service oriented architecture,” Expert Systems with Applications, vol. 34, no. 4, pp. 2921-2936, 2008, doi: 10.1016/j.eswa.2007.05.017.

P.-W. Zhang, J.-X. Chen, and X.-Y. Jia, “A prediction based adaptive load balancing algorithm in SOA,” Microelectronics &

Computer, vol. 28, no. 11, pp. 174-177, 2011. Available: http://www.journalmc.com/en/article/id/17b5cf43-c512-4d5a-b361-c564e29c6501.

P.-Y. Zhang, S. Y. Technology, and Nanjing, "Dynamic Web service composition," Journal of Software, Nov. 2018. Available: https://www.jsjkx.com/CN/abstract/abstract8055.shtml.

Rajendran, V., Ramasamy, R. K., & Mohd-Isa, W. N. (2022). Improved eagle strategy algorithm for dynamic web service composition in the IoT: a conceptual approach. Future Internet, 14(2), 56.

Ramasamy, R. K., Chua, F. F., Haw, S. C., & Ho, C. K. (2022). WSFeIn: A Novel, Dynamic Web Service Composition Adapter for Cloud-Based Mobile Application. Sustainability, 14(21), 13946.

S. Namuye, L. Mutanu, G. Chege, and J. Macharia, "Leveraging health through the enhancement of information access using Mobile and service oriented technology," in 2014 IST-Africa Conference Proceedings, Pointe aux Piments, Mauritius, 2014, pp. 1-9, doi: 10.1109/ISTAFRICA.2014.6880661

S. T. Waghmode and B. M. Patil, "Optimised and adaptive dynamic load balancing in the distributed database server," in 7th International Conference on Computing in Engineering & Technology (ICCET 2022), Feb. 2022, pp. 145-149, doi: 10.1109/ICCET2022.9800278.

S. Wang, Y. Gong, G. Chen, Q. Sun, and F. Yang, "Service vulnerability scanning based on service-oriented architecture in Web service environments," Journal of Systems Architecture, vol. 59, no. 9, pp. 731-739, 2013, doi: 10.1016/j.sysarc.2013.01.002.

Syed, D., Muhammad, G., & Rizvi, S. (2024). Systematic Review: Load Balancing in Cloud Computing by Using Metaheuristic Based Dynamic Algorithms. Intelligent Automation & Soft Computing, 39(3).

T. Wira Harjanti, H. Setiyani, and J. Trianto, "Load Balancing Analysis Using Round-Robin and Least-Connection Algorithms for Server Service Response Time," Applied Technology and Computing Science Journal, vol. 5, no. 2, pp. 119-128, 2022, doi: 10.33086/atcsj.v5i2.3743.

V. Indhumathi and G. M. Nasira, "Service oriented architecture for load balancing with fault tolerant in grid computing," in 2016 IEEE International Conference on Advances in Computer Applications (ICACA), 2016, pp. 313-317, doi: 10.1109/ICACA.2016.7887972.