DEThresh: Enhancing Cloud Computing Performance with Differential Evolution-Driven Datacenter Selection and Threshold-based Load Balancing Optimization

Ajay Katangur

ajaykatangur@missouristate.edu

Missouri State University

Shusmoy Chowdhury

Missouri State University

Research Article

Keywords: Cloud, Datacenter Selection, Differential Evolution, Load Balancing, Overload, Threshold, Underload

Posted Date: February 7th, 2024

DOI: https://doi.org/10.21203/rs.3.rs-3929293/v1

Additional Declarations: No competing interests reported.

DEThresh: Enhancing Cloud Computing Performance with Differential Evolution-Driven Datacenter Selection and Threshold-based Load Balancing Optimization

Shusmoy Chowdhury and Ajay Katangur

Department of Computer Science, Missouri State University, 901 S National Ave, Springfield, 65897, Missouri, USA.

Contributing authors: shusmoy26@missouristate.edu; ajaykatangur@missouristate.edu;

Abstract

The advent of cloud computing has introduced a novel paradigm in user services, enabling on-demand access to information technology services, irrespective of time and location, with a pay-per-use model. Meeting user expectations necessitates cloud service providers to furnish an infrastructure that is efficient, reliable, and resilient to faults. Addressing the substantial volume of user requests, selecting cloud datacenters and load balancing strategies become pivotal for delivering real-time services economically and selecting appropriate datacenters and virtual machines. This research introduces DEThresh, combining a smart differential evolution-based datacenter selection method with a threshold-based load balancing mechanism aimed at enhancing resource management by focusing on cost and performance considerations. The differential evolution-based datacenter selection employs evolutionary operations to identify the most suitable DC for the userbase. Simultaneously, threshold-based load balancing ensures the reasonable distribution of tasks and optimal resource utilization, preventing task overload on virtual machines and underutilization due to task scarcity. Simulation results demonstrate that the proposed DEThresh algorithm exhibits superior response and data processing time compared to well-established

popular algorithms currently in use, especially under varying user-base and datacenter conditions. Impressively, DEThresh achieves this even with fewer datacenters, ultimately reducing cloud service costs.

Keywords: Cloud, Datacenter Selection, Differential Evolution, Load Balancing, Overload, Threshold, Underload

1 Introduction

Growth of information technology has been enhanced in recent years. With this rapid growth, our daily lifestyle has been improved by relying on the latest technology available in the market. Cloud services have emerged as an appropriate substitution for traditional computing for clients. The services provide services to clients based on their location, time, and pricing policies [1].. Through these services, clients can access the pool of configurable computing resources (applications, servers, networks, storage). Many cloud providers, such as Google, IBM Cloud, Rackspace, Amazon Web Services (AWS), Microsoft Azure, Red Hat, Verizon Cloud, and VMware, provide these services to clients.

The primary objective of the cloud is to use the distributed resources effectively to achieve high throughput and performance. Cloud can resolve problems that need high computing power using those resources. Cloud distributes its resources all around the world. It executes the tasks at different datacenters (DCs) and provides faster services to clients [2]. Cloud Computing has many resource distribution systems, such as grid computing and peer-to-peer computing, which allow resource-sharing and data transfer facilities [3–5].

The cloud can be divided based on two criteria: location and services. Clouds can be categorized as public, private, hybrid, and community based on location. Public clouds are accessible to everyone, and the location of the cloud infrastructure will be on the premises of the service-providing company. The public cloud can be cost-effective but is highly vulnerable to attacks. Private clouds are specific to the users or organizations. It ensures the most increased security and control level for the user. On the contrary, the maintenance cost of the private cloud is very high compared to the public cloud. Hybrid clouds are a combination of private and public clouds. A hybrid cloud is designed based on the organizational requirements to serve distinct purposes. Community clouds are common cloud infrastructures with shared data and management used by multiple organizations.

Classification of cloud considering the services can be defined as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS)[6, 7]. Within the IaaS model, the cloud provides fundamental information technology resources encompassing networking capabilities and computational entities. This framework affords enhanced flexibility and heightened control over the allocation and utilization of computing resources. PaaS alleviates an enterprise's need to manage fundamental infrastructure

elements, such as computer hardware and operating systems. This enables a directed emphasis on the seamless deployment of applications. SaaS allows users to focus on utilizing specific software applications, alleviating the need for considerations regarding managing underlying infrastructure and services.

Besides these services, the cloud also provides other services like Storage as a Service (SaaS), Database as a Service (DaaS), Communication as a Service (CaaS), Network as a Service (NaaS), Security as a Service (SECaaS), Monitoring as a Service (MaaS), Expert as a service (EaaS) and Testing as a service (TaaS) [8, 9]. Cloud clients use these services for application purposes without knowing the technology required for their computing environment. These applications can be varied based on education, health monitoring [10], data analysis, and robotics. The cloud infrastructure affords a dynamic and scalable framework, allowing for the provisioning and procuring of diverse and configurable resources under the specific requirements of a given application [11].

The provision of diverse services by the cloud necessitates implementing quality of service (QoS) monitoring. This imperative practice evaluates the delivered services, ensures their alignment with user demands, and upholds the stipulations outlined in the service level agreement (SLA). As the cloud needs to maintain the QoS and SLA, it may face several issues and challenges consisting of datacenter (DC) selection [12–30], Load balancing [31–42], performance analysis and modeling [43–48], throughput and response time [49], security and privacy issues [50–54], resource management [55, 56] and QoS. DC selection and load balancing policy are the main challenges regarding QoS and SLA. DC selection allocates the most suitable DC for the userbase based on their requirement so that the resources in the cloud can be utilized properly. Furthermore, load balancing processes the user request in the virtual machines (VMs) of the DCs without making the VMs overloaded with user requests or idle for lack of user requests.

The escalating adoption of cloud services has resulted in a proportional increase in the expenses associated with constructing new DCs. A primary determinant influencing the cost dynamics of the cloud is the low resource utilization ratio, given that the establishment and maintenance of additional DCs entail significant financial outlays. Notably, many DCs exhibit suboptimal resource utilization, with ratios ranging from 5% to 15% [57]. The strategic selection of DC locations across diverse regions can mitigate this inefficiency by enabling a limited number of strategically positioned DCs to effectively serve a substantial portion of the global userbase. This approach diminishes the imperative for numerous smaller DCs and allows existing DCs to dynamically scale up during peak usage periods, thereby mitigating the necessity for additional facilities. To further enhance efficiency, optimal DC selection algorithms play a pivotal role in ensuring an equitable distribution of workloads while simultaneously minimizing response and data processing times. The robust

and optimal selection of DCs becomes imperative in aligning with the predetermined user requirements and the prevailing state of the cloud environment for each incoming request.

Load balancing constitutes the strategic allocation of resources and proficient task assignment to mitigate response time and enhance the utilization of resources. This practice ensures the equitable distribution of workloads across available resources, such as network links, central processing units, and disk drives, to attain optimal throughput and response time while preventing the overload of any individual resource [58]. Load balancing algorithms play a pivotal role in realizing the objectives of cloud computing. Nevertheless, formulating effective load balancing algorithms encounters numerous challenges, including the dispersion of DCs and nodes across geographic locations, network and communication delays, and the dynamic evolution of user requirements. In response to these challenges, load balancing algorithms necessitate designs capable of managing node failures, implementing virtual machine (VM) migration strategies to transfer overloaded VMs to remote locations, and accommodating heterogeneous nodes to optimize resource utilization and minimize response time. Effective storage management is imperative for achieving optimal resource utilization. The on-demand accessibility and scalability of cloud services empower users to access services promptly and scale their usage up or down as required. Lastly, the intricacy of load balancing algorithms must be mitigated to ensure efficient performance.

In this study, we have introduced a novel DEThresh approach designed to enhance the performance and maintain QoS and SLA of cloud services. DEThresh integrates an evolutionary-based differential evolution (DE) DC selection policy with a threshold-based load balancing strategy. Utilizing the DE algorithm involves evolutionary processes such as mutation, recombination, and selection, thereby guiding the progression of DC selection for cloud userbases. Evolutionary algorithms, inspired by the principle of survival of the fittest, continually refine solutions over time.

The threshold-based load balancing policy categorizes VMs as either underloaded or overloaded, directing task allocation based on the prevalent load conditions and computation power of the VMs. This classification relies on two threshold values, underload and overload thresholds, which aid in distinguishing between VMs experiencing low or high loads. Task assignment to VMs is suspended when the overload threshold is reached, while the underload threshold facilitates the resumption of task delegation to identified VMs. Introducing a well-defined gap between these threshold values mitigates frequent oscillations in VM categorization, ensuring stability in determining whether a VM is underloaded or overloaded.

The subsequent sections of this paper are organized as follows: Section 2 furnishes an extensive examination of prior research, delineating diverse DC selection algorithms and load balancing policies applied in cloud computing. In Section 3, we delve into the intricacies of the cloud DC selection and load balancing issue under consideration, elucidating the research objectives we aim

to address. Section 4 offers a comprehensive presentation of the DEThresh algorithm, detailing its underlying principles and operational intricacies. The experiment setup and a meticulous analysis comparing the performance of the proposed DEThresh algorithm against widely used DC selection and load balancing algorithms in the cloud are expounded in Section 5. Additionally, Section 5 scrutinizes variations in outcomes resulting from adopting different userbase sizes and numbers of DCs, providing evidence of the cost-effectiveness of our DEThresh algorithm. Finally, Section 6 serves as the culmination of this research, summarizing our contributions and delineating potential avenues for future extensions and exploration within this domain.

2 Related Work

Optimizing resource provisioning in a cloud environment necessitates formulating a DC selection policy that strategically balances cost and performance considerations [59]. Likewise, implementing a robust load balancing mechanism is designed to schedule user requests with minimal latency. Notably, recent research endeavors have been directed toward investigating and enhancing cloud-based DC selection and load balancing methodologies.

2.1 DC Selection Policy

Several algorithms for DC selection policies have been proposed to choose the most suitable DC to enhance user satisfaction and ensure efficient resource utilization. However, the determination of decision factors varies among these proposed approaches. Notably, some DC selection policies consider the overall cost a critical decision factor [12–14, 60]. In contrast, others prioritize response time [15–18], and another set focuses on data processing time [20, 61]. Regrettably, those strategies aimed at minimizing overall cost tend to increase response time without considering diverse workloads, bandwidth, and network latency. Similarly, approaches designed to reduce response time often increase overall cost, data processing time, resource underutilization, and overload on the selected DC [15, 21, 22].

In the real world, there are two conventional DC selection policy approaches. The first is the static approach, known as the closest datacenter or proximity-based DC selection policy, which selects the nearest DC for user requests, potentially causing overload when multiple requests originate from the same region. Devyaniba Chudasama et al. [23] and Dhaval Limbani et al. [24] introduced an enhanced or extended proximity-based routing policy to address this issue. This policy facilitates the selection of the nearest DC based on minimizing data transfer costs. Consequently, requests can be directed to other DCs with minimal transfer costs in cases of DC overload, thereby enhancing overall performance. Sharma et al. [12] proposed a round-robin DC selection, which distributes user requests from a region across different DCs within the same region. This prevents any single DC from becoming overloaded with a substantial workload. Simulation results indicate improved performance

under identical DC configurations; however, variability in configurations may yield unsatisfactory outcomes.

Addressing this concern, Misra et al. [25] developed a priority-based Round Robin service broker policy, allocating workload among available DCs to optimize overall cost and minimize response time. Nevertheless, improper DC selection can elevate overall costs and response times due to unfair resource allocation and network latency. In response, Bhavani and Guruprasad [26] devised an efficient resource allocation model based on network delay between user bases and DCs, successfully directing user requests to suitable DCs. Nandwani et al. [27] introduced a weight-based DC selection algorithm in the cloud environment, enabling appropriate DC selection based on the weights assigned to each DC. Results from this algorithm demonstrate reduced response time, processing time, and overall costs.

Zakaria Benlalia et al. [62] integrated considerations of cost and efficiency in an endeavor to optimize both user cost and response times. The policy introduced in this study prioritizes the selection of the most suitable DC based on a function involving the efficiency-to-cost ratio. Given the multifaceted nature of DC selection algorithms, which must efficiently choose the optimal DC for user requests while accounting for variables such as time, cost, and availability, the authors emphasize the importance of the efficiency-to-cost ratio. It is chosen if a DC with the minimum balance is available; otherwise, the closest available DC is selected. Notably, the authors should have furnished a statistical performance comparison of the proposed algorithm relative to other baseline algorithms.

Furthermore, Kunal Kishor et al. [27] proposed a broker policy utilizing proportionate weights to determine the selection of a DC for servicing a specific user request. The incorporation of proportional weights ensures an equitable distribution of workload among DCs.

The second approach involves the dynamic selection of a DC-based on user requirements at a specific moment, referred to as the dynamic DC selection policy. In the optimized response time DC selection policy, a DC is chosen based on its superior response time, and this policy maintains a sorted list of DCs according to their response times. While the optimized response time DC selection policy yields satisfactory results, it can potentially increase the data processing time of the selected DC, leading to an overload of tasks. Tailong et al. [63] introduced a modified optimized response time algorithm to refine the prevailing optimized response time DC selection policy within the CloudAnalyst framework. The algorithm systematically computes individual processes' response time and waiting time, subsequently determining the optimal scheduling process. Although this algorithm exhibits a notable reduction in response time compared to the optimized response time DC selection policy under predefined criteria, it manifests limitations in adaptability to dynamic cloud environments due to its fixed nature.

Researchers have also explored using metaheuristic search or evolutionary algorithms to identify the optimal DC for the user base. Manasrah et

al. [22] introduced a variable service broker routing policy (VSBRP) as a heuristic-based technique to minimize response time by selecting a DC based on communication channel bandwidth, latency, and job size. This approach involves redirecting user requests to the following DC using VSBRP to improve response and processing times for specific jobs while preventing DC overload.

In a different vein, Alfonso Quarati et al. [64] proposed a genetic approach for DC selection, concentrating on resource allocation for applications with diverse QoS requirements. This DC selection approach supports cloud brokering in allocating batches of jobs to hybrid cloud infrastructures. Ahmad M. Manasrah et al. [29] introduced a differential evaluation-based service broker policy to identify an optimal DC for requests based on user constraints. The proposed service broker policy aims to enhance response time by minimizing the overall cost, which comprises VM and data transfer costs.

Yacine Kessaci et al. [30] employed a Pareto-based metaheuristic approach to devise a service broker policy. The algorithm, named Multi-objective Genetic Algorithm Cloud Brokering (MOGA-CB), considers response time and the cost of VMs as two objectives, demonstrating efficiently effective Pareto sets of solutions. Najib A. Kofahi et al. [65] put forth an efficient service broker routing policy that enhances user satisfaction and cloud performance. This policy utilizes the Vector Space Model and a multi-objective scalarization function to optimize conflicting objectives. The authors also identified crucial factors for DC selection in the cloud environment, including financial cost, DC capabilities and availability, communication channel specifications, and user requirements.

The present study underscores the imperative utilization of metaheuristic algorithms in facilitating the judicious selection of cloud DCs for enhanced efficiency. Specifically, the DE algorithm emerges as a preeminent stochastic optimization technique with notable robustness in addressing real-parameter scenarios. Continuously adapted and refined by researchers, the DE algorithm has demonstrated its efficacy in resolving optimization challenges [62, 66, 67]. Notably, a modification proposed in [62] aimed at surmounting constrained optimization issues involves the adaptive adjustment of the scale factor and recombination rate based on a uniform distribution. Consequently, this adaptive approach ensures a harmonious equilibrium between global and local searches, thereby fostering an efficient solution for space exploration.

2.2 Load Balancing Policy

Following the selection of a DC, the user's request undergoes comprehensive analysis and is then forwarded to the chosen DC, contingent upon the availability of resources. It is imperative to ensure that the servers (VMs) within the DC are neither excessively burdened nor underutilized, necessitating a uniform workload distribution. The application of load balancing emerges as a pivotal concept in this context, serving to sustain optimal performance in cloud applications. An efficient load balancer is essential for this purpose, as unequal load distribution can arise from various factors, with task scheduling

being a significant contributor. Without proper task scheduling, the utilization of resources would be suboptimal. Load Balancing encompasses two fundamental objectives: resource allocation and task scheduling [33]. Firstly, assigning tasks to appropriate VMs must transpire, avoiding overloaded or vacant nodes and ensuring an equitable workload distribution. Subsequently, post-allocation task scheduling techniques are used to optimize task completion based on user requirements, meeting specified deadlines articulated in the SLA.

Amandeep Kaur Sidhu et al. comprehensively examined various load balancing techniques and the associated challenges in cloud computing [31]. The paper delineates load balancing algorithms into two principal classifications: system load and system topology. Three distinct approaches emerge regarding system load: static, dynamic, and mixed. Similarly, within the system topology, three types are identified: load balancing with predefined static rules, dynamic load balancing, and an adaptive approach that adjusts load distribution in response to system status changes by dynamically modifying parameters and algorithms. The study emphasizes key metrics for effective load balancing in the cloud, encompassing system scalability with any finite number of nodes, optimized resource utilization, enhanced performance at a reasonable cost, minimized response time, and reduced overhead associated with each node. Formulating an effective load balancing technique necessitates the consideration of major objectives such as the cost-effectiveness of the method, scalability and flexibility of the system, and the prioritization of resources or tasks. The paper enumerates a notable array of currently employed load balancing algorithms, including Round Robin, Connection Mechanism, Randomized, Equally Spread Current Execution algorithm, Throttled Load Balancing Algorithm, A task scheduling Algorithm Based on Load Balancing, Biased Random Sampling, Min-Min Algorithm, Max-Min Algorithm, and Token Routing. The authors discuss the advantages and challenges associated with each algorithm.

The round-robin (RR) Algorithm operates in a circular and ordered manner, assigning each process a fixed time slot without priority. Widely employed for its simplicity, a common challenge in load balancing arises from the need to save and update the VM allocation state post-user requests. Kaurav and Yadav et al. proposed an enhancement to RR using the Genetic Algorithm (GA) to enhance load balancing efficiency in DCs [39]. The GA-based approach resolves RR issues by scanning a hash map containing all VMs, allocating tasks to available VMs, or selecting the best-fit VM through GA analysis. Results indicate a significant reduction in server Response Time, but the complexity of GA increases with a more extensive search space [40].

Issawi et al. addressed Quality of Service (QoS) in cloud applications by tackling burstiness workload issues in load balancing [41]. They introduced an Adaptive Load Balancing algorithm (Adaptive LB (RR + Random)), dynamically alternating between RR and Random task scheduling policies based on workload conditions. While Adaptive LB reduces Response Time, the static quantum in RR leads to increased waiting time. Richhariya et al.'s hybrid approach combines RR with a priority policy [42]. The Improved RR approach

incorporates two processors: a small one to calculate time slices and a main processor to arrange processes based on burst time as the priority. The objective is to reduce Response Time, although performance testing results are not provided.

The Throttled Load Balancing algorithm (TLB) is a dynamic load balancing algorithm that seeks an appropriate VM to execute tasks upon receiving a client request [32]. TLB maintains an index table, also known as the allocation table, containing information about all VMs, including their respective states (e.g., available, busy, idle). When a request is received, TLB checks the availability and capacity of VMs. If an available VM with sufficient space is found, the task is accepted and allocated to that VM. If no available VM is identified, TLB returns 1, indicating the request is queued for expedited processing [34, 35]. TLB outperforms the RR algorithm but lacks consideration for advanced load balancing requirements, such as processing time.

Nayak et al. [36] introduced a Modified Throttled Load Balancing algorithm, which maintains an index table of VMs and their states. The authors improved Response Time and VM utilization by selecting the first available VM for task assignment. The VMs are then shifted after the initial selection, deviating from the traditional TLB, where the VM at the first index is consistently chosen for every request.

Gosh et al. [37] proposed a priority-based approach, termed Priority Modified Throttled Load Balancing (PMTLB), that enhances the execution time of the existing algorithm. PMTLB focuses on allocating incoming tasks using a queue to prioritize high-priority tasks over low-priority ones, thereby ensuring an equitable workload distribution among multiple VMs. Although this approach improved response time and waiting time compared to TLB and RR algorithms, it may still lead to starvation and high response time for low-priority jobs.

Phi et al. [38] addressed workload distribution by maintaining two tables of VMs with states: available and busy. In contrast to traditional TLB, which uses a single table for all VMs, this dual-table approach facilitates the detection of VM availability. However, this algorithm's marginal improvement in response time suggests further optimization to enhance its overall performance.

Load balancing is a critical mechanism that mitigates the risk of overloading or underloading VMs during computational tasks. So, it is crucial to understand and address the factors influencing load balancing, culminating in formulating a proficient load balancing technique tailored for the cloud environment. This research endeavors to employ a dual threshold-based approach to preemptively avert the overloading and underloading of VMs, thereby optimizing the utilization of available resources.

2.3 Combined DC Selection and Load Balancing Policy

Minhaj Ahmad Khan et al. [68] introduced the Normalized Hybrid Service Brokering with Throttled Round-Robin load balancing (NHSB_TRR) to deliver highly efficient and cost-effective cloud services. This approach involves the

generation of normalized values for both static and dynamic parameters, optimizing them for efficiency and cost within specific intervals. The selection of a DC is based on the minimum sum of normalized values, and a weighted threshold is employed to distribute the load among static and dynamically selected DCs. Load balancing is achieved through a round-robin method applied to VMs triggered by notifications from the DC controller. Experimental assessments were conducted across various configurations encompassing diverse user bases and DCs. Utilizing CloudAnalyst for simulation, the results demonstrate that the NHSB_TRR approach significantly enhances efficiency and minimizes costs, surpassing established approaches such as ORT_RR, CDC_RR, ORT_THR, and ORT_ES. Overall, the NHSB_TRR approach exhibits reduced response time and DC processing time in 70.83% of cases, with improvements reaching up to 17.39% and 31.35% over alternative methods. Additionally, it incurs lower costs in 72.22% of cases compared to other service brokering approaches. The load balancing mechanism within the service brokering outperforms round-robin and equally spread load balancing approaches by 29.98% and 17.30%, respectively.

Naha et al. [69] proposed three distinct DC selection algorithms—Cost-Aware (CA), Load-Aware (LA), and Load-Aware Over Cost (LAOC)—as well as a load balancing algorithm named State-Based Load Balancing (SBLB). The CA algorithms prioritize DCs with the most economical charges, while the LA algorithm evenly distributes requests across all available DCs. The LAOC algorithm, incorporating a cost table, dynamically allocates traffic to DCs with the lowest costs. The SBLB algorithm maintains two tables based on VM states, categorizing VMs as busy or available. When a VM reaches its usage threshold, it transitions to the busy state; otherwise, it remains available. The load balancer selects available VMs based on user requests and updates the table accordingly. Without available VMs, the DC controller queues requests until resources become available. The load balancer reallocates VMs for subsequent tasks upon completion of processing. Simulation results indicate that cost-efficient brokering incurs more significant processing and response times, while faster brokering results in higher utilization costs. The broker dynamically decides whether to prioritize cost savings or expedited processing based on user requirements. Furthermore, simulations demonstrate that the SBLB algorithm outperforms existing algorithms in a simulated environment, suggesting that an effective load balancing algorithm can minimize execution time to benefit both cloud users and providers.

In this paper, we advance a comprehensive resolution denoted as DEThresh to address the challenges of selecting DCs and load balancing in cloud services. Utilizing a differential evolution-based approach for DC selection facilitates the identification of the most optimal DC based on the dynamic requirements of the userbase. After DC selection, the load balancing mechanism, predicated on threshold principles, ensures equitable workload distribution among VMs, mitigating issues such as VM overloading and underloading. This research's primary objectives encompass minimizing userbase response time, reducing

data processing time for DCs, and concurrent optimizing the overall cost associated with cloud services.

3 Problem Formulation for Cloud

3.1 Problem Statement

Let us assume a Cloud C which consists of userbase UB and datacenter DC and defined as follows:

$$C \leftarrow \{UB, DC\}$$

Userbase UB contains u number of user groups from different regions $Ub_1, Ub_2, Ub_3 \ldots Ub_u$, and to process the user requests from these userbases the cloud has d number of DCs $Dc_1, Dc_2, Dc_3 \ldots Dc_d$, located in different regions of the world.

$$UB = \{Ub_1, Ub_2, Ub_3 \ldots Ub_u\}$$

$$DC = \{Dc_1, Dc_2, Dc_3 \ldots Dc_d\}$$

Every DC, denoted as Dc_d, consists of varying number of physical servers characterized by diverse or comparable configurations. Each datacenter DC_d combines a different number of physical servers with different or similar configurations $\{Pm_1, Pm_2, Pm_3, Pm_4, ...Pm_p\}$.

$$Dc_d \leftarrow \{Pm_1, Pm_2, Pm_3, Pm_4, ...Pm_p\}$$

Within the operational framework of this cloud environment, diverse userbases participate in the transmission of variable quantities of user tasks represented as $\{Ts_1, Ts_2, Ts_3, Ts_4, \ldots, Ts_n\}$ to the cloud infrastructure. This specification underscores the heterogeneous characteristics of the cloud ecosystem, wherein the amalgamation of multiple DCs and userbases fosters a dynamic and geographically diverse computing landscape.

$$Ub_u \leftarrow \{Ts_1, Ts_2, Ts_3, Ts_4, \ldots, Ts_n\}$$

Every physical machine engenders an abstraction comprising numerous VMs sequentially. The initial VM is denoted as VM_1, and the concluding VM is represented as VM_v.

$$Pm_n \leftarrow \{VM_1, VM_2, VM_3, VM_4, ...VM_v\}$$

Consider a cloud service with a userbase denoted by the variable u and a corresponding number of DCs denoted by d, serving as clients. The cloud service must employ an optimal DC selection policy to handle user requests efficiently. The objective is to identify the most suitable DC for the userbases at any given moment. Fig 1 illustrates the process of DC selection for the userbases.

Following the identification of a specific datacenter (Dc_d) for a designated userbase (Ub_u) during a specified timeframe, in accordance with user requisites, all tasks assigned to the user are subsequently routed to said DC for

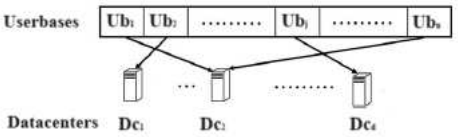

Fig. 1 DC selection

the particular duration. After that, the user tasks are processed in the virtual machine through effective load balancing mechanism. The userbase (Ub_u) initiates the dispatch of a quantity denoted as n tasks to the aforementioned DC, as depicted in Fig 2.

$$Dc_d \leftarrow Ub_u$$
$$Ub_u = \{Ts_1, Ts_2, Ts_3, Ts_4, \ldots, Ts_n\}$$

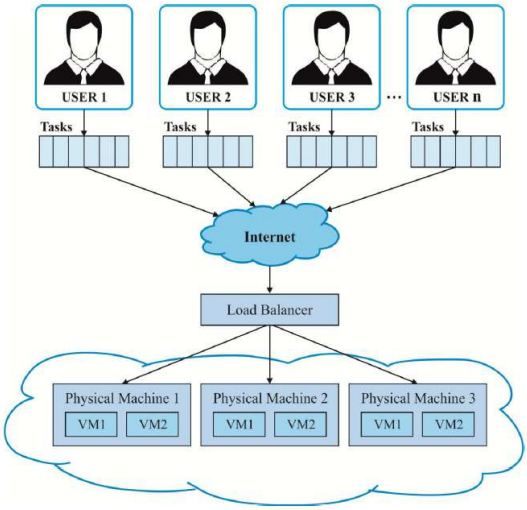

Fig. 2 Load Balancing

The DC denoted as Dc_d encompasses a set of p physical servers, represented as $\{Pm_1, Pm_2, Pm_3, Pm_4, \ldots, Pm_p\}$. To augment the performance metrics of the cloud system denoted as C, an abstraction layer has been introduced atop these physical servers in the form of VMs. Consequently, a total of v VMs have been instantiated, derived from the aforementioned p physical servers. The allocation of tasks to these VMs is executed through load balancing mechanisms, with the principal objective of preventing the overloading or

underutilization of VMs due to task imbalances.

$$Dc_d \leftarrow \{Pm_1, Pm_2, Pm_3, Pm_4, ...Pm_p\}$$
$$Pm_n \leftarrow \{VM_1, VM_2, VM_3, VM_4, ...VM_v\}$$
$$Dc_d \leftarrow \sum_{i \leftarrow 1}^{p} \sum_{j \leftarrow 1}^{v} \{VM_j\}_i$$

The load balancer distributes a variable number of tasks, denoted as Ts, across distinct virtual machines (VM). The allocation of tasks to VMs can be discerned by using equation 1.

$$\begin{aligned} VM_1 &= \{Ts_1, Ts_2, Ts_6, Ts_{10}, ...Ts_i\} \\ VM_2 &= \{Ts_3, Ts_5, Ts_{11}, Ts_{30}, ...Ts_j\} \\ &\vdots \\ VM_v &= \{Ts_{25}, Ts_{43}, Ts_{56}, Ts_{66}, ...Ts_k\} \end{aligned} \tag{1}$$

3.2 Objectives of Cloud

3.2.1 Userbase Response Time

The temporal interval, denoted as response time, delineates the duration elapsed from initiating a user's request to receiving the commensurate response from the cloud-based infrastructure. The response time, denoted as T_r, pertaining to a userbase can be formally characterized as:

$$T_r = f_t - a_t + t_d \tag{2}$$

Here, the variables a_t, f_t, and t_d denote the arrival time, finish time, and transmission delay, respectively.

Specifically, transmission delay denoted as t_d, encompasses the temporal interval during which data traverses the network between a source and destination within the cloud.

$$t_d = t_l + t_{tr} \tag{3}$$

In this context, t_l denotes the latency inherent in the network, while t_{tr} represents the time required for transferring data associated with a singular request, denoted as S_d, from the source to the destination.

$$t_{tr} = \frac{S_d}{bw_{peruser}} \tag{4}$$

$$bw_{peruser} = \frac{bw_t}{n_t} \tag{5}$$

In this context, bw_t signifies the aggregate bandwidth allocation accessible to the userbase, while n_t denotes the count of user requests presently undergoing

transmission. We would like to point out that the magnitudes of these parameters show constancy within the cloud environment and are precisely restricted by the individual cloud service providers.

$$\begin{aligned} T_r &= f_t - a_t + t_d \\ T_r &= f_t - a_t + t_l + t_{tr} \\ T_r &= f_t - a_t + t_l + \frac{S_d}{bw_{peruser}} \\ T_r &= f_t - a_t + t_l + \frac{S_d}{\frac{bw_t}{n_t}} \\ T_r &= f_t - a_t + t_l + \frac{S_d * n_t}{bw_t} \end{aligned} \tag{6}$$

Initially, the parameter T_r was introduced to denote the response time pertaining to an individual user base. Nevertheless, it is imperative to acknowledge the potential existence of a variable number, denoted as u, of distinct user bases within the cloud infrastructure. Cloud services' primary goal resides in minimizing the aggregated response time across all user bases encompassed by the cloud environment. Consequently, the articulation of the objective function F_1 is delineated as follows.

$$\begin{aligned} min(F_1) &= \sum_{i \leftarrow 1}^{u} T_r^i \\ min(F_1) &= \sum_{i \leftarrow 1}^{u} (f_t^i - a_t^i + t_l^i + \frac{S_d * n_t^i}{bw_t}) \\ min(F_1) &= \sum_{i \leftarrow 1}^{u} f_t^i - \sum_{i \leftarrow 1}^{u} a_t^i + \sum_{i \leftarrow 1}^{u} t_l^i + \frac{S_d * \sum_{i \leftarrow 1}^{u} n_t^i}{bw_t}) \end{aligned} \tag{7}$$

3.2.2 Datacenter Data Processing Time

Data processing time in a cloud environment indicates the requisite period for task execution, encompassing considerations of resource allocation dynamics, scheduling policies, and other pertinent factors within the emulated DC. The data processing time for an individual DC, denoted as T_d, may be elucidated as follows:

$$T_d = t_e + t_d + t_{qe} \tag{8}$$

In this context, the variable t_e represents the duration requisite for completing a specified computational task, delineating the temporal span necessary to execute said task. The variable t_d denotes network latency, elucidating the temporal interval during which data traverses distinct components, including transmissions between VMs or storage systems. Additionally, t_{qe} is employed

to characterize the queueing delay intrinsic to each task, signifying the temporal delay incurred due to a task awaiting processing in the queue before its initiation.

Within the DC environment, numerous tasks denoted as Ts_1, Ts_2, ..., Ts_r undergo processing. For any given point in time, the queue accommodates a total of r tasks, with each task necessitating a processing time denoted as t_p. Under these conditions, the variable t_{qe} is determined by the cumulative processing time of tasks present in the queue.

$$t_{qe} = \sum_{j \leftarrow 1}^{r} t_p^j \tag{9}$$

$$T_d = t_e + t_d + \sum_{j \leftarrow 1}^{r} t_p^j \tag{10}$$

The task execution is a direct metric for assessing the effectiveness of the scheduling framework. In virtualization technology, where concurrent task execution is facilitated across all VMs, determining the comprehensive execution time entails the identification of the VM characterized by the most protracted task execution duration [70]. The execution time, denoted as t_e, is explicated as follows:

$$t_e = \max_{m \leftarrow 1}^{M} Vm_t^m$$

$$Vm_t^m = \sum_{t \leftarrow 1}^{N} \frac{Ts_t}{Vm_m}$$

$$t_e = \max_{m \leftarrow 1}^{M} \sum_{t \leftarrow 1}^{N} \frac{Ts_t}{Vm_m} \tag{11}$$

Within the parameters of the provided assertion, the symbol Vm_t^m conveys the time required for the m-th VM to execute all the tasks allocated to m th VM. In this context, Ts_t represents the t-th task designated to a VM and Vm_m defines the computation power of the m-th VM, while M and N respectively denote the total count of VMs and tasks encompassed within the system.

$$T_d = \max_{m \leftarrow 1}^{M} \sum_{t \leftarrow 1}^{N} \frac{Ts_t}{Vm_m} + t_d + \sum_{j \leftarrow 1}^{r} t_p^j \tag{12}$$

The variable T_d is the duration required to process data within an individual DC. Notably, there exist d distinct DCs denoted as $Dc_1, Dc_2, \ldots, Dc_d$. Cloud services' fundamental objective is to reduce the cumulative data processing duration across all user-associated DCs within the encompassing cloud environment. Accordingly, the formulation of the objective function F_2 is expounded

as follows.

$$
\begin{aligned}
min(F_2) &= \sum_{n\leftarrow 1}^{d} T_d^n \\
min(F_2) &= \sum_{n\leftarrow 1}^{d} (\max_{m\leftarrow 1}^{M} \sum_{t\leftarrow 1}^{N} \frac{Ts_t}{Vm_m} + t_d + \sum_{j\leftarrow 1}^{r} t_p^j)^n \\
min(F_2) &= \sum_{n\leftarrow 1}^{d} (\max_{m\leftarrow 1}^{M} \sum_{t\leftarrow 1}^{N} \frac{Ts_t}{Vm_m})^n + \sum_{n\leftarrow 1}^{d} (t_d)^n \\
&\quad + \sum_{n\leftarrow 1}^{d} (\sum_{j\leftarrow 1}^{r} t_p^j))^n
\end{aligned} \tag{13}
$$

3.2.3 Cost of the Cloud

In practical contexts, cloud computing infrastructures implement a billing mechanism predicated upon the volume of user-generated traffic and bandwidth consumption. To emulate authentic operational conditions, we adopt Tencent Cloud's integrated pricing model, as delineated in [71], which encompasses both bandwidth and rate of flow considerations. The mathematical representation of the objective function for task execution cost is provided in equation 14.

$$
C_t = \sum_{i\leftarrow 1}^{M} \sum_{j\leftarrow 1}^{N} C_{vm}^i * t_{ij} + C_{vm_{bw}}^i \tag{14}
$$

In the context of the presented equation, the variable C_{vm}^i indicates the configuration cost associated with the i-th VM. This cost parameter signifies the financial obligation imposed upon the user in utilizing the specified VM. Concurrently, the variable t_{ij} indicates the assignment status of the i-th VM to the j-th task, where 'N' represents the total number of tasks under consideration. $C_{vm_{bw}}^i$ is employed to characterize the bandwidth cost incurred at the i-th VM, encapsulating the expenses associated with the requisite network bandwidth for optimal functioning.

$$
t_{ij} = \begin{cases} 1 & \text{if } VM_j \text{ is used in } task_i \\ 0 & \text{otherwise} \end{cases} \tag{15}
$$

The primary aim of cloud service providers is to mitigate overall costs for users engaged in using cloud services. If the user needs D number of DCs to execute

their tasks, the overall objective function for cost F_3 will be,

$$\begin{aligned} min(F_3) &= \sum_{d\leftarrow 1}^{D} C_t \\ min(F_3) &= \sum_{d\leftarrow 1}^{D}\sum_{i\leftarrow 1}^{M}\sum_{j\leftarrow 1}^{N} C_{vm}^{i} * t_{ij} + C_{vm_{bw}}^{i} \end{aligned} \tag{16}$$

3.3 DEThresh Objective Functions

From the above function, the objective function for our DEThresh algorithm is defined as shown in equation 17

$$\begin{cases} min(F_1) = \sum_{i\leftarrow 1}^{u} f_t^i - \sum_{i\leftarrow 1}^{u} a_t^i + \sum_{i\leftarrow 1}^{u} t_l^i + \frac{S_d * u * \sum_{i\leftarrow 1}^{u} n_t^i}{u * bw_t}) \\ min(F_2) = \sum_{n\leftarrow 1}^{d} (\max_{m\leftarrow 1}^{M} \sum_{t\leftarrow 1}^{N} \frac{Ts_t}{Vm_m})^n + \sum_{n\leftarrow 1}^{d} (t_d)^n \\ \qquad\qquad + \sum_{n\leftarrow 1}^{d} (\sum_{j\leftarrow 1}^{r} t_p^j))^n \\ min(F_3) = \sum_{d\leftarrow 1}^{D} \sum_{i\leftarrow 1}^{M} \sum_{j\leftarrow 1}^{N} C_{vm}^{i} * t_{ij} + C_{vm_{bw}}^{i} \end{cases} \tag{17}$$

Here F_1 corresponds to the userbase response time objective function, whereas F_2 corresponds to the datacenter data processing time objective function and F_3 defines the overall cost objective function of the cloud.

4 Proposed DEThresh Algorithm

Our proposed DEThresh algorithm incorporates a differential evolution-based DC selection policy and threshold-based load balancing policy to achieve better performance for the cloud services. The DE-based DC selection will help to find the most suitable DC for the user at a particular moment. After the DC selection, we will use the threshold-based load balancing policy to distribute the workload among the VMs. The goal of the load balancing policy is to spread the workload to ensure maximum resource utilization of the VMs.

4.1 Proposed DE-based DC selection

DE is a highly effective stochastic optimization algorithm to optimize real parameters. This algorithm shares computational procedures inspired by Darwin's evolution concept [72–74], diverging from the genetic algorithm by executing its operations in reverse order. DE variants introduce perturbations to the population members of the present generation by randomly selecting and comparing small subsets of individuals. The algorithmic instantiation of DE-based DC selection delineated in algorithm 1, relies on the mutation strategy to induce alterations in the search space. This strategy proves instrumental in facilitating the exploration of optimal solutions within a global search space, as opposed to a confined local search space. The termination of DE-based DC selection yields optimal DC sequences for userbases, thereby contributing to

the enhancement of response times and data processing efficiency within the cloud environment.

Algorithm 1: Proposed DE based DC selection

Input : 1. Userbase (UB_1, UB_2, UB_3, UB_4,.... UB_m)
2. Datacenter (DC_1, DC_2, DC_3, DC_4,..... DC_n)

Output: Most suitable DC sequence for the userbases

UB_1	UB_2	UB_3	UB_4	....	UB_m
DC_2	DC_1	DC_n	DC_5	....	DC_2

$population \leftarrow InitializePopulation()$
while *fitness not achieved* **do**
  $newPopulation \leftarrow []$ **for** $i \leftarrow$ *1 to population.size()* **do**
    $v_i \leftarrow differentialMutation(population)$
    $newPopulation_i \leftarrow recombination(v_i, x_i)$
  end
  $population \leftarrow Selection(newPopulation, population)$
end
$dcSelection(population)$

4.1.1 Intitalization

Creating the population in the Differential Evolution (DE) algorithm is critical, representing the aggregate of chromosomes participating in the evolutionary process. Chromosomes, synonymous with individuals, play a pivotal role in shaping the solution structure within DE algorithms. Throughout the algorithmic evolution, these chromosomes transform, influencing the algorithm towards optimal solutions. The structural composition of chromosomes in the DE algorithm is detailed in Table 1. Amongst the population, one individual ultimately emerges as the algorithm's solution. Establishing diversity within the population is essential and is achieved through a random generation process. The introduction of randomness facilitates exploration across diverse search spaces within cloud computing. Furthermore, chromosomes generated from the CDC policy are incorporated into the initial population to enhance population diversity. This strategic integration is based on the assumption that if the nearest DCs exhibit fitness during evolution, they will persist and progress toward optimal solutions. Conversely, if they lack fitness, they undergo transitions within the evolutionary algorithm, generating chromosomes that converge toward optimal solutions. This deliberate inclusion augments the algorithm's efficacy in exploring and identifying optimal outcomes.

Table 1 Chromosome Structure

UB_1	UB_2	UB_3	UB_4	....	....	UB_m
DC_2	DC_1	DC_n	DC_5	....	.....	DC_2

4.1.2 Differential Mutation

Mutation denotes an abrupt modification in the genetic attributes of a chromosome. In differential evolution computing, this mutation can be construed as an unforeseen shift or perturbation [72]. The most rudimentary manifestation of Differential Evolution (DE) mutation involves the generation of a mutant vector for each mutation event. The mutation process in the proposed DE algorithm is elucidated in algorithm 2. Three distinct random base vectors are selected from the population to achieve this. Subsequently, the discrepancy between any two of these three vectors is adjusted by a scalar and random value denoted as F. The value of F can be from 0 to 1. The meticulous selection process employed for the base vector during mutation significantly contributes to the overall efficacy of the DE algorithm. In the framework of our proposed DE algorithm, a strategic approach is used where the preeminent individual from the population is bequeathed during the mutation step. Specifically, in algorithm 2, Cr_1 represents the optimal individual within the population, while Cr_2 and Cr_3 signify any other randomly chosen vectors within the solution space.

Algorithm 2: Differential Mutation

Input : *population*
Output: $mutantVector(v_i)$
$sortbyFitness(population)$
$Cr_1 \leftarrow population.bestFit()$
$Cr_2 \leftarrow population.random((Cr) => \{Cr! = Cr_1\})$
$Cr_3 \leftarrow population.random((Cr) => \{Cr! = Cr_1 \ and \ Cr! = Cr_2\})$
$F \leftarrow random(0, 1)$
$v_i \leftarrow Cr_1 + F(Cr_2 - Cr_3)$

4.1.3 Recombination

The augmentation of population diversity is facilitated by implementing the recombination operation after generating the mutant vector through mutation [72]. Algorithm 3 delineates the procedural framework of the recombination operation within our proposed DE. The algorithm incorporates a parameter denoted as the recombination rate *recombinationRate*, which governs the execution of the recombination procedure. Specifically, a random number c within the range of 0 to 1 is generated; if c surpasses the designated recombination rate, the original vector x_i is selected in the recombination event. Conversely,

if the generated random number falls below the *recombinationRate*, the recombination will choose the mutant vector v_i.

Algorithm 3: Recombination

Input : v_i, x_i
Output: $triaVector(u_i)$
$c \leftarrow random(0, 1)$
if $c \leq recombinationRate$ **then**
    $u_i \leftarrow v_i$
else
    $u_i \leftarrow x_i$
end

4.1.4 Selection

The proposed selection algorithm denoted as algorithm 4, adopts a greedy selection strategy wherein the trial vector takes precedence over the parameter vector if it yields a superior fitness function value. At generation i, this selection process determines the survival of the trial or target vector for the subsequent generation. The fitness computation is conducted through algorithm 5. In the selection algorithm, trial vectors possessing equal or lower values than their corresponding target vectors replace the latter in the succeeding generation, as delineated in algorithm 4. Conversely, if a trial vector yields a higher value, it persists within the population. This methodology minimizes the objective function, thereby augmenting or sustaining the population's fitness [75]. Notably, when both target and trial vectors yield identical objective function values, the trial vector supersedes the target vector. This approach facilitates the traversal of flat fitness landscapes across generations, thereby fostering continual advancements in optimizing the objective function.

Algorithm 4: Selection Algorithm

Input : $newPopulation, population$
Output: $population$
for $i \leftarrow 1$ *to* $population.size()$ **do**
    if $calculateFitness(newPopulation.at(i) \geq calculateFitness(population.at(i))$ **then**
        $population.replace(i, newPopulation.at(i))$
    end
end

Algorithm 5: Fitness Calculation

Input : Chromosome UB_1 UB_2 UB_3 UB_4 .... UB_m

DC_2	DC_1	DC_n	DC_5	....	DC_2

Output: Fitness Value of the chromosome

$dcSet \leftarrow \{\}$
$D_t \leftarrow 0$
$R_t \leftarrow 0$
$C_t \leftarrow 0$ **for** $i \leftarrow 1$ *to chromosome.length* **do**
  if *dcSet not contains chromosome.at*(i)*.DC* **then**
    $dc \leftarrow chromosome.at(i).DC$
    $D_t \leftarrow D_t + calculateDCprocessingTime(dc)$
    $D = C_t \leftarrow C_t + calculateCost(dc)$
    $dcSet.add(dc)$
  end
  $ub \leftarrow chromosome.at(i).UB$
  $R_t \leftarrow R_t + calculateUBRresponseTime(ub)$
end
return R_t, D_t, C_t

4.1.5 Termination

The termination criterion governs the cessation of the algorithm. Our DE algorithm experiences each generation's mutation, recombination, and selection operations. Our proposed DE algorithm's cessation is terminated upon identifying convergence, as determined by the fitness function. After the proposed DE algorithm is terminated, algorithm 6 is implemented to ascertain the optimal DC selection. The population is arranged in descending order based on their respective fitness values. Subsequently, a comparative analysis is conducted between the fitness of the top-performing chromosome within the population and the fitness associated with the preceding DC selection employed in the service broker policy. If the most highly ranked individual in the population demonstrated superior fitness, it was incorporated into the service broker policy. Conversely, if the existing DC selection exhibited superior fitness, it is retained for further use within the DC selection process.

4.2 Threshold Based Load Balancing Policy

The proposed load balancing mechanism, outlined in algorithm 1, operates by receiving as input a set of tasks $\{T_1, T_2, T_3, T_4, \ldots, T_m\}$ from the userbase, and a set of VMs $\{VM_1, VM_2, VM_3, VM_4, \ldots, VM_n\}$ from the selected DC, as determined by algorithm 7.VMs devoid of assigned tasks are identified as underloaded VMs. Upon the arrival of a task in the DC, a selection is made from the underloaded VMs. Subsequently, an assessment is conducted to ascertain whether the load on the chosen VM meets or exceeds the overload threshold. If the selected VM is overloaded or otherwise, algorithm 9 is

Algorithm 6: DC selection strategy of our proposed DE

Input : 1. *population*
2. *prevDCselection*
Output: *bestDCselection*
sortByFitness(*population*)
populationBest ← *population.at*(0)
if *populationBest.fitness* ≥ *prevDCselection* **then**
bestDCselection ← *populationBest*
else
bestDCselection ← *prevDCselection*
end

employed. If the VM is underloaded, the task is allocated to it. However, if the VM is overloaded, it is removed from the underloaded VM list and added to the overloaded VM list. A subsequent search in the underloaded VM list is conducted to identify an alternative underloaded VM for task assignment. If no underloaded VMs exist, the task is assigned to an overloaded VM with a lesser task count.

Algorithm 7: Proposed Threshold based loadbalancing

Input : 1. Tasks (T_1, T_2, T_3, T_4,.... T_m)
2. Virtual Machines (VM_1, VM_2, VM_3, VM_4,..... VM_n)
Output: Continuously allocates the tasks to the VMs based on VM load
$underLoadVMs \leftarrow [VM_1, VM_2, VM_3, VM_4.....VM_n]$
$overloadVMs \leftarrow []$
while *tasks is incoming* **do**
$T \leftarrow tasks.pop()$
while *underloadVMs is not empty* **do**
$allocatedVM \leftarrow undeloadVMs.randomVM()$
if $checkOverload(allocatedVM)$ **then**
$allocateTasktoVM(t, allocatedVM)$
$allocatedVM.taskCount \leftarrow allocatedVM.taskCount + 1$
else
$overloadVMs.push(allocatedVM)$
end
end
end

Upon task completion, algorithm 8 is invoked to decrement the task count. For VMs identified as overloaded, a check is initiated using algorithm 10 to determine if the VM has transitioned to an underloaded state. If the VM has

become underloaded, it is removed from the overloaded VMs list and added to the underloaded VMs list. The existence of a gap between the underloaded and overload thresholds serves to optimize resource utilization within the VMs, mitigating the potential overcrowding of tasks. The primary objective of

Algorithm 8: Task Completion Function

Input : 1. Completed Task ($completedTask$)
2. Underloaded VM List $overloadVMs$
3. Overloaded VM List $overloadVMs$

Output: Updates the VM load after task execution

$allocatedVM \leftarrow completedTask.VM$
$allocatedVM.taskCount \leftarrow allocatedVM.taskCount - 1$
if *overloadVMs contains allocatedVM &* $checkUnderLoadVM(allocatedVM)$ **then**
  $overloadVMs.remove(allocatedVM)$
  $underloadVMs.add(allocatedVM)$
else
  $underloadVMs.update(allocatedVM)$
end

our threshold algorithms is to distribute the computational workload among all VMs within the DC equitably. Frequently observed is when certain VMs exhibit an undue concentration of tasks, leading to overcrowding, while concurrently, other machines remain idle. Consequently, the aggregate processing time within the DC is augmented despite the availability of ample resources for task execution. To rectify this imbalance and optimize load distribution across nodes, we have endeavored to alleviate the situation by categorizing VMs into two distinct classifications.

4.2.1 Overloaded VMs

During the task assignment process, algorithm 9 will be employed to ascertain whether concurrent tasks presently burden the VM designated for task allocation. This evaluation entails verifying whether the available computational capacity of the VM exceeds 25% of its initial computational power. Failure to meet this criterion will result in categorizing the VM as overloaded and excluding it from the list of underloaded VMs. So the threshold for determining overload VM can be defined as

$$VM^{i}_{overloadThreshold} \leftarrow 0.25 * VM^{i}_{computationPower}$$

4.2.2 Underloaded VMs

During task completion, it is essential to assess the potential overloading of VMs and see if a transition from an overloaded to an underloaded state has occurred. When the VM is identified as underloaded, a reduction in the task allocation for the VM is warranted. The underload status is evaluated using algorithm 10. Specifically, if the available computational capacity of the VM exceeds 75% of its initial computational power, it is designated as an underloaded VM. Consequently, such identified underloaded VMs are removed from the overload VM list and integrated into the underloaded VM list. So, the threshold for determining underload VM can be defined as

$$VM^{i}_{underloadThreshold} \leftarrow 0.75 * VM^{i}_{computationPower}$$

Algorithm 9: Check Underload VM

```
Input  : VM
Output: true or false
if VM.availablecomputationPower() ≥ 0.25 *
  VM.totalcomputationPower() then
    return true
else
    return false
end
```

Algorithm 10: Check Underload VM

```
Input  : VM
Output: true or false
if VM.availablecomputationPower() ≥ 0.75 *
  VM.totalcomputationPower() then
    return true
else
    return false
end
```

While employing a singular threshold value suffices for identifying underloaded and overloaded VMs, a rationale exists for adopting two distinct values. When VMs surpass the overload threshold, task allocation ceases on that particular VM, prompting the search for an alternative underloaded VM. However, in the case of a single threshold value, the overloaded VM, upon completing a task, would promptly transition into an underloaded state. Introducing dual

threshold values ensures the overloaded node does not immediately convert into an underloaded VM. Instead, it continues to execute the assigned tasks until reaching the underload threshold, subsequently assuming the status of an underloaded VM. This approach mitigates the swift alternation between underloaded and overloaded states for resources on the overloaded VM, introducing a gradual process wherein the completion of tasks precedes the transition to an underloaded state.

5 Result Analysis

5.1 Experiment Setup

Cloud computing represents a highly intricate process influenced by variables beyond direct control, such as network latency and disparate server workloads. Consequently, evaluating the performance of internet-based applications on authentic cloud platforms proves exceptionally challenging [76–78]. To address this challenge, simulation-based experiments become imperative, offering a virtual and cost-free avenue within stable and manageable environments [79–81]. Various cloud simulators are available for research purposes, among which the CloudAnalyst Platform [82] will be employed in our investigation to assess the efficacy of our proposed algorithm. Distinguished by its graphical user interface (GUI) and an extension of CloudSim, CloudAnalyst exhibits user-friendly attributes and adaptability to real-world scenarios. Our study will subject the algorithm to diverse real-world scenarios, facilitating a comprehensive analysis of its performance.

To assess performance, comparable cloud scenarios employed in prior research endeavors were utilized [83–87]. The userbase configuration (UBC_1, UBC_2, UBC_3)[83, 84] and datacenter (DC) [85–87] configuration are explained based on the methodologies delineated in those studies. In CloudAnalyst, the world is divided into six distinct geographical regions denoted as R_0, R_1, R_2, R_3, R_4, and R_5. The DCs may be situated within the same geographical region or across disparate regions. , The performance of DEThresh will be benchmarked against two prevalent DC policies, namely the Closest Datacenter Policy (CDC) and Optimized Response Time Policy (ORT), in conjunction with two widely adopted load balancing policies: Round Robin(RR) and Throttled Load Balancing (TLB) Policy [88].

5.2 Performance Varying the Userbase

The experimental results presented in Tables 2 and 3 provide a comprehensive analysis of the overall response time and data processing time in a cloud computing environment characterized by a fixed number of 4 DCs and varying userbase sizes. The evaluation encompasses multiple algorithms, shedding light on their respective performances. The cloud scenarios used in our experiment

are shown below.

$$CS_1 = \begin{cases} UB \leftarrow \{UBC_1\} \\ DC \leftarrow \{DC_{1,R_0}, DC_{2,R_0}, DC_{3,R_2}, DC_{4,R_3}\} \end{cases}$$

$$CS_2 = \begin{cases} UB \leftarrow \{UBC_2\} \\ DC \leftarrow \{DC_{1,R_0}, DC_{2,R_0}, DC_{3,R_2}, DC_{4,R_3}\} \end{cases}$$

$$CS_3 = \begin{cases} UB \leftarrow \{UBC_3\} \\ DC \leftarrow \{DC_{1,R_0}, DC_{2,R_0}, DC_{3,R_2}, DC_{4,R_3}\} \end{cases}$$

$$CS_4 = \begin{cases} UB \leftarrow \{UBC_1\} \\ DC \leftarrow \{DC_{1,R_0}, DC_{2,R_2}, DC_{3,R_3}, DC_{4,R_4}\} \end{cases}$$

$$CS_5 = \begin{cases} UB \leftarrow \{UBC_2\} \\ DC \leftarrow \{DC_{1,R_0}, DC_{2,R_2}, DC_{3,R_3}, DC_{4,R_4}\} \end{cases}$$

Table 2 and 3 reveal that the CDC_RR algorithm exhibits suboptimal performance compared to other algorithms. This underperformance is attributed to the CDC policies, which tend to favor the closest DC, leading to an uneven distribution of tasks and overcrowding. Simultaneously, while avoiding DC overcrowding, RR policy needs to improve to ensure optimal resource utilization for VMs. The integration of TLB proves instrumental in enhancing CDC's performance significantly. Furthermore, a notable improvement is observed when transitioning from CDC policies to the ORT DC selection policy, which dynamically selects the DC with the best response time. Integrating TLB load balancing with ORT further elevates the cloud's performance, as evidenced by improvements in both response time and data processing time.

Table 2 Overall Response Time (Milliseconds) varying the userbase

Cloud Scenerio	CDC_RR	CDC_TLB	ORT_RR	ORT_TLB	DEThresh
CS_1	2752.02	1477.62	1030.32	605.9	425.51
CS_2	4943.09	2542.61	2831.28	1402.17	736.49
CS_3	2431.85	1315.16	916.96	601.2	477.76
CS_4	2111.43	1142.99	778.5	490.74	353.73
CS_5	3901.54	2029.62	1953.08	1004.08	560.48

Table 3 Overall Data Processing Time (Milliseconds) varying the userbase

Cloud Scenerio	CDC_RR	CDC_TLB	ORT_RR	ORT_TLB	DEThresh
CS_1	2343.26	1227.52	776.68	406.3	198.62
CS_2	4205.65	2150.22	2243.68	1055.99	408.59
CS_3	2111.52	1104.54	694.65	410.81	261.5
CS_4	1829.68	948.28	571.11	309.26	142.95
CS_5	3514.26	1801.77	1615.17	748.27	262.97

In Fig 3 and Fig 4, visual representations provide a precise comparative analysis of response and data processing time across various algorithms. The figures illustrate the superiority of the proposed DEThresh algorithm, surpassing all other algorithms in terms of response time for userbases and data processing time for DCs. This superior performance aligns with the algorithm's ability to ensure proper resource utilization and user satisfaction, meeting QoS and SLA requirements.

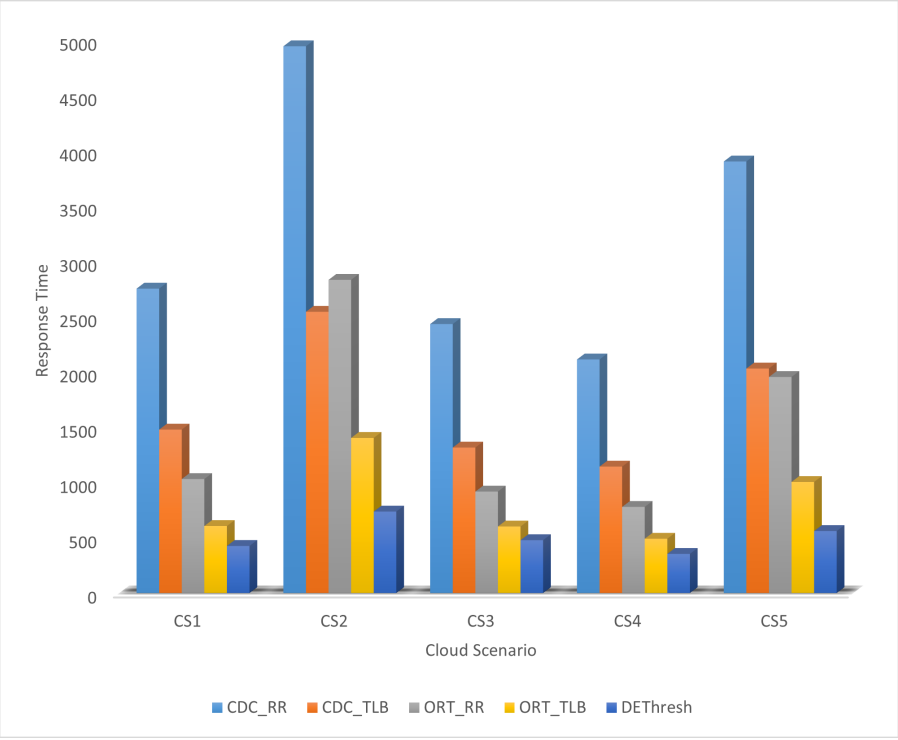

Fig. 3 Comparison of overall response time (Milliseconds) of different algorithm varying userbase

The analysis of response time and data processing time is critical in evaluating the efficacy of our proposed DEThresh algorithm compared to existing algorithms, namely CDC_RR, CDC_TLB, ORT_RR, and ORT_TLB. To provide a comprehensive understanding of the performance metrics, we have calculated the percentage of improvement achieved by DEThresh relative to each individual algorithm in Table 4 and Table 5.

Table 4 details the improvement in response time. DEThresh exhibits a notable enhancement, reaching a maximum improvement of 85.64% compared to CDC_RR. On average, DEThresh demonstrates an impressive 84% improvement in response time over CDC_RR. Comparisons with other algorithms reveal a 70% improvement over CDC_TLB, a 61.3% improvement

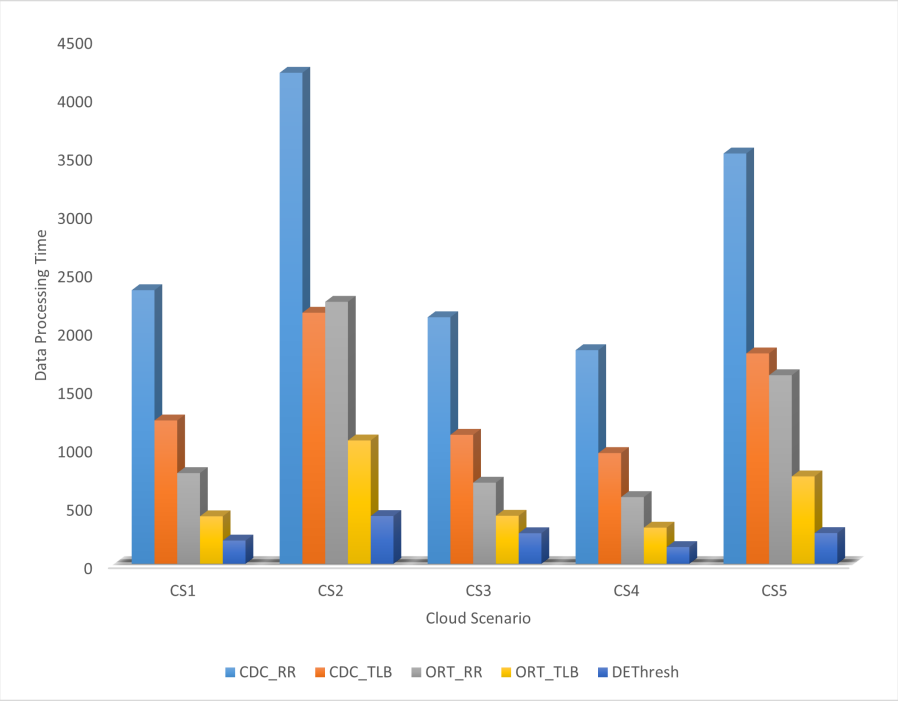

Fig. 4 Comparison of overall data processing time (Milliseconds) of different algorithms varying userbase

over ORT_RR, and a minimum but still substantial 34% improvement over ORT_TLB. This comprehensive analysis establishes that DEThresh consistently outperforms other algorithms in diverse cloud scenarios, aligning with the primary objective of minimizing response time, as specified in equation 7. The visual representation in Fig 5 further supports these findings, reinforcing the claim of DEThresh effectively minimizing response time.

Table 4 Response Time Improvement of DEThresh compared to other algroithms

Algorithm	CS_1	CS_2	CS_3	CS_4	CS_5	Average Improvement
CDC_RR	84.53%	85.10%	80.35%	83.25%	85.64%	84.00%
CDC_TLB	71.20%	71.04%	63.67%	69.05%	72.40%	70.00%
ORT_RR	58.70%	74.00%	47.90%	54.56%	71.30%	61.30%
ORT_TLB	29.77%	47.48%	20.53%	27.92%	44.18%	34.00%

Moving to Table 5, which focuses on data processing time, DEThresh again demonstrates significant improvements. The maximum improvement in data processing time is an impressive 91% compared to CDC_RR, while the minimum improvement, albeit substantial, is 53.5% compared to ORT_TLB. Comparisons with CDC_TLB and ORT_RR reveal improvements of 82.3%

and 74.45%, respectively. Overall, DEThresh consistently achieves a minimum of 50% improvement in minimizing data processing time compared to other algorithms. This aligns seamlessly with the predefined objective function of minimizing data processing time, as expressed in equation 13.The graphical representation in Fig 6 visually reinforces the data processing time improvements achieved by DEThresh, providing a clear comparative illustration of the algorithm's effectiveness.

Table 5 Data Processing Time Improvement of DEThresh compared to other algroithms

Algorithm	CS_1	CS_2	CS_3	CS_4	CS_5	Average Improvement
CDC_RR	91.53%	90.30%	87.62%	92.20%	92.52%	91.00%
CDC_TLB	83.82%	81.00%	76.33%	84.93%	85.41%	82.30%
ORT_RR	74.43%	81.80%	62.36%	74.97%	83.72%	75.45%
ORT_TLB	51.12%	61.31%	36.35%	53.78%	64.86%	53.50%

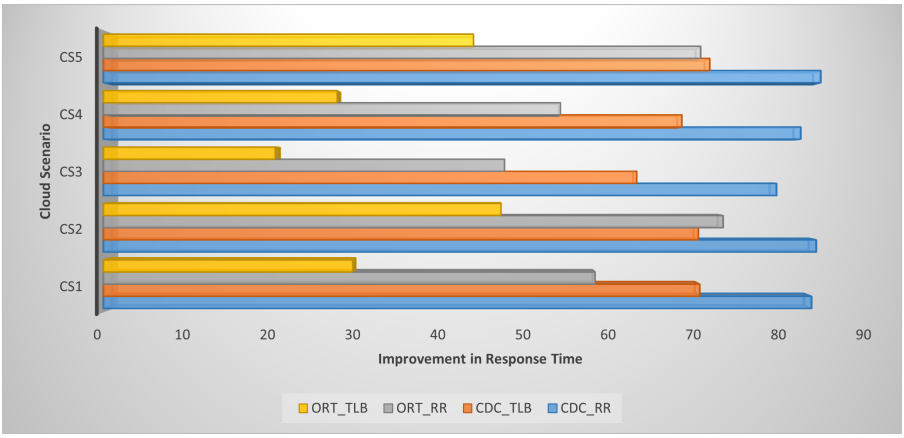

Fig. 5 Comparison of response time (Milliseconds) improvement of different algorithms with DEThresh

In conclusion, the experimental findings underscore the critical role of algorithmic choices and DC selection policies in shaping the performance of cloud computing systems. Our analysis indicates that DEThresh surpasses its counterparts in response time and also excels in minimizing data processing time, showcasing its overall superiority in diverse cloud computing scenarios. Our proposed DEThresh algorithm emerges as a promising solution and contributes to enhanced resource utilization and user satisfaction in compliance with stringent QoS and SLA standards.

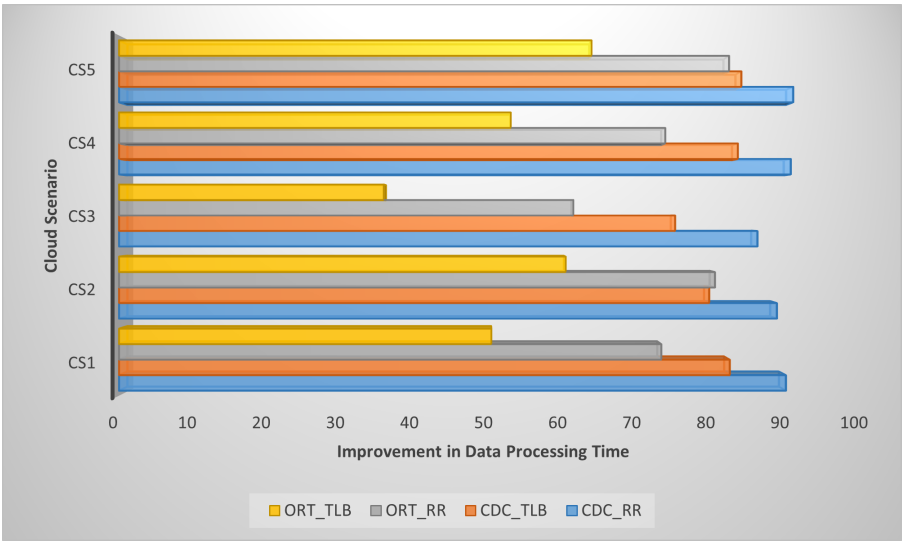

Fig. 6 Comparison of data processing time improvement of different algorithms with DEThresh

5.3 Performance Varying the number of datacenters

Table 4, Table 5, Fig 5, and Fig 6 presented in the preceding section provide evidence suggesting that the improvement of DEThresh is minimum compared to other algorithms considering response time and data processing time while comparing with ORT_TLB for DC selection and load balancing. Due to this rationale, our emphasis was directed toward assessing the efficacy of two different methodologies, DEThresh and ORT_TLB, across diverse scenarios involving varying quantities of DCs. The userbase and DC configuration for this experiment are shown below:

$$\begin{cases} UB \leftarrow \{UBC_1\} \\ DC \leftarrow \{DC_{1,R_0}, DC_{2,R_1}, DC_{3,R_2}, DC_{4,R_3}, DC_{5,R_4}\} \end{cases}$$

The results, outlined comprehensively in Table 6 and Table 7, elucidate the response and data processing times of DEThresh compared to ORT_TLB. Specifically, for 3, 4, and 5 DCs, the observed improvements in response time are 33.3%, 28%, and 39.56%, respectively, in contrast to the performance of ORT_TLB. Similarly, with regard to data processing time, the enhancements are 54.5%, 53.78%, and 58.38% for 3, 4, and 5 DCs, respectively. This consistent pattern indicates that an augmented number of DCs positively correlates with improved performance. In contrast, reducing the number of DCs leads to a decline in overall system performance.

For a more illustrative representation of the comparative analysis, Fig 7 and Fig 8 were generated to visually depict the performance of DEThresh and ORT_TLB under different numbers of DCs. DEThresh, consistently positioned

Table 6 Response Time (Milliseconds) Varying the datacenters

Algorithm	3DC	4DC	5DC
ORT_TLB	1058.83	490.74	465.28
DEThresh	706.41	353.73	281.2

Table 7 Data Processing Time (Milliseconds) Varying the datacenters

Algorithm	3DC	4DC	5DC
ORT_TLB	765.11	309.26	303.05
DEThresh	348.1	142.95	126.12

itself beneath ORT_TLB, for both response and data processing times. This consistent trend across various DC scenarios underscores the inherent outperformance of DEThresh in terms of DC selection and load balancing compared to ORT_TLB.

Crucially, the key insight gleaned from Fig 7 and Fig 8 is that DEThresh consistently outperforms ORT_TLB, irrespective of the number of DCs in consideration. From 6 and Table 7 it is noted that DEThresh yields comparable or superior outcomes over ORT_TLB even when deployed with fewer DCs. This implies a robust adaptability and efficiency of DEThresh in diverse operational contexts.

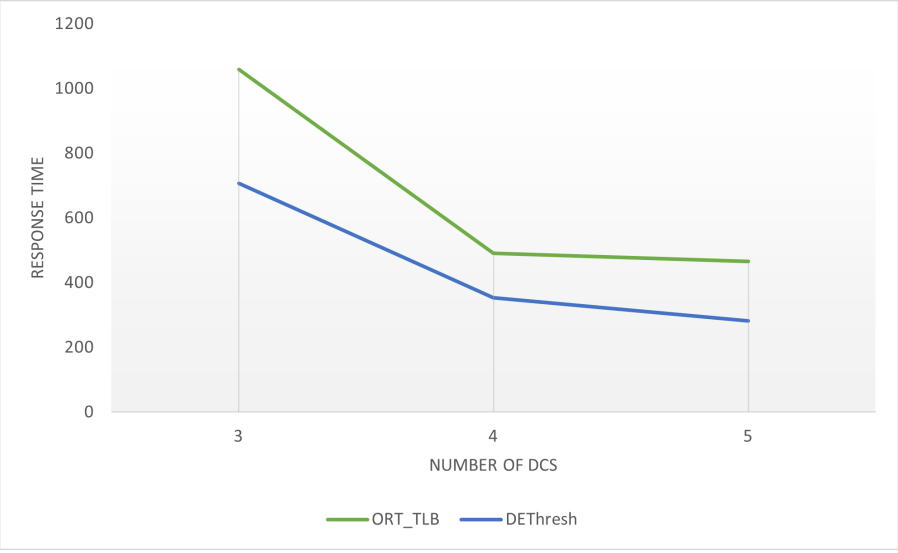

Fig. 7 Comparison of response time of the algorithms by varying the number of DCs

In conclusion, the comprehensive analysis presented in Table 6 and Table 7, along with the visual representation in Fig 7 and Fig 8, collectively affirms that DEThresh surpasses ORT_TLB regarding both response time and data

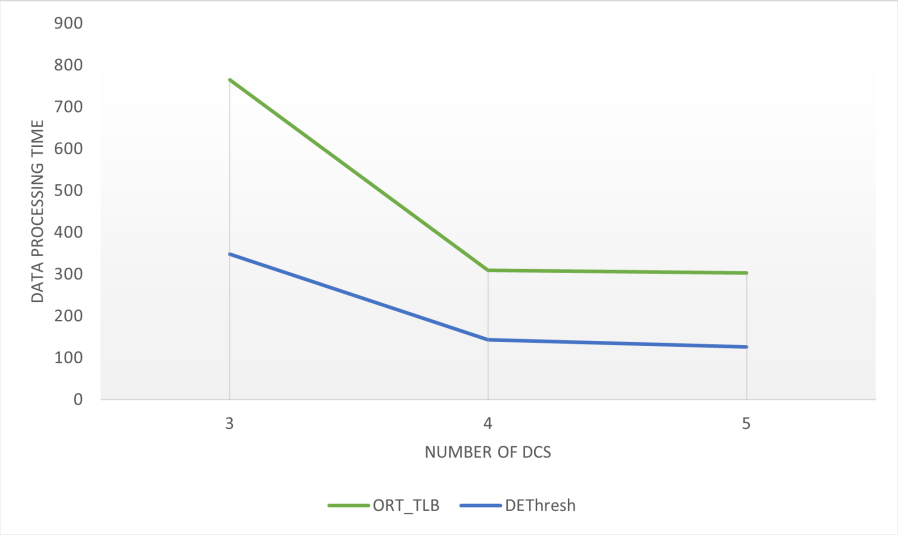

Fig. 8 Comparison of data processing time of the algorithms by varying the number of DCs

processing times. In this manner, our DEThresh successfully fulfills its primary goal of minimizing response time and data processing times, as depicted in equations 7 and 13, respectively. This achievement is demonstrated across different numbers of DCs. Importantly, this superiority persists across varying numbers of DCs, suggesting the versatility and reliability of DEThresh in dynamic computing environments. Furthermore, the potential for DEThresh to achieve comparable or superior outcomes with fewer DCs indicates a potential for resource optimization, positioning DEThresh as a promising alternative to existing DC selection and load balancing policies regarding resource utilization.

5.4 Cost Analysis

The expansion of cloud services through the establishment of new DCs contributes to increased costs for cloud providers. This escalation of the expenses is subsequently transferred to users, aiming to enhance profitability. Table 8 outlines the comprehensive cost analysis associated with varying numbers of DCs, as simulated through CloudAnalyst. The table illustrates a direct proportionality between the total cost and the number of DCs employed. This relationship is visually depicted in Fig 9, affirming a linear correlation between total cost and the quantity of DCs. It is important to note that the expenses incurred in constructing and maintaining these additional DCs should be factored into this comparative analysis.

Upon closer examination of Fig 7 and Fig 8, the superior performance of our DEThresh algorithm becomes evident. In contrast to the ORT_TLB counterpart, DEThresh exhibits enhanced response time and data processing efficiency, even when utilizing fewer DCs. This observation underscores

Table 8 Cost of the datacenters

Number of DCs	Cost as Per CloudAnalyst Metrics
3	2435.85
4	2495.85
5	2555.85

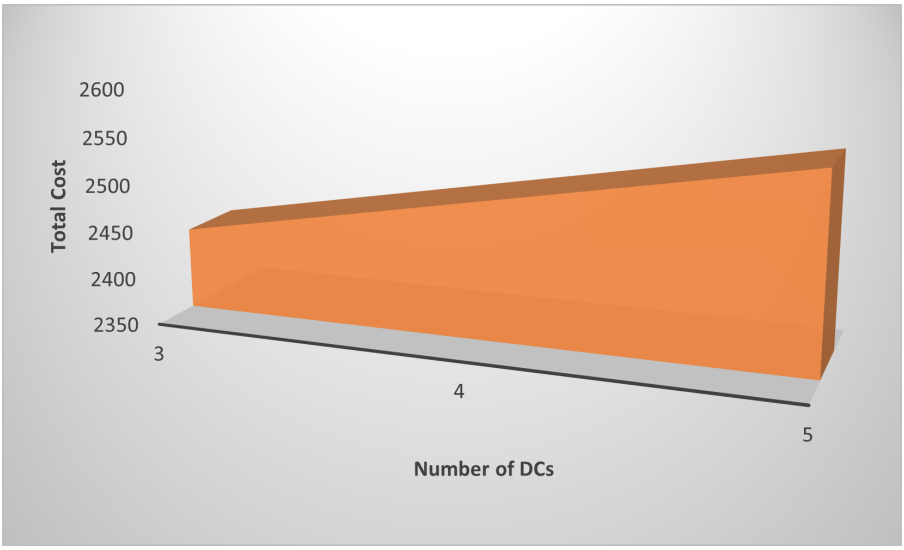

Fig. 9 Comparison of cost by varying the number of DCs

the optimization of available resources by the DEThresh algorithm, obviating the need to construct and maintain additional DCs. Consequently, the DEThresh algorithm emerges as a reasonable and cost-effective solution for cloud providers, concurrently reducing the financial burden on cloud users.

In summary, the proposed DEThresh algorithm attains a dual objective of minimizing the overall cost of cloud services and obviating the necessity for constructing and sustaining new DCs. By achieving these goals, the DEThresh algorithm aligns with the fundamental objective function of cost minimization, as articulated in equation 16.

5.5 Performance of Individual Datacenter

The investigation undertaken pertains to the evaluation of the performance of individual DCs with a specific focus on the DC request servicing time. The request servicing time denotes the duration required for processing user requests within each DC. Table 9 presents the request servicing times for each DC, taking into account CDC_TLB, ORT_TLB, and DEThresh. The graphical representations in Fig 10, Fig 11, Fig 12, and Fig 13 illustrate the performance trends observed in each DC.

Table 9 Individual datacenter performance (Milliseconds)

	DC_1			DC_2			DC_3			DC_4		
Cloud Scenario	CDC_TLB	ORT_TLB	DEThresh	CDC_TLB	ORT_TLB	DEThresh	CDC_TLB	ORT_TLB	DEThresh	CDC_TLB	ORT_TLB	DEThresh
CS_1	1787.7	939.93	589.31	1899.08	1043.49	625.03	11.25	8.7	26.19	103.15	125.92	257.1
CS_2	3452.87	2460.78	1481.79	2302.74	1746.86	611.6	11.26	8.13	21.24	30.7	29.8	399.64
CS_3	1524.77	804.26	575.75	1860.79	1229.45	757.53	10.63	8.24	34	85.64	108.91	232.13
CS_4	1229.57	2460.78	627.48	1704.77	1746.86	507.78	9.73	8.13	21.9	71.71	29.8	173.32
CS_5	2517.66	1638.92	970.45	2182.42	1226.17	738.71	9.73	7.4	36.54	32.15	30.61	33.27

Upon analyzing the Fig 10 and Fig 11, it becomes evident, particularly in the cases of DC1 and DC2, that DEThresh outperforms CDC_TLB and ORT_TLB in terms of request servicing time. However, Fig 12 and Fig 13 reveal that, in the case of DC3 and DC4, DEThresh's performance is not superior to the other algorithms. These figures imply that our proposed DEThresh algorithm utilizes DC3 and DC4 a little more than ORT_TLB and CDC_TLB, alleviating the load on DC1 and DC2. Consequently, big improvements are observed in DC1 and DC2, while DC3 and DC4 experience minimal degradation in comparison to ORT_TLB and CDC_TLB.

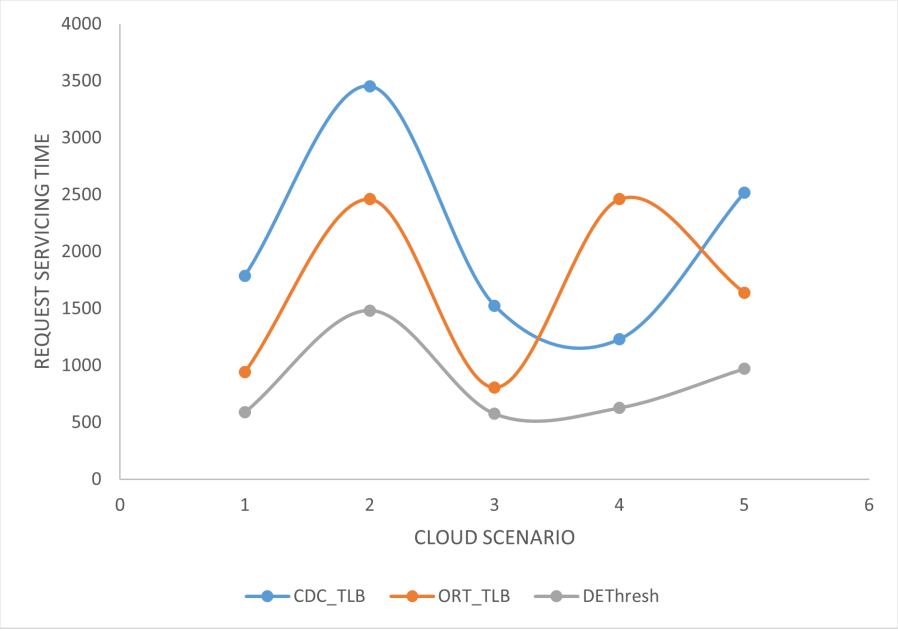

Fig. 10 Comparison of request servicing time of DC_1

To provide a more quantitative perspective on these findings, we computed the improvement and degradation in request servicing time for each DC when compared to CDC_TLB and ORT_TLB, as detailed in Tables 10 and 11. The tables highlight that the improvement in DC1 and DC2 outweighs the degradation observed in DC3 and DC4. This quantitative analysis aligns with the visual observations from Fig 14 and Fig 15.

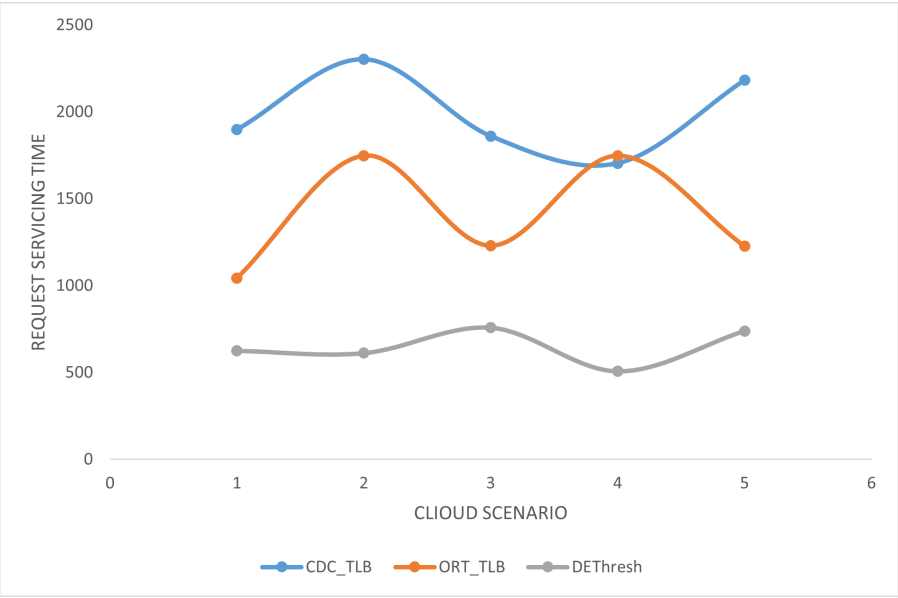

Fig. 11 Comparison of request servicing time of DC_2

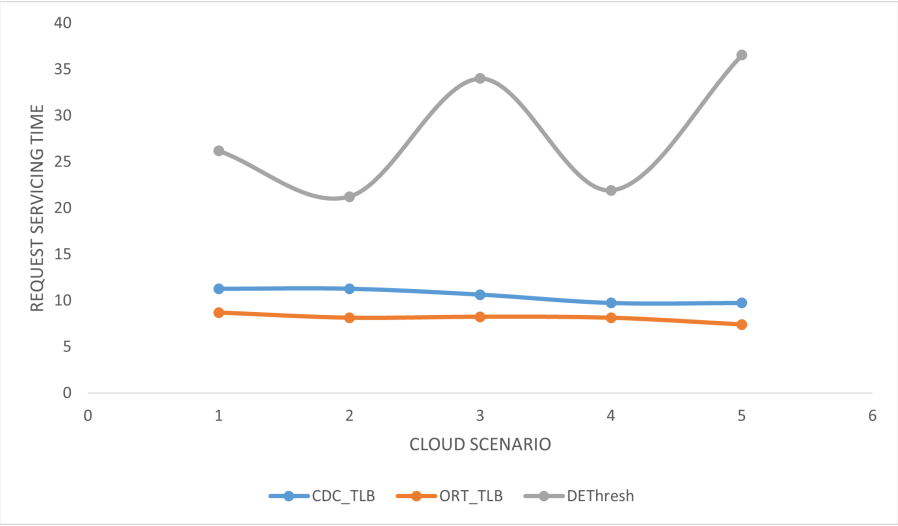

Fig. 12 Comparison of request servicing time of DC_3

Table 10 Improvement or Degradation (Milliseconds) of DEThresh compared to CDC_TLB

Datacenter	CS_1	CS_2	CS_3	CS_4	CS_5
DC_1	1198.39	1971.08	949.02	602.09	1547.21
DC_2	1274.05	1691.14	1103.26	1196.99	1443.71
DC_3	-14.94	-9.98	-23.37	-12.17	-26.81
DC_4	-153.95	-368.94	-146.49	-101.61	-1.12

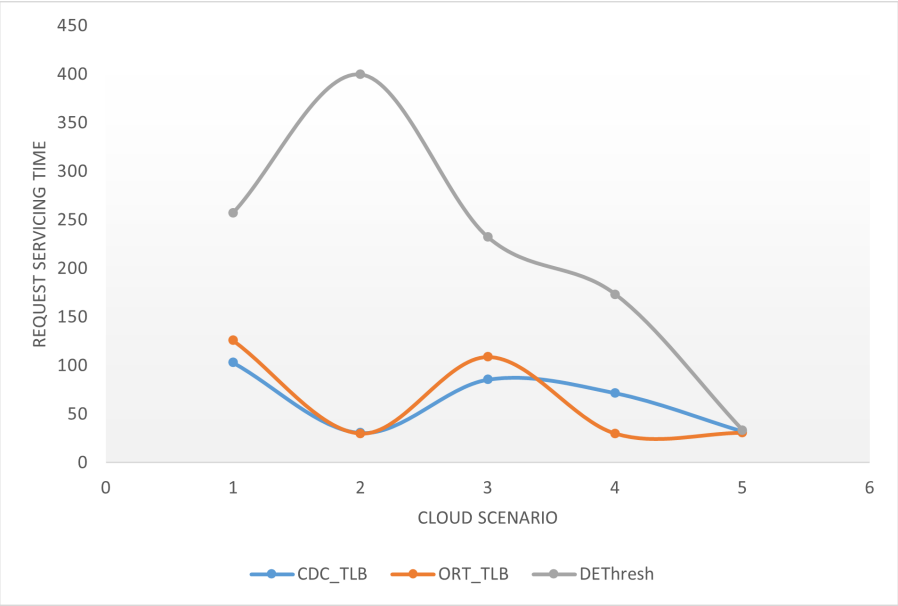

Fig. 13 Comparison of request servicing time of DC_4

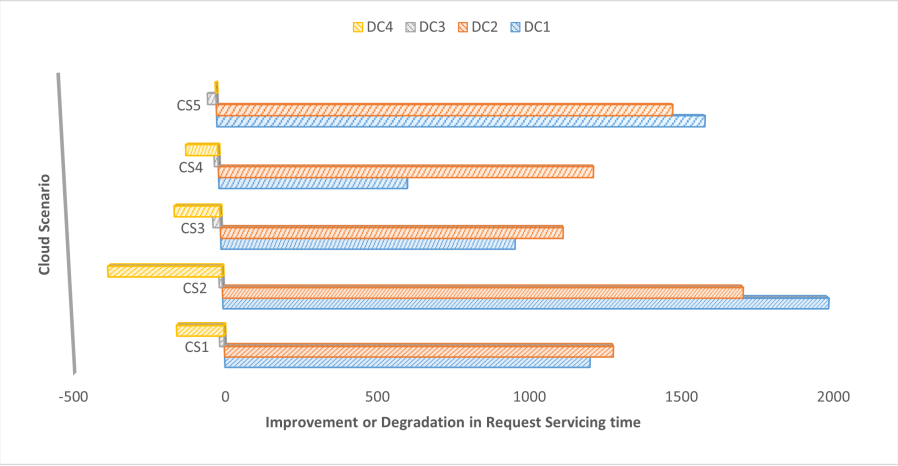

Fig. 14 Comparison of improvement or degradation of DEThresh compared to CDC_TLB for individual datacenters

Table 11 Improvement or Degradation (Milliseconds) of DEThresh compared to ORT_TLB

Datacenter	CS_1	CS_2	CS_3	CS_4	CS_5
DC_1	350.62	978.99	228.51	1833.3	668.47
DC_2	418.46	1135.26	471.92	1239.08	487.46
DC_3	-17.49	-13.11	-25.76	-13.77	-29.14
DC_4	-131.18	-369.84	-123.22	-143.52	-2.66

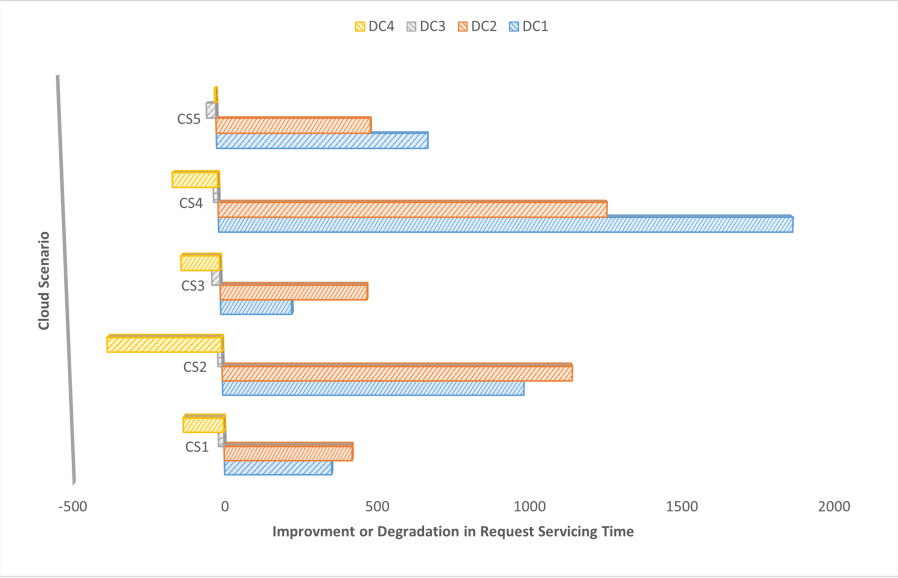

Fig. 15 Comparison of improvement or degradation of DEThresh compared to ORT_TLB for individual datacenters

Despite the observed degradation in DC3 and DC4, our DEThresh algorithm's overall impact on the DCs is positive as shown by the big improvements in request servicing time in DC1 and DC2. Basically, DEThresh strategically utilizes all DCs and achieves an enhancement in the overall performance as compared to other algorithms. This overarching improvement is encapsulated in Table 3 and Fig 4, solidifying the effectiveness of DEThresh in optimizing the request servicing time across the entire DC configuration.

5.6 Individual Userbase Performance

In our investigation of the performance of UBC_1 with 4 DCs, we thoroughly analyzed individual userbases, focusing on response time metrics. The results are presented in Table 12, which details the maximum, minimum, and average response times across 46 userbases within our cloud scenario. Additionally, Fig 16 visually represents the maximum response time achieved by two algorithms, DEThresh and ORT_TLB, for each userbase. Upon examination of Fig 16, a noteworthy finding emerges—DEThresh exhibits superior performance in terms of maximum response time compared to ORT_TLB. This suggests that DEThresh effectively allocates user tasks among the VMs in the DCs, thereby avoiding overloading or underutilization. The primary objective of our algorithms is to strike a balance between VM workload, preventing overburdening and underutilization. Fig 16 provides compelling evidence that our proposed DEThresh successfully fulfills this goal. Furthermore, Fig 17 delves into the minimum response time for individual userbases. It is evident from Fig 17 that DEThresh yields comparable minimum response times

Table 12 Individual Userbase Performance (Milliseconds) for 4 DCs

	Maximum		Minimum		Average	
Userbase	ORT_TLB	DEThresh	ORT_TLB	DEThresh	ORT_TLB	DEThresh
UB1	13797.47	4808.92	51.95	90.37	1123.24	1275.7
UB2	9120.4	2125.25	175.3	164.7	532.15	481.94
UB3	19047.5	4300.23	238.4	230.31	2269.42	427.48
UB4	231.44	230.17	42.22	41.51	87.27	107.22
UB5	15409.82	1478.74	377.54	401.31	915.63	589.43
UB6	4142.05	2632	164.15	169.94	462.46	405.63
UB7	14622.75	2805.47	67.04	73.3	386.05	354.7
UB8	13325.5	3050.07	65.04	69.59	328.9	301.82
UB9	15219.22	3554.18	65.82	66.54	352.11	435.09
UB10	13249.84	3267.46	67.54	72.57	377.23	358.86
UB11	3213.77	2368.34	45.21	41.17	140.38	83.46
UB12	72.8	79.06	40.5	40.9	53.27	53.45
UB13	15445.66	1818.49	65.75	75.57	368.79	384.59
UB14	5699.36	2466.97	44.37	44.11	140.05	115.68
UB15	9947.76	3553.23	66.82	70.04	368.56	341.36
UB16	7476.19	3703.81	42.87	42.43	138	84.13
UB17	14470.62	3548.69	67.54	73.04	313.65	341.28
UB18	6824.9	1555.85	39.81	40.78	138.71	90.42
UB19	8640.03	2794.64	67.81	65.79	374.3	347.65
UB20	4027.87	1841.4	157.23	161.48	284.62	225.69
UB21	7808.46	1283.13	260.5	325.09	593.2	516.84
UB22	1254.19	1095.53	161.89	159.77	264.29	229.38
UB23	13222.59	2801.68	68.8	229.2	371.06	366.97
UB24	14819.02	3511.75	65.5	72.54	372.12	303.27
UB25	10399.23	3693.34	280.19	322.11	608.26	518.71
UB26	83.58	594.09	39.97	37.14	53.85	59.27
UB27	7218.68	2738.41	160.93	158.33	282.66	224.59
UB28	9637.59	1655.89	154.74	164.75	283.46	224.67
UB29	10252.6	4475.2	160.02	159.33	282.24	225.11
UB30	81.49	77.12	40.1	40.1	53.8	53.74
UB31	10814.31	1452.73	269.52	294.67	574.18	523.33
UB32	80.1	79.16	40.62	40.61	53.75	53.69
UB33	14031.72	2445.17	265.19	296.09	555.21	771.09
UB34	4670.31	3670.73	281.68	278.67	569.29	550.75
UB35	8684.43	1272.42	163.05	165.82	297.11	228.29
UB36	4120.7	2215.14	164.73	162.31	280.56	224.46
UB37	5728.01	1812.38	167.07	164.31	290.84	229.05
UB38	69.72	71.11	37.38	41.48	53.42	53.4
UB39	7812.67	1809.32	165.64	160.32	295.99	228.07
UB40	11449.95	3740.43	42.4	40.91	145.42	98.05
UB41	9539.46	3310.54	162.58	161.44	298.28	231.59
UB42	12569.03	3841.61	69.25	73.29	351.76	319.59
UB43	71.44	2392.46	36.15	40.85	53.83	57.2
UB44	12788.89	2538.02	251.68	379.75	628.86	522.9
UB45	10785.34	1879.94	169.43	160.33	284.86	223.91
UB46	8248.93	1465.67	44.9	42.92	152.57	86.55

to ORT_TLB across all userbases. However, a more comprehensive assessment of overall response times in Fig 18 reveals that DEThresh consistently outperforms ORT_TLB. In summary, our DEThresh algorithm distributes tasks evenly among VMs, ensuring that no VM is overwhelmed with tasks or underutilized due to a lack of tasks.

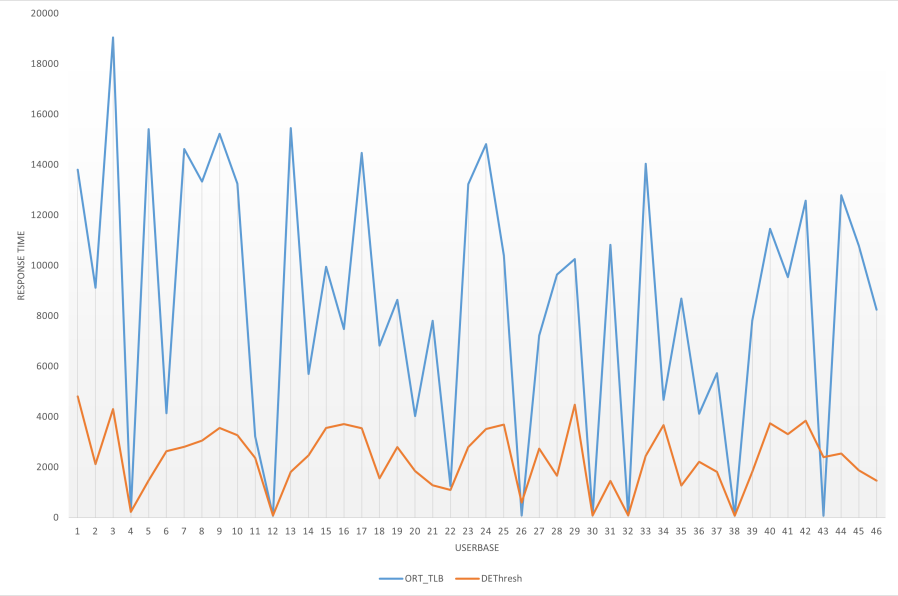

Fig. 16 Comparison of maximum response time for individual userbase

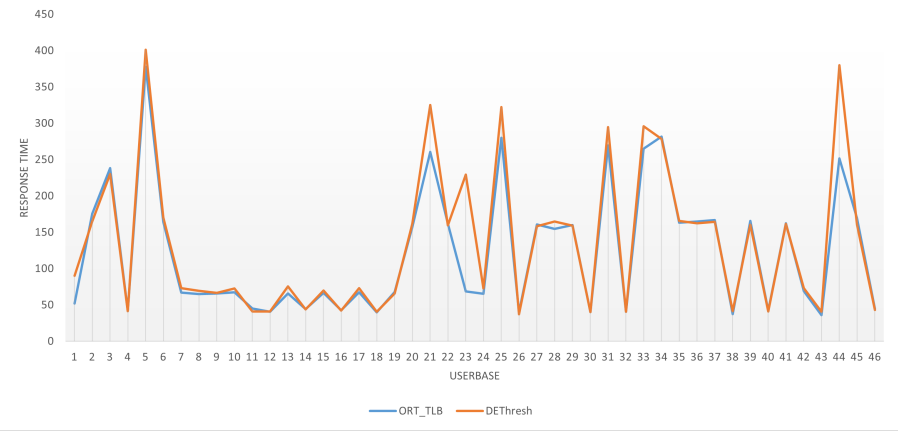

Fig. 17 Comparison of minimum response time for individual userbase

In conclusion, the results presented in Fig 16, Fig 17, and 18 collectively indicate that DEThresh achieves optimal resource utilization for VMs in DCs compared to alternative algorithms. The algorithm successfully addresses the challenge of balancing workloads and enhancing efficiency and performance in our cloud computing environment.

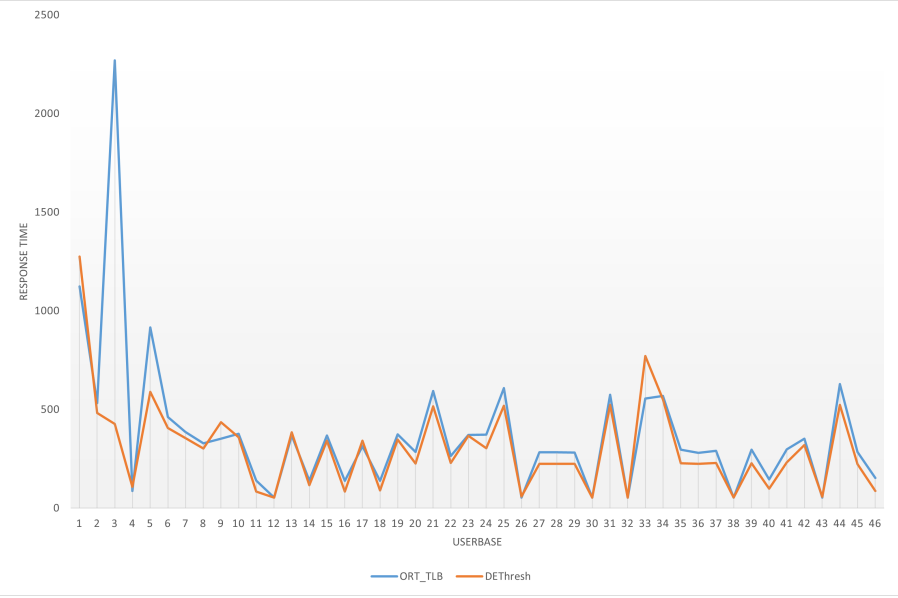

Fig. 18 Comparison of average response time for individual userbase

6 Conclusion

The integration of cloud technology has become an indispensable aspect of our daily lives, adeptly managing an extensive array of user requests to ensure the satisfaction of cloud users. In this context, our proposed DEThresh algorithm aims to optimize task processing, contributing to an enhanced user experience. The DEThresh algorithm combines a Differential Evolution-based DC selection algorithm with a Threshold-based load balancing policy.

The Differential Evolution-based DC selection algorithm draws inspiration from natural processes, employing Darwinian evolution principles to identify the most suitable DC for the user base. Given the dynamic nature of the cloud platform, the algorithm evolves over time, aligning seamlessly with the platform's changing demands. The selected DC is then utilized in the threshold-based load balancing policies, directing all user requests from a particular user base to this chosen DC.

The primary objective of the threshold-based load balancing policy is to ensure equitable distribution of workloads, maximizing resource utilization without burdening VMs with excessive tasks. This algorithm categorizes VMs as underloaded or overloaded based on their available computation power, directing user requests only to underloaded VMs. The inclusion of a gap between underload and overload thresholds facilitates proper resource allocation, preventing frequent overloading of VMs.

Our experimentation involved using the CloudAnalyst simulator to simulate various real-world cloud scenarios. By varying both user base size and the number of DCs, we observed that DEThresh consistently outperformed

baseline DC selection algorithms (CDC and ORT) in conjunction with baseline load balancing policies (RR and TLB) for both scenarios. Importantly, DEThresh demonstrated superior performance even with fewer DCs, eliminating the need for additional DCs and ensuring optimal resource utilization. This cost-effective attribute enhances the profitability of cloud service providers.

While our DEThresh algorithm exhibited superior performance within the controlled environment of the simulator, further validation in real-world cloud services is essential. Subsequent research endeavors could involve refining the algorithm by exploring different mutation, recombination, and selection strategies within the differential evolution DC selection context. Additionally, incorporating machine learning techniques for classifying VMs in the threshold-based load balancing process represents a promising avenue for further improvement.

7 Declaration Section

Ethics approval and consent to participate: Yes
Consent for publication: Yes
Availability of data and materials: Not applicable
Competing interests: Not applicable
Funding: O'Reilly Auto Parts
Authors' contributions - provide individual author contribution: Equal contribution from both author's
Acknowledgements: This material is based upon work supported by a grant from the O'Reilly Auto Parts.
Authors' information: Shusmoy Chowdhury, Ajay Katangur

References

[1] Buyya, R., Broberg, J., Goscinski, A.M.: Cloud computing: Principles and paradigms. John Wiley & Sons (2010)

[2] Dasgupta, K., Mandal, B., Dutta, P., Mandal, J.K., Dam, S.: A genetic algorithm (ga) based load balancing strategy for cloud computing. Procedia Technology **10**, 340–347 (2013)

[3] Jafari Navimipour, N., Sharifi Milani, F.: A comprehensive study of the resource discovery techniques in peer-to-peer networks. Peer-to-Peer networking and applications **8**, 474–492 (2015)

[4] Navimipour, N.J.: A formal approach for the specification and verification of a trustworthy human resource discovery mechanism in the expert cloud. Expert Systems with Applications **42**(15-16), 6112–6131 (2015)

[5] Jafari Navimipour, N., Sharifi Milani, F.: A comprehensive study of the resource discovery techniques in peer-to-peer networks. Peer-to-Peer networking and applications **8**, 474–492 (2015)

[6] Rani, D., Ranjan, R.K.: A comparative study of saas, paas and iaas in cloud computing. International Journal of Advanced Research in Computer Science and Software Engineering **4**(6) (2014)

[7] Kavis, M.: Architecting the cloud. Wiley Online Library (2023)

[8] Hacigumus, H., Iyer, B., Mehrotra, S.: Providing database as a service. In: Proceedings 18th International Conference on Data Engineering, pp. 29–38 (2002). IEEE

[9] Navimipour, N.J., Rahmani, A.M., Navin, A.H., Hosseinzadeh, M.: Expert cloud: A cloud-based framework to share the knowledge and skills of human resources. Computers in Human Behavior **46**, 57–74 (2015)

[10] Punj, R., Kumar, R.: Technological aspects of wbans for health monitoring: a comprehensive review. Wireless Networks **25**, 1125–1157 (2019)

[11] Banerjee, S., Adhikari, M., Kar, S., Biswas, U.: Development and analysis of a new cloudlet allocation strategy for qos improvement in cloud. Arabian Journal for Science and Engineering **40**, 1409–1425 (2015)

[12] Rathi, D.R., Sharma, V., Bole, S.K.: Round robin data center selection in single region for service proximity service broker in cloud analyst. International Journal of Computer & Technology **4**(2), 254–260 (2013)

[13] Jaikar, A., Kim, G.-R., Noh, S.-Y.: Effective data center selection algorithm for a federated cloud. Advanced Science and Technology Letters **35**, 66–69 (2013)

[14] Limbani, D., Oza, B.: A proposed service broker strategy in cloudanalyst for cost-effective data center selection. International Journal of Engineering Research and Applications, India **2**(1), 793–797 (2012)

[15] Zhang, J., Huang, H., Wang, X.: Resource provision algorithms in cloud computing: A survey. Journal of network and computer applications **64**, 23–42 (2016)

[16] Radi, M.: Weighted round robin policy for service brokers in a cloud environment. In: The International Arab Conference on Information Technology (ACIT2014), Nizwa, Oman, pp. 45–49 (2014)

[17] Kapgate, D.: Improved round robin algorithm for data center selection in cloud computing. International Journal of Engineering Sciences and

Research Technology **3**(2), 686–691 (2014)

[18] Semwal, A., Rawat, P.: Performance evaluation of cloud application with constant data center configuration and variable service broker policy using cloudsim. International Journal of Enhanced Research In Science Technology & Engineering **3**(1), 1–5 (2014)

[19] Kapgate, D.: Weighted moving average forecast model based prediction service broker algorithm for cloud computing. International Journal of Computer Science and Mobile Computing **3**(2), 71–79 (2014)

[20] Naha, R.K., Othman, M.: Cost-aware service brokering and performance sentient load balancing algorithms in the cloud. Journal of Network and Computer Applications **75**, 47–57 (2016)

[21] Arya, D., Dave, M.: Priority based service broker policy for fog computing environment. In: Advanced Informatics for Computing Research: First International Conference, ICAICR 2017, Jalandhar, India, March 17–18, 2017, Revised Selected Papers, pp. 84–93 (2017). Springer

[22] Manasrah, A.M., Smadi, T., ALmomani, A.: A variable service broker routing policy for data center selection in cloud analyst. Journal of King Saud University-Computer and Information Sciences **29**(3), 365–377 (2017)

[23] Chudasama, D., Trivedi, N., Sinha, R.: Cost effective selection of data center by proximity-based routing policy for service brokering in cloud environment. International Journal of Computer Technology and Applications **3**(6), 2057–2059 (2012)

[24] Limbani, D., Oza, B.: A proposed service broker strategy in cloudanalyst for cost-effective data center selection. International Journal of Engineering Research and Applications, India **2**(1), 793–797 (2012)

[25] Mishra, R.K., Kumar, S., Naik, B.S.: Priority based round-robin service broker algorithm for cloud-analyst. In: 2014 IEEE International Advance Computing Conference (IACC), pp. 878–881 (2014). IEEE

[26] Bhavani, B., Guruprasad, H.: A comparative study on resource allocation policies in cloud computing environment. Compusoft **3**(6), 893 (2014)

[27] Nandwani, S., Achhra, M., Shah, R., Tamrakar, A., Joshi, K., Raksha, S.: Weight-based data center selection algorithm in cloud computing environment. In: Artificial Intelligence and Evolutionary Computations in Engineering Systems: Proceedings of ICAIECES 2015, pp. 515–525 (2016). Springer

[28] Kishor, K., Thapar, V.: An efficient service broker policy for cloud computing environment. International Journal of Computer Science Trends and Technology (IJCST) **2**(4) (2014)

[29] Manasrah, A.M., Aldomi, A., Gupta, B.B.: An optimized service broker routing policy based on differential evolution algorithm in fog/cloud environment. Cluster Computing **22**, 1639–1653 (2019)

[30] Kessaci, Y., Melab, N., Talbi, E.-G.: A pareto-based genetic algorithm for optimized assignment of vm requests on a cloud brokering environment. In: 2013 IEEE Congress on Evolutionary Computation, pp. 2496–2503 (2013). IEEE

[31] Sidhu, A.K., Kinger, S.: Analysis of load balancing techniques in cloud computing. International Journal of computers & technology **4**(2), 737–741 (2013)

[32] Somani, R., Ojha, J.: A hybrid approach for vm load balancing in cloud using cloudsim. International Journal of Science, Engineering and Technology Research (IJSETR) **3**(6), 1734–1739 (2014)

[33] Shafiq, D.A., Jhanjhi, N., Abdullah, A.: Proposing a load balancing algorithm for the optimization of cloud computing applications. In: 2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), pp. 1–6 (2019). IEEE

[34] Patel, S., Patel, R., Patel, H., Vahora, S.: Cloudanalyst: a survey of load balancing policies. International Journal of Computer Applications **117**(21) (2015)

[35] Shafiq, D.A., Jhanjhi, N., Abdullah, A.: Load balancing techniques in cloud computing environment: A review. Journal of King Saud University –Computer and Information Sciences (2021)

[36] Nayak, S., Patel, P.: Analytical study for throttled and proposed throttled algorithm of load balancing in cloud computing using cloud analyst. International Journal of Science Technology & Engineering **1**(12), 90–100 (2015)

[37] Ghosh, S., Banerjee, C.: Priority based modified throttled algorithm in cloud computing. In: 2016 International Conference on Inventive Computation Technologies (ICICT), vol. 3, pp. 1–6 (2016). IEEE

[38] Phi, N.X., Tin, C.T., Thu, L.N.K., Hung, T.C.: Proposed load balancing algorithm to reduce response time and processing time on cloud computing. Int. J. Comput. Netw. Commun **10**(3), 87–98 (2018)

[39] Kaurav, N.S., Yadav, P.: A genetic algorithm-based load balancing approach for resource optimization for cloud computing environment. Int J Inf Comput Sci **6**(3), 175–184 (2019)

[40] Katyal, M., Mishra, A.: A comparative study of load balancing algorithms in cloud computing environment. arXiv preprint arXiv:1403.6918 (2014)

[41] Issawi, S.F., Al Halees, A., Radi, M.: An efficient adaptive load balancing algorithm for cloud computing under bursty workloads. Engineering, Technology & Applied Science Research **5**(3), 795–800 (2015)

[42] Richhariya, V., Dubey, R., Siddiqui, R.: Hybrid technique for load balancing in cloud computing using modified round robin algorithms. J Comput Math Sci **6**(12), 688–695 (2015)

[43] Garg, S.K., Versteeg, S., Buyya, R.: A framework for ranking of cloud computing services. Future Generation Computer Systems **29**(4), 1012–1023 (2013)

[44] Ghosh, R., Longo, F., Naik, V.K., Trivedi, K.S.: Modeling and performance analysis of large scale iaas clouds. Future generation computer systems **29**(5), 1216–1234 (2013)

[45] Li, J., Qiu, M., Ming, Z., Quan, G., Qin, X., Gu, Z.: Online optimization for scheduling preemptable tasks on iaas cloud systems. Journal of parallel and Distributed Computing **72**(5), 666–677 (2012)

[46] Pérez-Miguel, C., Mendiburu, A., Miguel-Alonso, J.: Modeling the availability of cassandra. Journal of Parallel and Distributed Computing **86**, 29–44 (2015)

[47] Mauch, V., Kunze, M., Hillenbrand, M.: High performance cloud computing. Future Generation Computer Systems **29**(6), 1408–1416 (2013)

[48] Sousa, E., Lins, F., Tavares, E., Cunha, P., Maciel, P.: A modeling approach for cloud infrastructure planning considering dependability and cost requirements. IEEE Transactions on Systems, Man, and Cybernetics: Systems **45**(4), 549–558 (2014)

[49] Pacini, E., Mateos, C., Garino, C.G.: Balancing throughput and response time in online scientific clouds via ant colony optimization (sp2013/2013/00006). Advances in Engineering Software **84**, 31–47 (2015)

[50] Khan, A.N., Kiah, M.M., Khan, S.U., Madani, S.A.: Towards secure mobile cloud computing: A survey. Future generation computer systems **29**(5), 1278–1299 (2013)

[51] Khorshed, M.T., Ali, A.S., Wasimi, S.A.: A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing. Future Generation computer systems **28**(6), 833–851 (2012)

[52] Lombardi, F., Di Pietro, R.: Secure virtualization for cloud computing. Journal of network and computer applications **34**(4), 1113–1122 (2011)

[53] Subashini, S., Kavitha, V.: A survey on security issues in service delivery models of cloud computing. Journal of network and computer applications **34**(1), 1–11 (2011)

[54] Zissis, D., Lekkas, D.: Addressing cloud computing security issues. Future Generation computer systems **28**(3), 583–592 (2012)

[55] Jennings, B., Stadler, R.: Resource management in clouds: Survey and research challenges. Journal of Network and Systems Management **23**, 567–619 (2015)

[56] Marinescu, D.C., Paya, A., Morrison, J.P., Olariu, S.: An approach for scaling cloud resource management. Cluster Computing **20**, 909–924 (2017)

[57] Fernández-Cerero, D., Jakóbik, A., Grzonka, D., Kołodziej, J., Fernández-Montes, A.: Security supportive energy-aware scheduling and energy policies for cloud environments. Journal of Parallel and Distributed Computing **119**, 191–202 (2018)

[58] Kumar, P., Kumar, R.: Issues and challenges of load balancing techniques in cloud computing: A survey. ACM Computing Surveys (CSUR) **51**(6), 1–35 (2019)

[59] Li, X., Ma, H., Zhou, F., Yao, W.: T-broker: A trust-aware service brokering scheme for multiple cloud collaborative services. IEEE Transactions on Information Forensics and Security **10**(7), 1402–1415 (2015)

[60] Chudasama, D., Trivedi, N., Sinha, R.: Cost effective selection of data center by proximity-based routing policy for service brokering in cloud environment. International Journal of Computer Technology and Applications **3**(6), 2057–2059 (2012)

[61] Rekha, P., Dakshayini, M.: Service broker routing polices in cloud environment: a survey. International Journal of Advances in Engineering & Technology **6**(6), 2717 (2014)

[62] Zou, D., Liu, H., Gao, L., Li, S.: A novel modified differential evolution algorithm for constrained optimization problems. Computers & mathematics with applications **61**(6), 1608–1623 (2011)

[63] Tailong, V., Dimri, V.: Load balancing in cloud computing using modified optimize response time. International Journal of Advanced Research in Computer Science and Software Engineering **6**(5) (2016)

[64] Quarati, A., D'Agostino, D.: Moea-based brokering for hybrid clouds. In: 2017 International Conference on High Performance Computing & Simulation (HPCS), pp. 611–618 (2017). IEEE

[65] Kofahi, N.A., Alsmadi, T., Barhoush, M., Al-Shannaq, M.A.: Priority-based and optimized data center selection in cloud computing. Arabian Journal for Science and Engineering **44**(11), 9275–9290 (2019)

[66] Mohamed, A.W., Mohamed, A.K.: Adaptive guided differential evolution algorithm with novel mutation for numerical optimization. International Journal of Machine Learning and Cybernetics **10**, 253–277 (2019)

[67] Wang, S., Li, Y., Yang, H.: Self-adaptive mutation differential evolution algorithm based on particle swarm optimization. Applied Soft Computing **81**, 105496 (2019)

[68] Khan, M.A.: Optimized hybrid service brokering for multi-cloud architectures. The Journal of Supercomputing **76**(1), 666–687 (2020)

[69] Naha, R.K., Othman, M.: Cost-aware service brokering and performance sentient load balancing algorithms in the cloud. Journal of Network and Computer Applications **75**, 47–57 (2016)

[70] Cui, Z., Zhao, T., Wu, L., Qin, A., Li, J.: Multi-objective cloud task scheduling optimization based on evolutionary multi-factor algorithm. IEEE Transactions on Cloud Computing (2023)

[71] Geng, S., Wu, D., Wang, P., Cai, X.: Many-objective cloud task scheduling. IEEE Access **8**, 79079–79088 (2020)

[72] Das, S., Suganthan, P.N.: Differential evolution: A survey of the state-of-the-art. IEEE transactions on evolutionary computation **15**(1), 4–31 (2010)

[73] Ahmad, M.F., Isa, N.A.M., Lim, W.H., Ang, K.M.: Differential evolution: A recent review based on state-of-the-art works. Alexandria Engineering Journal **61**(5), 3831–3872 (2022)

[74] Ruse, M.: Charles darwin on human evolution. Journal of Economic Behavior & Organization **71**(1), 10–19 (2009)

[75] Gämperle, R., Müller, S.D., Koumoutsakos, P.: A parameter study for differential evolution. Advances in intelligent systems, fuzzy systems,

evolutionary computation **10**(10), 293–298 (2002)

[76] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., *et al.*: A view of cloud computing. Communications of the ACM **53**(4), 50–58 (2010)

[77] Iosup, A., Ostermann, S., Yigitbasi, M.N., Prodan, R., Fahringer, T., Epema, D.: Performance analysis of cloud computing services for many-tasks scientific computing. IEEE Transactions on Parallel and Distributed systems **22**(6), 931–945 (2011)

[78] Dillon, T., Wu, C., Chang, E.: Cloud computing: issues and challenges. In: 2010 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 27–33 (2010). Ieee

[79] Dinh, H.T., Lee, C., Niyato, D., Wang, P.: A survey of mobile cloud computing: architecture, applications, and approaches. Wireless communications and mobile computing **13**(18), 1587–1611 (2013)

[80] Wickremasinghe, B., Calheiros, R.N., Buyya, R.: Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In: 2010 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 446–452 (2010). IEEE

[81] Neves Calheiros, R., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience, 23–50 (2011)

[82] Wickremasinghe, B., Calheiros, R.N., Buyya, R.: Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In: 2010 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 446–452 (2010). IEEE

[83] Chowdhury, S., Katangur, A.: Threshold based load balancing algorithm in cloud computing. In: 2022 IEEE International Conference on Joint Cloud Computing (JCC), pp. 23–28 (2022). https://doi.org/10.1109/JCC56315.2022.00011

[84] Chowdhury, S., Katangur, A., Sheta, A., Psayadala, N.R., Liu, S.: Genetic algorithm based service broker policy to find optimal datacenters in cloud services. In: 2023 8th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), pp. 270–278 (2023). https://doi.org/10.1109/ICCCBDA56900.2023.10154791

[85] Chowdhury, S., Katangur, A.: Optimal datacenter selection for cloud services using swarm intelligence. In: 2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pp. 1366–1371 (2023). https://doi.org/10.1109/CSCWD57460.2023.10152594

[86] Katangur, A., Akkaladevi, S., Vivekanandhan, S.: Priority weighted round robin algorithm for load balancing in the cloud. In: 2022 IEEE 7th International Conference on Smart Cloud (SmartCloud), pp. 230–235 (2022). IEEE

[87] Mulla, B.P., Krishna, C.R., Tickoo, R.K.: Load balancing algorithm for efficient vm allocation in heterogeneous cloud. International Journal of Computer Networks & Communications (IJCNC) Vol **12** (2020)

[88] Shahid, M.A., Alam, M.M., Su'ud, M.M.: Performance evaluation of load-balancing algorithms with different service broker policies for cloud computing. Applied Sciences **13**(3), 1586 (2023)