Chowdhury *et al. Journal of Cloud Computing* (2024) 13:156
https://doi.org/10.1186/s13677-024-00719-4

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

Optimization of datacenter selection through a genetic algorithm-driven service broker policy

Shusmoy Chowdhury[1], Ajay Katangur[1*] and Alaa Sheta[2]

Abstract

Establishing an optimal datacenter selection policy within the cloud environment is paramount to maximize the performance of the cloud services. Service broker policy governs the selection of datacenters for user requests. In our research, we introduce an innovative approach incorporating the genetic algorithm with service broker policy to assist cloud services in identifying the most suitable datacenters for specific userbases. The effectiveness of our proposed genetic algorithm was rigorously evaluated through experiments conducted on CloudAnalyst platform. The results clearly indicate that our proposed algorithm surpasses existing service broker policies and previous research works done in this field in terms of reducing response time and data processing time. The results analysis validates its efficacy and potential for enhancing cloud service performance and reducing the cost of overall cloud infrastructure.

Keywords Cloud computing, Genetic algorithm service broker policy

Introduction

Cloud computing has gained widespread popularity by offering various industries a broad spectrum of services. This has reduced user costs and significantly increased profitability for cloud service providers [1, 2]. The advent of cloud computing has ushered in a transformational era for the internet ecosystem [3]. It has provided server less computing experience [4, 5] for the user and replaced the previous paradigm where substantial computing power was required on the client side.

In the modern cloud computing environment, datacenters (DCs) play a critical role in efficiently processing user requests and delivering quick responses, which is essential for ensuring high user satisfaction. DCs are managed and overseen by third parties who offer pay-per-use computing infrastructures to cloud users through internet access [6, 2]. With its vast on-demand resources, cloud computing is distributed extensively over the internet [7–12]. The cloud infrastructure encompasses a distributed network of DCs deployed across diverse geographical locations, where virtual machines (VMs) are instantiated on available hardware resources to fulfill user requests dynamically [13]. Cloud service providers strategically allocate services across these DCs, considering diverse pricing strategies and performance attributes to ensure the reliability and accessibility of cloud-based services. However, cloud users often exhibit varied quality of service (QoS) preferences, such as a preference for cost-effective DCs or having high-performance characteristics. Consequently, a cloud service broker (CSB) acts as an intermediary for cloud users, routing their requests to the most suitable datacenter based on specified performance and cost criteria [14].

The CSB employs brokering policies that dynamically select the optimal DC for each incoming user request. The datacenter(DC) selection process is pivotal

*Correspondence:
Ajay Katangur
ajaykatangur@missouristate.edu
[1] Department of Computer Science, Missouri State University, 901 S National Ave, Springfield 65897, Missouri, USA
[2] Department of Computer Science, Southern Connecticut State University, 501 Crescent St, New Haven 06515, Connecticut, USA

Springer Open

© The Author(s) 2024. **Open Access** This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/4.0/.

in influencing cloud resource utilization, performance metrics, service costs, and overall user satisfaction. The increasing adoption of cloud services has led to a corresponding rise in the costs associated with constructing new DCs. A key factor driving these costs is the low resource utilization ratio, as the establishment and maintenance of additional DCs require substantial financial investments. Many DCs operate at sub-optimal resource utilization levels, with ratios between 5% and 15% [15].

Our research is motivated by the need to address inefficiencies in DC selection within cloud environments. Strategically selecting DC locations across various regions can address this inefficiency by allowing a few well-placed DCs to effectively serve a large portion of the global userbase. This strategy reduces the need for numerous smaller DCs and enables existing DCs to dynamically scale during peak usage periods, thereby lessening the requirement for additional facilities. As a result, CSB algorithms are crucial for enhancing efficiency of the cloud environment. These algorithms ensure an equitable distribution of workloads while minimizing response and data processing times. The robust and optimal selection of DCs is essential to meet user requirements and adapt to the current state of the cloud environment for each incoming request.

Current service broker policies are insufficient for handling the dynamic nature of cloud ecosystems, where user requirements, network conditions, and DC workloads fluctuate continually. As cloud services expand globally, selecting the right DC in real time becomes crucial for maintaining QoS and reducing operational costs. The state of art service broker policies and existing research work were unable to handle the dynamic changes of the cloud environment, hence often overcrowded the DCs with request. This will degrade the overall performance of the cloud environment and instantiate building new DCs to fulfill the demand. Therefore, we propose an innovative approach that leverages the power of Genetic Algorithms (GA) to dynamically optimize service broker policies. This evolutionary approach introduces a flexible, adaptive solution capable of improving DC selection by iteratively refining choices based on performance metrics such as response time and data processing time.

The main contribution of our research is the development of evolutionary service broker policy that harnesses the power of GA to identify the most suitable DC for userbases. Our goal is to significantly outperform existing methods in both reducing response times and optimizing DC resource utilization. In the dynamic landscape of a cloud environment, a perpetual sequence of activities unfolds. On an hourly basis, the ingress of new requests introduces fluctuations in both quantity and size, necessitating an adaptive and efficient service broker policy over time. By integrating GA into the DC selection process, our approach enables cloud services to adapt to changing conditions and user requirements in real time. The conceptual underpinning of genetic algorithms and meta-heuristic search methods is rooted in Darwin's theory of evolution [16]. Such algorithms, characterized by their ability to refine and enhance outcomes iteratively considering the dynamic changes of the cloud environment and exhibit a propensity to yield superior results over successive iterations. Operating on the principles of natural selection and survival of the fittest, these algorithms progress toward optimal solutions by building upon the most favorable outcomes from each generation across multiple iterations [17].

We have modified the GA with novel elitist selection, crossover and mutation strategies appropriate for a cloud environment. The proposed algorithm has been rigorously tested on the CloudAnalyst platform under scaled down version of real world cloud scenarios. Our results demonstrate superior performance compared to traditional policies, particularly in reducing response and processing times while maintaining cost efficiency. Furthermore our results shows that the proposed service broker policy can achieve similar results like as the traditional service broker policies with less number of DCs. This indicates our proposed service broker policy can achieve better resource utilization compared to others. As a result the necessity of new DCs will decrease, thus save the cost of building new DCs. It will have a direct impact on the environment by reducing the global warming gasses from the new DCs. Our research contributes to the ongoing efforts to enhance cloud service performance and provides a robust framework for future advancements in cloud infrastructure management.

The structure of this paper is as follows: "Related work" section provides a concise overview of previous work on service broker policies in cloud computing, highlighting their advantages and challenges. In "Service broker policy problem" section, we have explained cloud computing service broker policy problem. "Problem encoding for genetic algorithm" section, we explain how we can represent the service broker problem in the domain of GA. "Proposed genetic algorithm" section delves into the details of our proposed GA for service broker policy with time complexity analysis and real world implementation of the proposed GA. In "Environment setup" section, we provide a detailed description of the CloudAnalyst simulation environment and simulation models imitating scaled-down versions of real-world cloud scenarios for the purpose of performance analysis.. In "Result analysis" section, we analyze our algorithm's results and performance comparing with traditional and other evolutionary service broker policy.

"Real-world adaptation of our proposed GA" section demonstrates real world case study of our GA driven service broker policy adaptation and analyzed the expected outcomes and improvement from the adaptation. Finally, in "Conclusion" section, we present a summary of our research findings and contributions and concluded with future improvement.

Related work

Extensive research has been conducted in cloud computing, addressing various challenges inherent to the field, including the strategic selection of DCs [18, 19], resource under utilization, security concerns, and so on. The primary objective of the service broker policy is to identify the most suitable DC adeptly, ensuring optimal performance for the userbase. To achieve this, the service broker policy must navigate diverse factors such as cost, time, and availability in its decision-making process. Noteworthy considerations encompassed by researchers involve the minimization of costs [20–23], response time enhancement [24–27], and the optimization of execution time [28, 29] when formulating an effective service broker policy.

The distance between users and the DCs plays a critical role in the performance of the cloud service [30, 31]. So the location-based service broker policy [29, 32] must consider the DC cost and network latency between DC and user [29, 32]. The weighted sum function considers multiple conditions in a service broker policy [33, 34]. Cost minimization is also considered while minimizing the network latency in location-based service broker policy [31, 35]. Round Robin-based service broker policy distributes incoming user requests among DCs within the same region in a round-robin manner [20]. However, the algorithm considers all the DCs to be in the same region and have the same configuration and overlooks the cost of the DC. To resolve the same configuration issue, a priority-based brokering algorithm is designed where higher processing power DCs are given higher priority, and more requests are sent to those DCs [36]. Again, sending more requests to the DCs decreases its performance and causes network congestion. Also, priority is given based on the processing power without considering the DC cost. Again, the static brokering mechanism statically deploys DC among multiple cloud service providers without considering dynamic changes in the cloud environment, such as DC loading and network congestion levels [37].

Closest Datacenter Policy(CDC) is centered on the prioritization of DC selection based on their proximity to the userbase, as gauged by network latency [38, 39]. When a service request emanates from the userbase, the CSB steers the incoming request towards the closest DC, identified by the shortest network latency. However, the policy can overload the nearest DC and its communication pathway. This vulnerability arises from the policy's need for more consideration of the intricacies of the incoming request, such as processing requirements and data size, as well as the available network bandwidth at the moment the request is made.

In the weight-based brokering algorithm, all the DCs are assigned weights based on the number of available VMs in the DC, and the number of requests directed to each DC is proportional to its weight [40]. In addition, fuzzy engines assign a weight for each DC in a particular region, considering each DC's performance and cost characteristics [41]. However, the fuzzy system considers static performance characteristics such as the Million Instructions per Second (MIPS) rating and number of processors. Still, it does not consider dynamic changes in the cloud environment or user cost and performance preferences. As a result, the DCs are overloaded with tasks due to not considering the dynamic changes, and thus, this algorithm degrades the performance of the DCs as well as the overall cloud environment.

The weight-based or priority-based algorithms [42–44] lack adaptability to dynamic changes within the cloud environment, such as performance degradation in DCs during overload conditions and network congestion. In the cost-based refined version of the CDC policy [22], user requests within a specific region are directed to the DC with the lowest data transfer cost. Regrettably, this policy overlooks the impact of network congestion and DC costs, increasing response time. An alternative cost-aware policy diverts user requests to the neighboring region during peak hours yet fails to account for dynamic network behavior, data size, and user request requirements [22]. Moreover, a service broker policy proposes three variations of cost-aware brokering algorithms [45]. The first variant directs user requests to the lowest-cost regional DC, the second evenly distributes incoming service requests among available DCs, and the third maintains a list of least-loaded low-cost DCs to distribute user requests. Cost-aware and performance-aware static brokering algorithms may increase DC response time while disregarding dynamic factors such as DC overloading and network congestion. Conversely, performance-aware static brokering algorithms may escalate service costs, rely exclusively on static performance characteristics, and exhibit an inability to adapt to real-time changes in the cloud environment. Additionally, both algorithms consider the precise processing requirements and data size for all user requests, neglecting user cost and performance preferences.

Optimized Response Time(ORT) policy under consideration prioritizes response time as the foundational

criterion for DC selection [38, 39]. In this methodology, CSB meticulously records the response time associated with each DC. Upon initiation of a request by the userbase, the CSB strategically directs the request to the DC with the shortest recorded response time. It is crucial to highlight that the ORT algorithm assumes that all requests have uniform processing requirements, data sizes, and execution times, which may not consistently align with real-world scenarios. Furthermore, the ORT policy needs to consider the nuanced considerations of cost and performance preferences associated with individual users' accounts.

Variable Service Broker Routing Policy (VSBRP) [45] utilizing a heuristic-based approach aimed at optimizing response times. This algorithm takes into account communication channel bandwidth, latency, and user request size to achieve its goal. The VSBRP dynamically redirects user requests to DCs that offer superior response and processing times, thereby preventing overloading of any single DC. However, the authors acknowledged that the implementation of their algorithm may incur substantial financial costs and increased power consumption.

The characteristics of cloud computing indicate a constant influx of diverse user requests, each with unique computational demands. These attributes emphasize the crucial need for an effective service broker policy. Notably, evolutionary algorithms have demonstrated superior performance compared to conventional DC selection policies [45–49]. These algorithms, rooted in Darwinian principles [16], adapt over time to generate optimal solutions. Leveraging natural selection and the survival of the fittest, they iteratively refine outcomes across successive generations, progressively achieving optimal solutions. Analogously, the evolution of humans from apes exemplifies this concept [16]. The Pareto-based genetic algorithm was used for cloud brokering [48] but the authors did not provide any real-world or simulation results for their algorithm. However, the authors provided theoretical results for the validation of the algorithms. Furthermore, other authors using the evolutionary algorithms [45, 46] for service broker policy had provided satisfactory results which motivated us to use the evolutionary algorithms to enhance the cloud performance.

Particle Swarm Optimization (PSO) [50] using multi-objective optimization technique has been designed for efficient IoT service placement in fog-enabled networks. The proposed algorithm focuses on minimizing service cost, response time, and maximizing fog resource utilization and throughput by using a fog-cloud control middle ware based on the MADE-k model. However the proposed algorithm assumes the fog layer will meet most demands, but in real-world scenarios, fog nodes may be less reliable or resource-limited, potentially increasing reliance on cloud infrastructure.

Differential evaluation-based(DE) service broker policy named "OSBRP" was proposed to find an optimal DC for the request based on the user's constraints to achieve better response time using minimization of overall DC cost [46]. The author gives a clear definition of the fitness function used in the research. The authors did not provide the problem encoding or description of the chromosome or individual. As a result, it is unclear about the implementation of the algorithm in the service broker policy. Another multi objective DE based service broker policy is designed focusing on the minimization of the response time and data processing time [51]. However DE primarily focuses on exploration through mutation, which may lead to less effective exploitation of the best solutions found. Furthermore DE can sometimes lead to a loss of diversity within the population if the parameters are not set appropriately. Although the results show better improvement, the author did not analyzed the scalability and applicability in the real world cloud scenarios.

Multi-objective Genetic Algorithm Cloud Brokering combined two objectives, response time and cost of the VM, and provided a Pareto set of solutions for DC selection [47]. However, the authors did not provide any statistical performance data considering the achievement of the service broker policy. An Ant Colony Optimization (ACO) meta-heuristic service broker policy directs the user requests toward the VM having the shortest processing time [49]. Again the policy ignored the DC cost and run-time network congestion. A Cuckoo optimization-based service brokering algorithm has been proposed to select optimum DCs for the user [52]. Still, this policy focused on DC response time without considering DC cost and user cost/performance preference (Table 1).

Our research endeavors to leverage evolutionary algorithms for enhancing DC selection processes. Previous studies have successfully employed evolutionary algorithms in service broker policies, primarily focusing on cost minimization, however the drawbacks of these algorithms was increased response and data processing times [45–48]. Hence, our focus is on minimizing response and data processing times. Thus it can enhance the resource utilization of the DCs which leads to less number of DCs and overall cost minimization of the cloud environment. The less number of DCs will also decrease the necessity of building new DCs and thus reduce the emission of global warming gases from these DCs. So, our proposed GA will indirectly have a positive impact on the climate change. Our approach involves initiating DC selection algorithms with a random population and iteratively evolving them using

Table 1 Summary of the existing DC Selection Policies in the Cloud Environment

Algorithm	Utilized techniques	Limitation	Evaluation tools	Performance metrics	Datasets
Proximity Based Datacenter Selection [38, 39]	Minimizes distance between user and nearest datacenter	Causes overcrowding and resource underutilization in nearby DCs	CloudAnalyst	Response and data processing times	Simulated userbase (CloudAnalyst configurations)
Optimized Response Time (ORT) [38, 39]	Minimizes response time	Can lead to unbalanced loads and overcrowding in optimal DCs	CloudAnalyst	Response and data processing times	Simulated userbase (CloudAnalyst configurations)
Priority-Based DC Selection [40–44]	Prioritize datacenters based on processing speed	Can increase costs when prioritizing aster datacenters	CloudAnalyst	Response and data processing times	Simulated userbase (CloudAnalyst configurations)
PSO-based IoT in Fog Networks [50]	Particle Swarm Optimization	Fog nodes may be less reliable or resource-limited, potentially increasing reliance on cloud infrastructure	CloudSim (Fog Computing Extension)	Energy consumption, resource utilization	Simulated IoT/fog nodes dataset
VSBRP [45]	Heuristic-based optimization	High financial costs and energy consumption	CloudAnalyst	Response and data processing times	Simulated DC and user requests
Pareto-based GA for Cloud Brokering [47]	Multi-objective GA	Complex Pareto sets require large data	Theoretical analysis, no real-world simulations	Theoretical validation (response time, VM cost)	N/A
ACO-based Service Broker [49]	Ant Colony Optimization (ACO)	Ignores network congestion and data processing time	CloudSim	Processing time, DC utilization	Simulated user requests and VM deployments
Cuckoo Optimization Service Broker [52]	Cuckoo Search Optimization	Does not consider cost and user performance preferences	CloudSim	Response time	Multi-cloud configurations, user requests
OSBRP using DE [46]	Differential Evolution	Lacks detailed encoding of chromosomes and implementation details	CloudAnalyst	Response time, cost efficiency	Simulated cloud and userbase environment

evolutionary strategies to attain optimal solutions. Specifically, we have employed GA for service broker policy, enhancing the efficiency and responsiveness of cloud services. Comparative analyses against baseline DC selection policies on the CloudAnalyst platform validate the efficacy of our algorithms in real-world scenarios.

Service broker policy problem

The cloud environment is a complex ecosystem comprised of various integral components, each playing a crucial role in the overall workflow. These components include the userbase, DCs, load balancing, and service broker policies. At the core of this ecosystem are two primary components: the userbase and the DCs. The userbase, consisting of individual users, initiates user requests within the cloud environment subsequently, the DCs process these user requests, which then transmit the responses back to the userbase. The dynamic interaction between these two fundamental components forms the backbone of the cloud environment's operational framework.

To optimize the performance of the cloud environment, and applications, the service providers undertake a meticulous configuration process. This involves tailoring the environment to meet specific requirements, considering user request sizes, peak and off-peak hour timings, the volume of user requests during these hours, DC configurations, and load balancing policies. These parameters are critical in ensuring the efficient functioning of the cloud infrastructure.

The service broker policy facilitates the intricate orchestration of the cloud environment, a pivotal mechanism within the cloud provider's servers. This policy considers the application and service providers' inputs, encompassing their specific requirements and preferences. The service broker policy then strategically designates a particular DC for a given userbase at a specific time.

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

$$C \leftarrow \{UB, DC\} \tag{1}$$

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.

$$\begin{aligned} UB &= \{Ub_1, Ub_2, Ub_3 \ldots Ub_u\} \\ DC &= \{Dc_1, Dc_2, Dc_3 \ldots Dc_d\} \end{aligned} \tag{2}$$

The cloud service must employ an optimal service broker policy to handle user requests efficiently. The objective is to identify the most suitable DC for the userbases at any given moment. "Figure 1" illustrates the process of DC selection for the userbases.

$$Dc_d \leftarrow Ub_u \tag{3}$$

The diversity of attributes associated with different components, such as latency, bandwidth, request volume, data transfer costs, and request sizes during peak and off-peak hours in the cloud environment, further complicates the decision-making process for service broker policy. Therefore, our service broker policy must be efficient and effective as time progresses. For this reason, we have tried to employ evolutionary algorithms, specifically genetic algorithm, in our proposed service broker policy.

The motive behind selecting evolutionary algorithms, classified as meta-heuristic search methodologies, draw inspiration from Darwinian evolution principles [16]. These algorithms dynamically adapt over successive iterations, aiming to generate optimal solutions. Leveraging the paradigm of natural selection and the survival of the fittest, this algorithmic approach advances iteratively, traversing various generations and incorporating the most successful outcomes from each iteration. In the service broker policy context, the evolutionary algorithm commences with random solutions and progressively refines them using evolutionary strategies to converge toward the optimal solution. Notably, the GA, renowned as one of the preeminent and widely employed evolutionary algorithms globally, is chosen for implementation in this research for the service broker policy.

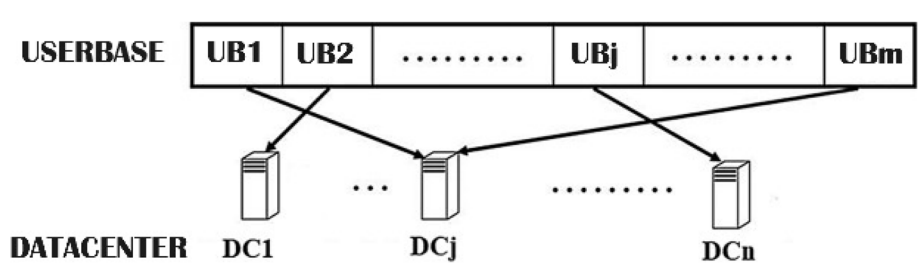

Fig. 1 Service broker policy

Problem encoding for genetic algorithm

To design a GA-based service broker policy, we need to define the cloud service broker policy in the context of genetic algorithms. The critical factors of the cloud environment are the userbase and the DCs. The userbase, which is a group of users, sends the user request, and the DCs process those requests and send the results or feedback to the userbase. Again, in the context of genetic algorithm, there are three crucial factors: chromosomes, population, and fitness function.

Chromosome structure

Chromosomes, also known as individuals, are the driving force of the genetic algorithm. Chromosomes define the solution structure of the genetic algorithm. Chromosomes evolve through the steps of the genetic algorithm and direct the algorithm towards an optimal solution. The chromosome structure of the cloud service broker policy is an array structure. The index of the array marks the userbase number, and the element of the array defines the DC allocated to that userbase. Table 2 shows the chromosome sample used in our research. The number of userbases present in the cloud environment determines the size of the chromosome. This chromosome configuration shapes the evolutionary process, leading the algorithm toward refined solutions.

Population

Population represents the number of chromosomes that participate in the genetic algorithm. One of the individuals or chromosomes of the population will be considered as the solution of the genetic algorithm. The population's size directly affects the genetic algorithm's time complexity. Every iteration of the algorithm is defined as generations. If the population size is significant, we need more generations to find the solutions. Again, if the population size is small, the algorithm may be unable to find the optimum solution because of less exploration and exploitation in search space. So, defining the population size plays a vital role in the genetic algorithm.

Furthermore, the initialization of the generations is essential for genetic algorithm. There must be diversity in the population. To uphold diversity within the context of our computational approach, we adopt a strategy wherein the initial population is generated through a random process. The deliberate introduction of randomness in the population ensures a broad exploration of diverse search spaces within the cloud computing domain.

Table 2 Chromosome representation of the genetic algorithm

UB_1	UB_2	UB_3	UB_4	....	....	UB_n
DC_2	DC_m	DC_1	DC_5	....	....	DC_3

Additionally, we incorporate chromosomes generated through the CDC policy into the initial solution. This assumption is grounded in the belief that DCs in the CDC policy embody a certain fitness level within the evolutionary framework. Consequently, these DCs are expected to persevere and progress toward optimal solutions. Conversely, if the DCs in CDC exhibit characteristics less aligned with evolutionary fitness, they will engage in the genetic algorithm, generating chromosomes that progressively evolve in a manner conducive to optimal solution search. This dual strategy seeks to harness both random exploration and guided evolution to enhance the overall efficacy of the genetic algorithm in optimizing solutions within cloud computing.

Fitness function

The fitness function measures the quality of the represented solution. The fitness function is a function that takes a candidate solution to the problem as input and produces as output how "fit" or "good" the solution is concerning the problem in consideration. The fitness function needs to be fast in computing and providing results about the individuals. So, to design an effective and efficient service broker policy, we need to focus on the response time of the userbase and data processing time of the DCs. In a cloud environment, varying numbers of VMs across different DCs introduce a dynamic element to the computational landscape. Moreover, the transfer cost associated with each DC exhibits variability, influenced by geographic region and the number of physical machines deployed. To enhance efficiency, our research endeavors to incorporate these multifaceted factors into evolutionary algorithms. The primary objective of this investigation is to discern and designate an optimal DC tailored to specific userbases.

Within this framework, our overarching goal is to optimize response and data processing time performance metrics within the identified DC. By systematically integrating considerations related to VM distribution, data transfer costs influenced by geographic and infrastructural factors, and the unique requirements of userbases, our GA seeks to refine the selection process iteratively. This meticulous optimization aims to achieve a configuration that minimizes response and data processing times and attains an optimal balance between computational efficiency and associated costs within the cloud environment. This research contributes to the broader discourse on resource allocation and management strategies in cloud computing, offering insights into the nuanced interplay between diverse parameters to achieve optimal performance.

Response time

Response time defines the time elapsed between initiating a user's request and receiving the corresponding response from the cloud-based infrastructure. The response time of a userbase, R_t, can be defined as

$$R_t = F_t - Ar_t + T_d \tag{4}$$

where Ar_t is the arrival time of the user request, F_t is the finish time of the user request, and T_d is the transmission delay.

Again, transmission delay, T_d, refers to the time data traverses the network between a source and destination within the cloud infrastructure.

$$T_d = T_l + T_{tr} \tag{5}$$

Here, T_l is the network latency, and T_{tr} is the time to transfer a single request's data size(D) from source to destination.

$$T_{tr} = \frac{D}{Bw_{peruser}} \tag{6}$$

$$Bw_{peruser} = \frac{Bw_{total}}{N_t} \tag{7}$$

Here Bw_{total} is the total available bandwidth for the userbase, and N_t is the number of user requests currently in transmission. The values of these parameters remain constant within the cloud environment and are specifically delineated by the respective cloud providers.

$$\begin{aligned} R_t &= F_t - Ar_t + T_d \\ R_t &= F_t - Ar_t + T_l + T_{tr} \\ R_t &= F_t - Ar_t + T_l + \frac{D}{Bw_{peruser}} \\ R_t &= F_t - Ar_t + T_l + \frac{D}{\frac{Bw_{total}}{N_t}} \\ R_t &= F_t - Ar_t + T_l + \frac{D * N_t}{Bw_{total}} \end{aligned} \tag{8}$$

Initially, we introduced R_t as the response time for a singular userbase. However, upon scrutiny of Table 2, it becomes evident that the chromosome structure may encompass n userbases. Given our overarching goal of minimizing the response time of the chromosome to expedite user feedback, the fitness function can be formulated as follows:

$$\begin{aligned} min(F_1) &= \sum_{i \leftarrow 1}^{n} R_t^i \\ min(F_1) &= \sum_{i \leftarrow 1}^{n} (F_t^i - Ar_t^i + T_l^i + \tfrac{D*N_t^i}{Bw_{total}}) \\ min(F_1) &= \sum_{i \leftarrow 1}^{n} F_t^i - \sum_{i \leftarrow 1}^{n} Ar_t^i + \sum_{i \leftarrow 1}^{n} T_l^i + \tfrac{D*n*\sum_{i \leftarrow 1}^{n} N_t^i}{n*Bw_{total}}) \end{aligned} \tag{9}$$

Data processing time

Data processing time in cloud environment reflects the time required to execute tasks, considering the dynamic allocation of resources, scheduling policies, and other factors within the simulated cloud infrastructure. Data processing time of a single DC D_t can be defined as

$$D_t = T_E + T_D + T_{Qe} \tag{10}$$

Here T_E denotes the task execution time, signifying the temporal duration required to fulfill a designated computational task. T_D indicates network latency, representing the temporal interval within which data traverses disparate components, encompassing transmissions between virtual machines or storage systems. Additionally, T_{Qe} characterizes the queuing delay inherent to each task, denoting the time incurred due to the task awaiting processing in the queue before its initiation.

Every DC processes many tasks (R_1, R_2,.... R_r). For a particular time, there are r number of tasks in the queue, and each task needs t_p for processing, then T_{Qe} will be

$$T_{Qe} = \sum_{j \leftarrow 1}^{r} t_p^j \tag{11}$$

$$D_t = T_E + T_D + \sum_{j \leftarrow 1}^{r} t_p^j \tag{12}$$

The task execution time required for task completion is a direct indicator of the efficacy of the scheduling scheme. With the advent of virtualization technology, wherein all virtual machines can execute tasks concurrently, determining the total execution time involves selecting the virtual machine with the lengthiest task execution duration [53]. The execution time T_E can be defined as

$$\begin{aligned} T_E &= \max_{m \leftarrow 1}^{M} Vm_{Time}^m \\ Vm_{Time}^m &= \sum_{t \leftarrow 1}^{N} \frac{T_t}{Vm_m} \\ T_E &= \max_{m \leftarrow 1}^{M} \sum_{t \leftarrow 1}^{N} \frac{T_t}{Vm_m} \end{aligned} \tag{13}$$

In the context of the given statement, the symbol Vm_{Time}^m signifies the completion time of the m-th VM after the execution of all tasks allocated to it. Herein, T_t denotes the t-th task assigned to a VM, and M and N respectively stand for the total number of VMs and tasks within the system.

$$D_t = \max_{m \leftarrow 1}^{M} \sum_{t \leftarrow 1}^{N} \frac{T_t}{Vm_m} + T_D + \sum_{j \leftarrow 1}^{r} t_p^j \tag{14}$$

We have denoted D_t as the data processing time of a single DC. But there can be d number of DCs (DC_1, DC_2... DC_d. So the fitness function can be formulated as following

$$\begin{aligned} min(F_2) &= \sum_{n \leftarrow 1}^{d} D_t^n \\ min(F_2) &= \sum_{n \leftarrow 1}^{d} (\max_{m \leftarrow 1}^{M} \sum_{t \leftarrow 1}^{N} \frac{T_t}{Vm_m} + T_D + \sum_{j \leftarrow 1}^{r} t_p^j) \\ min(F_2) &= \sum_{n \leftarrow 1}^{d} (\max_{m \leftarrow 1}^{M} \sum_{t \leftarrow 1}^{N} \frac{T_t}{Vm_m})^n + \sum_{n \leftarrow 1}^{d} (T_D)^n \\ &\quad + \sum_{n \leftarrow 1}^{d} (\sum_{j \leftarrow 1}^{r} t_p^j))^n \end{aligned} \tag{15}$$

Fitness function

The fitness function of our cloud service broker policy research is shown in 16:

$$\begin{cases} min(F_1) = \sum_{i \leftarrow 1}^{n} F_t^i - \sum_{i \leftarrow 1}^{n} Ar_t^i + \sum_{i \leftarrow 1}^{n} T_l^i + \frac{D*n*\sum_{i \leftarrow 1}^{n} N_l^i}{n*Bw_{total}}) \\ min(F_2) = \sum_{n \leftarrow 1}^{d} (\max_{m \leftarrow 1}^{M} \sum_{t \leftarrow 1}^{N} \frac{T_t}{Vm_m})^n + \sum_{n \leftarrow 1}^{d} (T_D)^n \\ \qquad + \sum_{n \leftarrow 1}^{d} (\sum_{j \leftarrow 1}^{r} t_p^j))^n \end{cases} \tag{16}$$

The dynamic nature of our fitness function arises from the temporal variability inherent in response time and data processing duration, both of which are subject to change over time. As a result our fitness calculation Algorithm 1 will provide different fitness scores based on the current configuration of the cloud.

Algorithm 1. 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 ← {};
D_t ← 0;
R_t ← 0;
for i ← 1 to chromosome.length do
    if dcSet not contains chromosome.at(i).DC then
        D_t ←
          D_t + calculateDCprocessingTime(chromosome.at(i).DC);
        dcSet.add(chromosome.at(i).DC)
    end
    R_t ← R_t + calculateUBRresponseTime(chromosome.at(i).UB)
end
return R_t, D_t
```

Proposed genetic algorithm

The GA is the most well-known and widely used evolutionary algorithms [54, 55]. It simulates biological evolution to tackle complex problems. Within a chromosome population, the GA seeks the optimal solution. It begins by initializing populations of chromosomes, employing fitness-based selection, crossover to generate new offsprings, and random mutation of these offspring. The GA identifies the optimal combination of DCs and userbases through these mechanisms. The process starts with population initialization, a step previously detailed during problem encoding. The GA incrementally guides the population toward the desired outcome with each generation. Our proposed GA based service broker policy is illustrated in Algorithm 2. "Figure 2" shows the flowchart of the GA used in our work. The following subsection will clearly elaborate the proposed GA in the service broker policy of cloud environment.

Algorithm 2. Proposed GA based service broker policy

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 ← Randomly Initialized including CDC results;
while maximumGeneration ≤ 1000 & fitness not achieved do
    firstParent, secondParent ← Selection(population);
    firstOffspring, secondOffspring ←
      Crossover(firstParent, secondParent);
    if randomPoint ≤ mutationRate then
        Mutation()
    end
    population.add(firstOffspring);
    population.add(secondOffspring);
    CalculateFitness(population) ;
end
dcSelection(population)
```

Selection

The selection process defines how individuals or chromosomes are chosen for crossover and mutation. Algorithm 3 depicts our population selection policy. In our work, we employed the CHC elitist selection policy [56]. Within CHC, offsprings fight for placement in the population alongside their parents and other offsprings [57, 58]. We have sorted the population based on their fitness values. The fittest individuals or chromosomes utilize crossover operations to generate even better individuals. Consequently, 50% of the sorted population is allocated for crossover. Conversely, less fit individuals undergo mutation operations to enhance their fitness [59]. "Figure 3" shows my population selection policy.

Algorithm 3. Selection algorithm of our propsosed GA

Input : population

Output : $firstParent \leftarrow$ Best Chromosome
$secondParent \leftarrow$ Second Best Chromosome

```
sortByFitness(population);
firstParent ← population.at(0);
secondParent ← population.at(1);
if currentPopulationSize ≥ initialPopulationSize then
    population ← population[1....(currentPopulationSize/2)]
end
```

Each population chromosome goes through crossover and mutation to create new offspring. Following GA operations, the population expands significantly over its initial size. Subsequently, the population is re-sorted, and the less fit half is eliminated, maintaining the original population size. This CHC selection

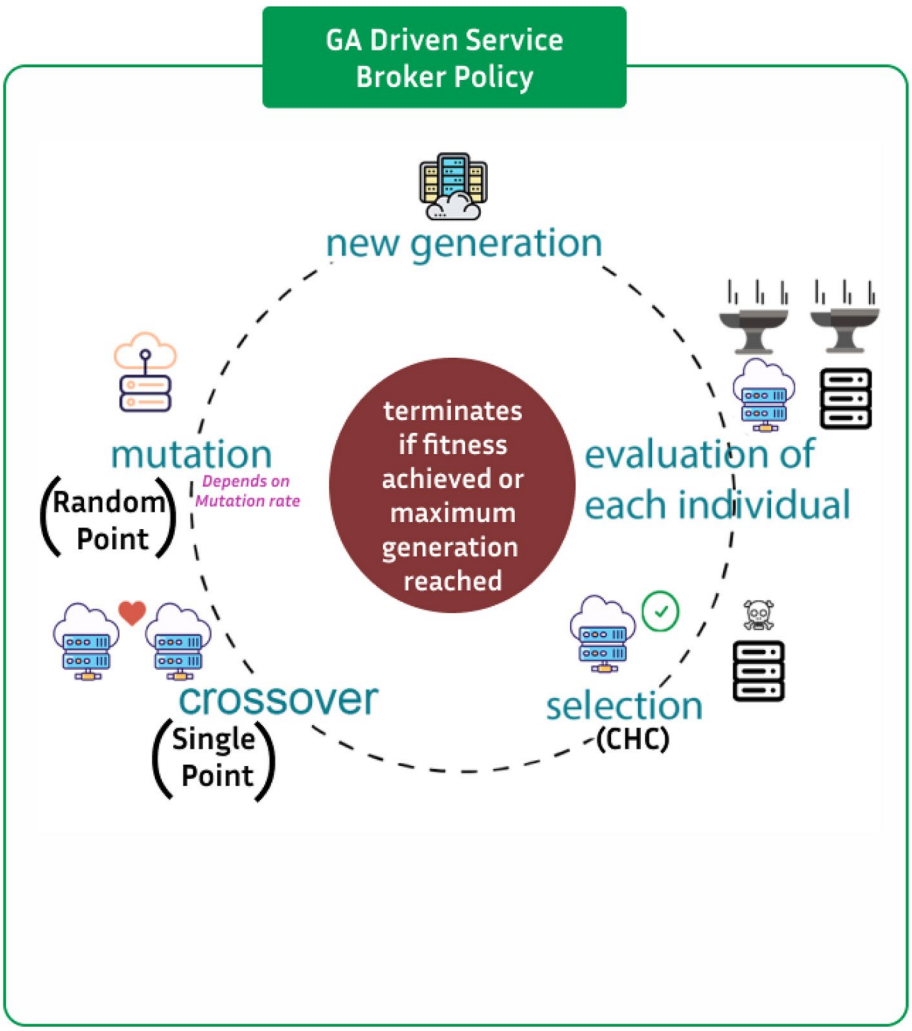

Fig. 2 Flow chart of genetic algorithm

method is notably elitist, preserving high-fitness individuals throughout the search process.

Crossover

The crossover process involves selecting genes from parent chromosomes to generate new offsprings, ensuring diversity within the population. In our GA, a single-point crossover method [57, 60, 61] is employed within the GA. This crossover operation happens on the best chromosomes, according to the selection mechanism detailed in the previous section.

Algorithm 4 represents this single-point crossover process within our GA. To elaborate, two parents are chosen as the best individuals in the population. A crossover point is then randomly determined between 0 and the chromosome length. For the first offspring, genes are combined before the first parent's crossover point and genes after the second parent's crossover point. Conversely, genes are combined for the second offspring after the first parent's crossover point and before the second parent's crossover point.

Algorithm 4. Crossover algorithm of our proposed GA

Input : $firstParent \leftarrow$ Best Chromosome
$secondParent \leftarrow$ Second Best Chromosome
Output : $firstOffspring \leftarrow$ First offspring after crossover
$secondOffspring \leftarrow$ Second offspring after crossover

```
crossoverPoint ← random(1, ubSize);
firstOffspring ← firstParent;
secondOffspring ← secondParent;
for c ← 1 to crossoverPoint do
    swap(firstOffspring.geneAt(c), secondOffspring.geneAt(c))
end
```

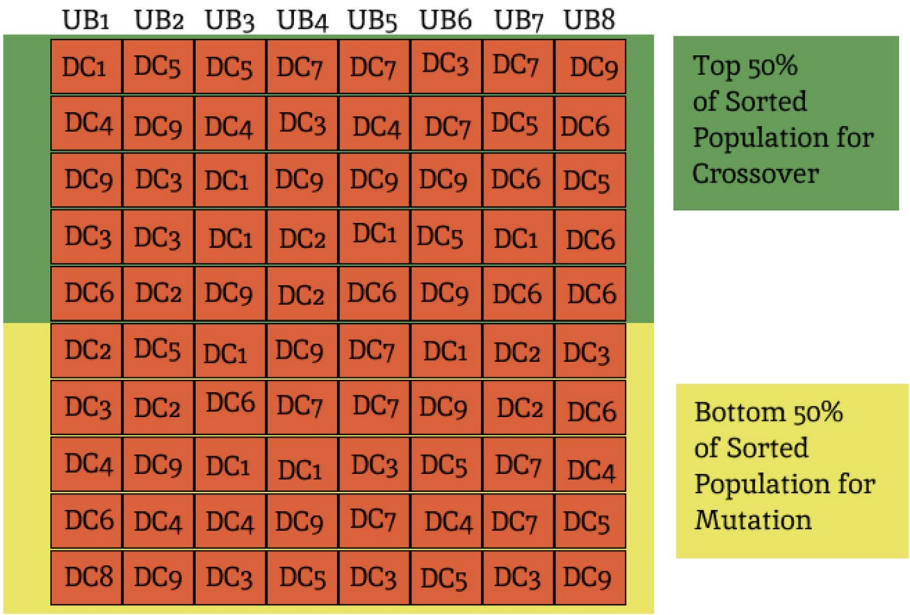

Fig. 3 Selection bases adopted search algorithms

"Figure 4" visually represents this single-point crossover process within the GA framework in our research. In "Fig. 4", the parents' genes from UB_4 to UB_8 are swapped to create new offsprings.

Mutation

After the crossover process in the GA, the next step involves mutation. Mutation introduces random alterations to the genes on the chromosomes. In our GA, we specifically target the less fit individuals for mutation. By minor changes to the less-fit individuals' genetic makeup, their overall fitness might improve, leading to a better solution [62, 63].

Algorithm 5 illustrates this mutation process within the GA. A random mutation point is selected within a range from 0 to the chromosome length during the mutation process. At this mutation point, a DC is randomly chosen and replaced with another DC. In "Fig. 5", a random userbase is selected (UB_4) from the chromosome. The allocated DC for the selected userbase is randomly changed with another DC.

Algorithm 5. Mutation algorithm of our propsosed GA

```
mutationIndividual ← random(populationSize/2, populationSize);
mutationPoint ← random(1, ubSize);
mutatedDC ← random(1, DCsize);
population.at(mutationIndividual).geneAt(mutationPoint).replaceDC(mutatedDC)
```

Termination

The termination condition determines the duration at which the algorithm halts. The GA goes through each generation's selection, crossover, and mutation processes. The choice of the number of generations significantly influences the GA's convergence. A limited number of generations can lead to premature convergence, trapping the GA in a local minimum and preventing the attainment of optimal results. Conversely, an excessive number of generations prolongs computation time, maximizing the algorithm's complexity. Therefore, determining an optimal number of generations is crucial for the GA's optimal performance. In our research, the GA algorithm halts when convergence is detected based on the fitness function or when a maximum number of generations is reached.

Following the termination of our proposed GA, Algorithm 6 was employed to identify the optimal DC selection for the service broker policy. The population was organized in descending order according to their respective fitness values. Subsequently, a comparison was conducted between the fitness of the top-performing chromosome in the population and the fitness associated with the preceding DC selection utilized in the service broker policy. If the highest-ranked individual in the population exhibited superior fitness, it was adopted for integration into the service broker policy. Conversely, if the existing DC selection demonstrated superior fitness, it was retained for continuation within the service broker policy.

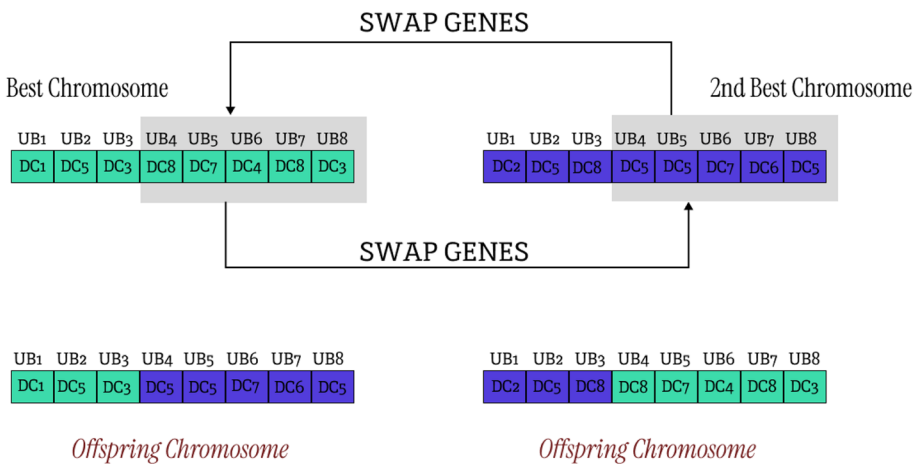

Fig. 4 Crossover of the chromosomes

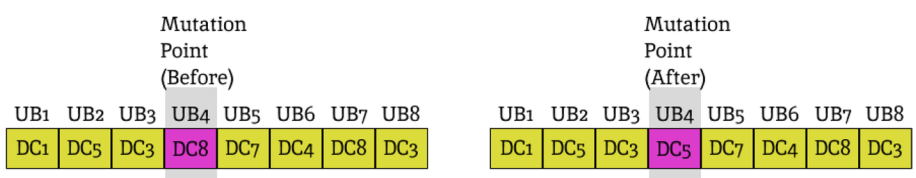

Fig. 5 Mutation of the chromosome

Algorithm 6. DC selection strategy of our proposed GA

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

Time complexity analysis

Let assume in our proposed GA the population size is n. The maximum generation the GA will run is g. The size of the chromosome used in our research is m. Maximum Crossover point in our research must be the smaller then the size of the chromosome which is $m-1$.

GA operations

- The time complexity for our selection Algorithm 3 will be

$$T_S = O(nlgn)[Sorting\ on\ Line\ 1] + O(1)[Line\ 2\text{-}5] \approx O(nlgn)$$
$$T_S = O(nlgn) \tag{17}$$

- The time complexity for our crossover Algorithm 4 will be

$$T_C = O(1)[Line\ 1\text{-}3] + O(m-1)[Line\ 4\text{-}6] = O(m-1)$$
$$T_C = O(m) \tag{18}$$

- The time complexity for our mutation Algorithm 5 will be

$$T_M = O(1) \tag{19}$$

DC selection strategy

The time complexity for our Optimal DC selection Algorithm 6 will be

$$T_{DC} = O(nlgn)[Sorting\ on\ Line\ 1] + O(1)[Line\ 2\text{-}6] \approx O(nlgn)$$
$$T_{DC} = O(nlgn) \tag{20}$$

Fitness calculation

The time complexity for our fitness calculation Algorithm 1 will be

$$T_{fit} = O(1)[Line\ 1\text{-}3] + O(m)[Line\ 4\text{-}10] + O(1)[Line\ 11] \approx O(m)$$
$$T_{fit} = O(m) \tag{21}$$

Time complexity of proposed GA

The total time complexity of our proposed GA can be defined as Eq. 22

$$T_{GA} = O(1) + g * [T_S + T_C + T_M + T_{fit}] + T_{DC} \tag{22}$$

Utilizing Eqs. 17, 18, 19, 21 and 20 we can calculate the overall time complexity of our algorithm as follows

$$\begin{aligned} T_{GA} &= O(1) + g * [O(nlgn) + O(m) + O(1) + O(m)] + O(nlgn) \\ T_{GA} &= g * [O(nlgn) + O(m)] + O(nlgn) \\ T_{GA} &= g * O(nlgn) + O(nlgn)[n \gg m\ so\ O(nlgn) > O(m)] \\ T_{GA} &= g * O(nlgn)[\text{Where } g \gg n] \end{aligned} \tag{23}$$

If the population size n increases, our proposed GA will need more generations to find optimal solutions, thus the maximum generation g will be increased. So, the overall time complexity is determined by the dominant term g. If we analyze the individual terms in Eq. 23

- $O(nlgn)$ This is a common time complexity for algorithms like merge sort, heap sort, or certain divide-and-conquer strategies. It means that the time to solve the problem increases slightly more than linearly as n grows.
- $g \gg n$ This means maximum generation g grows much faster than population size n. For example, if $g = n^2$ or even an exponential function like 2^n, then g quickly dominates the term $nlgn$.

When combining these two, if maximum generation g grows significantly faster than population size n, the time complexity will be dominated by maximum generation g, because the maximum generation g will overpower $nlgn$ even as n increases.

Real world cloud implementation of proposed GA

"Figure 6" illustrates the interaction between key components of the proposed GA-driven service broker policy in a real world cloud computing environment. The process begins with the userbase sending a service request to the CSB. This request represents the demand for cloud resources to process user tasks, such as accessing a web application. The CSB receives the request and runs the GA to optimize the DC selection. The GA randomly generates a population of potential solutions (DC Selection) including the DC selection from CDC policy to determine the most efficient DC for processing the request.

The GA processes the request using three key operations: Selection, Crossover, and Mutation. Selection identifies the best chromosomes from the population based on performance metrics (response time, processing time, cost). Crossover combines selected chromosomes to generate new offspring solutions. Mutation introduces random changes to explore new possibilities and avoid local optima. These steps are repeated over multiple iterations, refining the solutions with each generation. After evolving through several iterations, the GA determines the optimal DC selection. The CSB uses the optimal selection provided by the GA to direct the user's request to the most suitable DC.

The selected DC processes the task, executes the user's request, and sends the response back to the userbase. This concludes the sequence, with the user receiving the requested service efficiently. This diagram shows how the GA enhances the decision-making process of DC selection by continually evolving and optimizing configurations, leading to reduced response times, better resource utilization, and cost savings.

Environment setup

A suitable environment is required to evaluate the accuracy, efficacy, and efficiency of the suggested algorithm. To gauge the efficacy and accuracy of a service broker policy, researchers may evaluate the associated algorithms through two primary methodologies: employing commodity cloud infrastructure or utilizing specialized simulation tools. Amazon Web Services (AWS), Microsoft Azure, Google, IBM Cloud, Rackspace, Red Hat, Verizon Cloud, and VMware are examples of the numerous cloud computing service providers [64]. The end-user can access diverse re-configurable computational assets, including servers, storage facilities, networking infrastructure, and applications. Integrating the proposed service broker policy with these providers is contingent upon substantiating algorithmic accuracy within a cloud-based environment.

While conducting tests on authentic cloud infrastructure yields a higher degree of precision in comparison to simulation tools, it is not without its challenges. The inherent complexity of the cloud environment, coupled with uncontrollable factors such as network congestion and variations in DCs workloads, introduces potential impediments to the repeatability of test results. Moreover, it will be hard to conduct simulations on commodity cloud infrastructure as setting up such an environment is going to be expensive.

In light of these challenges, simulation tools emerge as a reliable alternative, offering the advantage of facilitating the parameterization of DC and network configurations. Consequently, to emulate authentic cloud scenarios

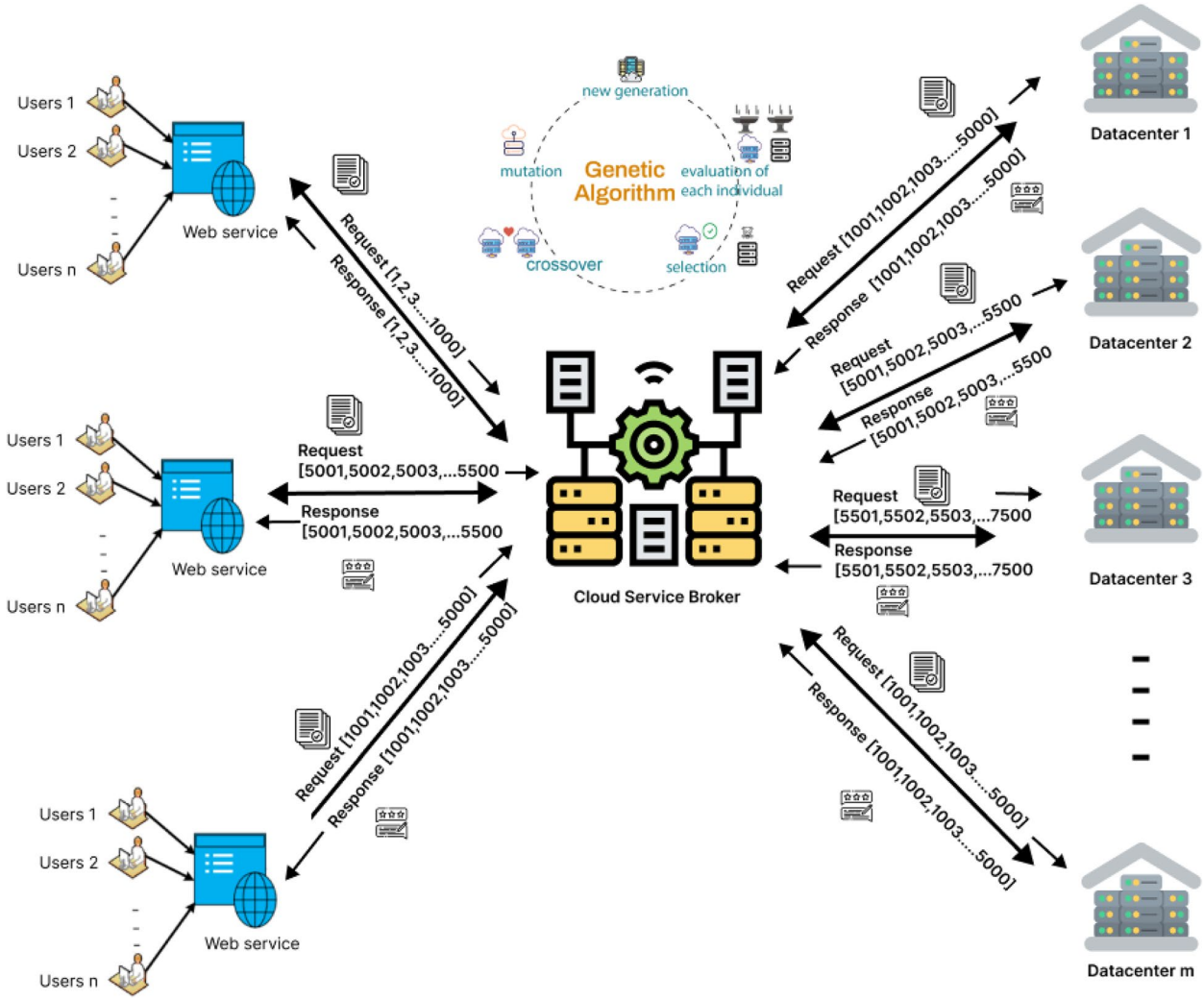

Fig. 6 Real world cloud implementation of proposed GA

and validate the efficacy of the algorithms, establishing a simulation environment is imperative. This enables researchers to control and manipulate various aspects of the testing environment and ensures the repeatability of simulation results. The usage of simulation tools such as CloudSim [65] or CloudAnalyst [66] is the most commonly employed approach by other researchers worldwide.

CloudSim stands out as a pioneering cloud simulation modeling platform, representing an early entrant in the domain [49, 52, 65]. CloudSim enables hosts, which DCs manage, to manage VMs. Since the platform needs a user interface, it is challenging to configure the system to work in real-world scenarios. Creating graphical reports is also tricky because the console displays the output results.

By offering users a graphical user interface to customize the cloud environment to meet their needs, the CloudAnalyst [66] platform overcomes these difficulties and widely used by the researchers [38–46]. CloudAnalyst provides an intuitive platform to easily simulate and analyze real-world cloud computing scenarios. By allowing users to model geographically distributed userbases and datacenters, it accurately reflects the global nature of cloud services. With configurable parameters such as network latency, traffic loads, and peak usage periods, it simulates real-world conditions like high-demand events or regional traffic surges. "Figure 7" displays the CloudAnalyst simulator's world view. Six geographical areas, designated as R_0, R_1, R_2, R_3, R_4, and R_5, make up the CloudAnalyst platform's world. On the map, the *red dots* on the globe stand

in for the DCs of cloud services, while the blue dots represent the userbase. The userbase, the DC, the load balancing policy, and the service broker policy are the four most critical vital components of the CloudAnalyst. We incorporated the proposed GA-based service broker policy in CloudAnalyst to implement the service broker policy.

Userbase configuration

A userbase is a group of users who generate continuous requests from a specific region. The userbase possesses several traits. Each userbase may have a similar or different number of requests per hour. For the userbase, peak hours are crucial. Peak hours are the period when the majority of users are online and the cloud is dealing with a huge volume of requests. Compared to the average number of users during off-peak hours, the average during peak hours is incredibly high. Tables 3, 4, and 5 show different userbase configurations used in our experiment. The configuration of the userbase is established by referencing the experimental setups employed by other researchers in the context of service broker policy [26, 27, 43].

DC configuration

The DC makes an effort to handle user requests effectively and efficiently. Every physical server in the DC network functions as the cloud's central processing unit for individual computers. VMs are an abstraction used by cloud computing over actual servers. The virtual machines' configuration can be identical or distinct to physical servers. All virtual machines' configurations must match that of the available physical servers. Creating more VMs is possible if there are resources or physical servers. The use of load-balancing algorithms divides the tasks among the VMs.

DC configuration is defined in Table 6.

In this research, twenty physical servers were employed in each DC [67–70]. The configuration details for each physical server are outlined in Table 7. Furthermore, each physical server is abstracted by utilizing VMs. For our experiments, 25 VMs were deployed, all possessing identical specifications, including 1024 MB of memory, a 10000 MB image size, and a 1000 Mbps bandwidth [67–70]. The DC and VM configurations are formulated by considering the experimental setups employed by other researchers in the context of load-balancing policy research [67–70].

Load balancing policy

Efficient load-balancing policies are pivotal in effectively managing user requests among VMs within a DC. A DC, comprising physical servers that may share uniform configurations or be individually customized, serves as the infrastructure in which cloud computing operations are conducted. VMs, employed to abstract the complexities

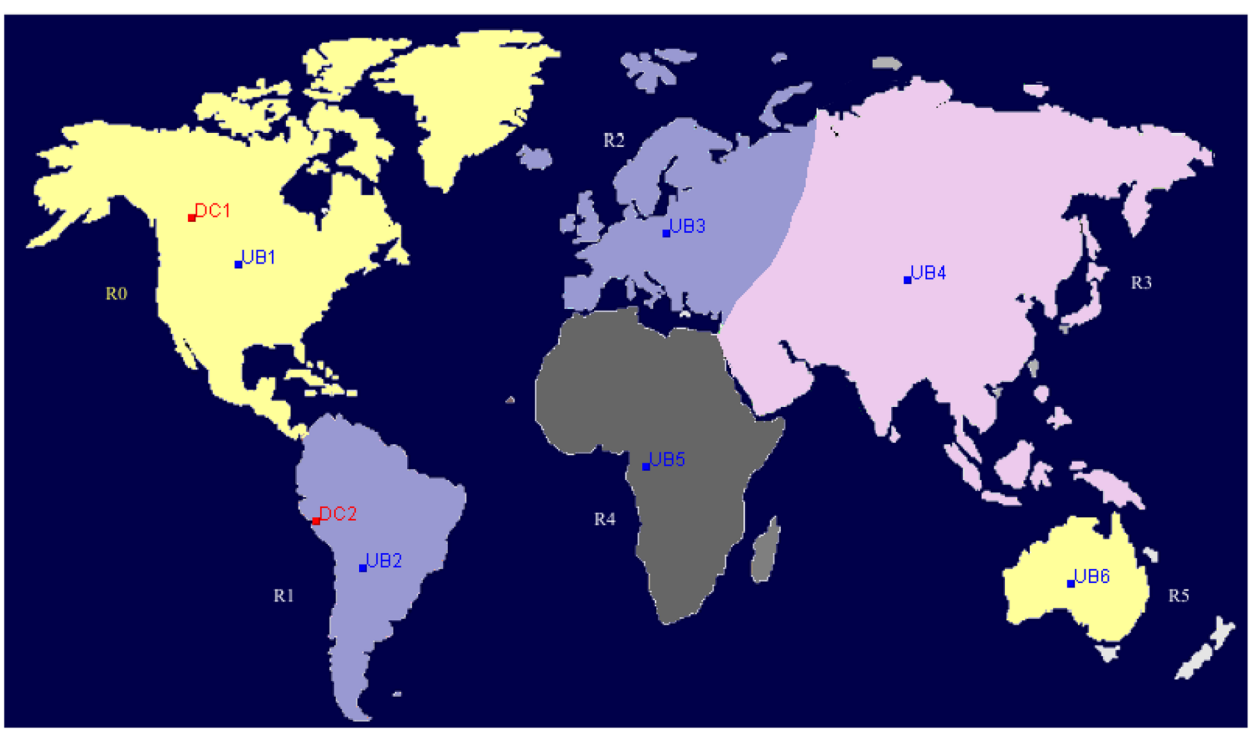

Fig. 7 CloudAnalyst simulator

Table 3 Userbase configuration 1 (UBC1)

Name	Region	Request per user per hr	Data size per request (Bytes)	Peak hours start (GMT)	Peak hours end (GMT)	Avg peak users	Avg Off-Peak Users
UB1,UB11,UB14,UB16,UB18,UB40	0	12,60,60, 60,60,60	100	13,3,3,3,3,3	15,9,9, 9,9,9	400000,10000,10000,10000,10000,1000	40000,1000,1000,1000,100,100
UB2,UB6,UB20,UB35,UB39	1	12,12,60,60,60	100	15,3,3,3,3	17,9,9,9,9	100000,80000,1000,1000,1000	10000,1000,1000,100,100
UB3,UB7,UB8,UB9,UB10, UB13,UB15,UB17,UB19,UB23,UB24	2	12,60,60,60,60,60,60,60,60,60	100	20,3,3,3,3,3,3,1,3,3,3	22,9,9,9,9,9,9,2,10,9,9	300000,10000,10000,10000,10000,10000,10000,1000,10000,1000,10000	30000,1000,1000,1000,1000,1000,1000,1000,1000,1000,1000
UB4,UB12,UB26,UB30,UB32,UB38	3	12,60,60,60,60,60	100	1,3,3,3,3,3	3,9,9,9,9,9	150000,10000,10000,10000,10000,1000	15000,1000,100,1000,1000,100
UB5,UB21,UB25,UB31,UB33,UB34	4	12,60,60,60,60,60	100	21,3,1,15,7,1	23,9,7,18,8,4	50000,1000,10000,10000,10000,10000	5000,1000,1000,1000,1000,1000
UB22,UB27,UB28,UB29,UB36,UB37,	5	60	100	3,15,3,9,3,3	9,19,9,12,9,9	10000,10000,10000,10000,1000,1000	1000,100,100,100,100,100

of DCs, execute tasks within the cloud computing environment. Dynamic workload distribution across VMs through load balancing is imperative to mitigate the risks of overload or under utilization.

Implementing a well-designed load-balancing policy is instrumental in optimizing the utilization of resources within the DC. Several load-balancing policies are available within cloud services, each designed to cater to specific operational needs. Examples include First Come-First Serve, Round Robin, Equally Spread Current Execution Load, and Throttled Load Balancing Policy [71–74].

In the context of load balancing, First Come First Serve assigns tasks to the first available VM until it reaches a state of overwork or unresponsiveness. Conversely, the round-robin load-balancing policy is exclusively utilized. This policy distributes tasks among VMs in a sequential, cyclical manner. Equally, Spread Current Execution Load ensures an even distribution of tasks among VMs. The Throttled Load Balancing Policy identifies active or available VMs for task assignment.

The round-robin load-balancing policy was selected as the most suitable algorithm for our experimental investigations. This strategic choice aligns with our research objectives and provides a consistent basis for evaluating performance metrics in a controlled environment.

Integration of proposed service broker policy in CloudAnalyst

Cloud environment is a multifaceted system characterized by attributes such as the userbase, DCs, load balancing policies, and service broker policies. "Figure 8" illustrates the workflow of the proposed service broker policy within CloudAnalyst. Cloud components exhibit various characteristics, including data transfer costs, request sizes during peak and off-peak hours, latency, bandwidth, and the number of requests.

The service broker policy is responsible for selecting the appropriate DC for the userbase based on these cloud attributes. Initially, the user configures the environment according to their specific requirements, such as the number of DCs needed for their requests, request sizes, peak and off-peak hour timings, and the number of requests during these periods. This configuration encompasses the DC settings and the load balancing policy.

Subsequently, the user requests and DC configurations are integrated into the service broker policy. The service broker policy then determines the optimal DC selection for a particular userbase at a given time, based on the provided inputs. Our proposed GA will be employed in service broker policy to identify the optimal DC for a specific userbase at any given moment.

Experimental scenarios

We analyzed to verify the accuracy of our algorithm using real-world scenarios within a simulation environment. Our userbase and DC configurations represent scaled-down versions of real-world scenarios. For instance, in the userbase configuration outlined in Table 5, the total of average users during peak hours is 1,370,000, which accounts for only 0.05% of the actual number of Facebook users (3.05 billion) as of 2023 [75]. Additionally, the DC configuration mirrors the free-tier setup of Amazon Elastic Compute Cloud (AWS EC2) [76]. Furthermore, we have taken inspiration from the previous work done in this field to design the experiment scenarios. Considering all these factors we have used five cloud configurations to examine the correctness of our proposed algorithm [67–70]. Table 8 shows the cloud configurations used for the experiments.

Result analysis

Tuning of genetic algorithm operators

Population size

An appropriate number of individuals helps to get optimal solutions rapidly. Large population size can result in increased computational time. On the other

Table 4 Userbase configuration 2 (UBC2)

Name	Region	Request per user per hr	Data size per request (Bytes)	Peak hours start (GMT)	Peak hours end (GMT)	Avg peak users	Avg off-peak users
UB1,UB11,UB14,UB40,UB16,UB18,UB46	0	12,60,60,60,60,60,60	100	13,3,3,13,3,4,6	15,9,9,21,9,9,24	400000, 10000, 10000, 10000, 10000, 10000, 10000	40000, 1000, 1000, 1000, 1000, 1010, 1000
UB2,UB6,UB35,UB39,UB20,UB41	1	12,12,60,60,60,60	100	15,9,3,7,3,10	17,11,23,15,9,15	100000, 80000, 10000, 10000, 10000, 10000	10000, 8000, 1000, 1000, 1000, 1000
UB23,UB24,UB3,UB7,UB8,UB9,UB10,UB13,UB15,UB17,UB19,UB42	2	60,60,12,60,60,60,60, 60,60,60,60,60	100	3,3,20,3,7,3,1,3,3,1,3,14	9,9,22,9,9,6,9,9,9,2,10,17	10000, 10000, 300000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000	1000, 1000, 30000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000
UB4,UB26,UB30,UB32,UB12,UB38,UB43	3	12,60,60,60,60,60,60	100	1,3,3,3,5,1,5	3,9,9,9,9,5,12	150000, 10000, 10000, 10000, 10000, 10000, 10000	15000, 100, 1000, 1000, 1000, 1000, 1000
UB21,UB25,UB5, UB31,UB33,UB34, UB44	4	60,60,12, 60,60,60, 60	100	5,1,21, 15,7,1, 7	9,7,23, 18,8,4, 19	10000, 10000, 50000, 10000, 10000, 10000, 10000	1000, 1000, 5000, 1000, 1000, 1000, 1000
UB22,UB27,UB28, UB29,UB36,UB37, UB45	5	60	100	23,15,3,9,5,8,3	24,19,9,12,9,19,9	10000	1000, 100, 100, 100, 1000, 1000, 1000

hand, a small population size will not lead to a proper exploration of diversity. We have tested our algorithm with various population sizes. Table 9 shows the response time with different population sizes. "Figure 9" compares the response time for different population sizes. The empirical analysis conducted in this study, as depicted in "Fig. 9", underscores the efficacy of our proposed GA-based service broker policy in achieving superior stability in response times. Specifically, it is evident that a population size of 50 yields a notably stabilized performance, unlike alternative population sizes, which failed to demonstrate consistent stability across multiple simulation runs. In conclusion, the results presented in Table 9 and "Fig. 9", affirm the merit of our proposed GA-based service broker policy, with a population size of 50 emerging as the optimal configuration for achieving stabilized response times.

Mutation operator

Since mutation involves a random alteration in the gene, it shouldn't be applied as a step every time in the genetic

Table 5 Userbase configuration 3 (UBC3)

Name	Region	Request per user per hr	Data size per request (Bytes)	Peak hours start (GMT)	Peak hours end (GMT)	Avg peak users	Avg off-peak users
UB1,UB11,UB14,UB16,UB18,UB40	0	12,60,60,60,60,60	100	13,3,3,3,4,3	15,9,9,9,9,9	400000, 10000, 10000, 10000, 10000, 10000	40000, 1000, 1000, 1000, 1010, 1000
UB2,UB6,UB35,UB39,UB20	1	12,12,60,60,60	100	15,9,3,3,3	17,11,9,9,9	100000, 80000, 10000, 10000, 10000	10000, 8000, 1000, 1000, 1000
UB3,UB23,UB24,UB7,UB8,UB9,UB10,UB13,UB15,UB17,UB19	2	12,60,60,60,60,60,60,60,60,60,60	100	20,3,3,3,7,3,1,3,3,1,3	22,9,9,9,9,6,9,9,9,2,10	300000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000	30000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000
UB4,UB26,UB30,UB32,UB12,UB38	3	12,60,60,60,60,60	100	1,3,3,5,3,3	3,9,9,9,9,9	150000, 10000, 10000, 10000, 10000, 10000	15000, 100, 1000, 1000, 1000, 1000
UB5,UB21,UB25,UB31,UB33,UB34	4	12,60,60,60,60,60	100	21,5,1,15,7,1	23,9,7,18,8,4	50000, 10000, 10000, 10000, 10000, 10000	5000, 1000, 1000, 1000, 1000, 1000
UB22,UB27,UB28,UB29,UB36,UB37	5	60	100	23,15,3,9,3,3	24,19,9,12,9,9	10000	1000, 100, 100, 100, 1000, 1000

Table 6 DC configuration

Region	Architecture	Operating system	Virtual machine monitor	Cost per VM hour	Cost per Mb memory hour	Storage cost per Gb	Data transfer cost per Gb	No of server
0-5	eX86	Linux	XEN	0.1$	0.05$	0.1$	0.1$	20

Table 7 Physical server configuration

Memory	Storage	Available bandwidth	Number of processor	Processor speed	Virtual machine policy
2048 MB (2GB)	100000 MB	10000	4	100	Time Shared/ Space Shared

algorithm. Therefore, a probabilistic approach is chosen to perform mutation in the genetic algorithm. In our research, we developed a probabilistic approach to find when modification will occur in the GA by generating numbers from 0 to 7. We then compared these numbers to specific contents, varying the value of x to explore different probabilities. Additionally, the likelihood of selecting a less-fit chromosome (50% of the population) is 1/25. We have experimented with different values of x from 1 to 7, which provided a mutation rate varying from 0.006 to 0.04. Table 10 shows the average response time for these various mutation rates. The results show that our GA's most suitable mutation rate was 0.028, which is very close to 0.03, commonly used in evolutionary algorithms [77–79].

Maximum generations

We have executed my algorithm several times with different configurations for 24 hours. Our algorithm is implemented every hour to obtain the optimal DC configuration for the hour. Due to the randomness of the evolutionary algorithms, it may take different generations to reach the optimal solution. Table 11 shows the different number of generations needed every hour for multiple simulation runs using the same configuration. We can see that the maximum number of generations required for any hour is 879. So, We have used 1000 generations as the maximum number of generations. Again, if the algorithm halts after maximum generations, We compared the best chromosomes from the algorithm with the previous best chromosomes used in the service broker policy. The best between those two will progress to the service broker policy.

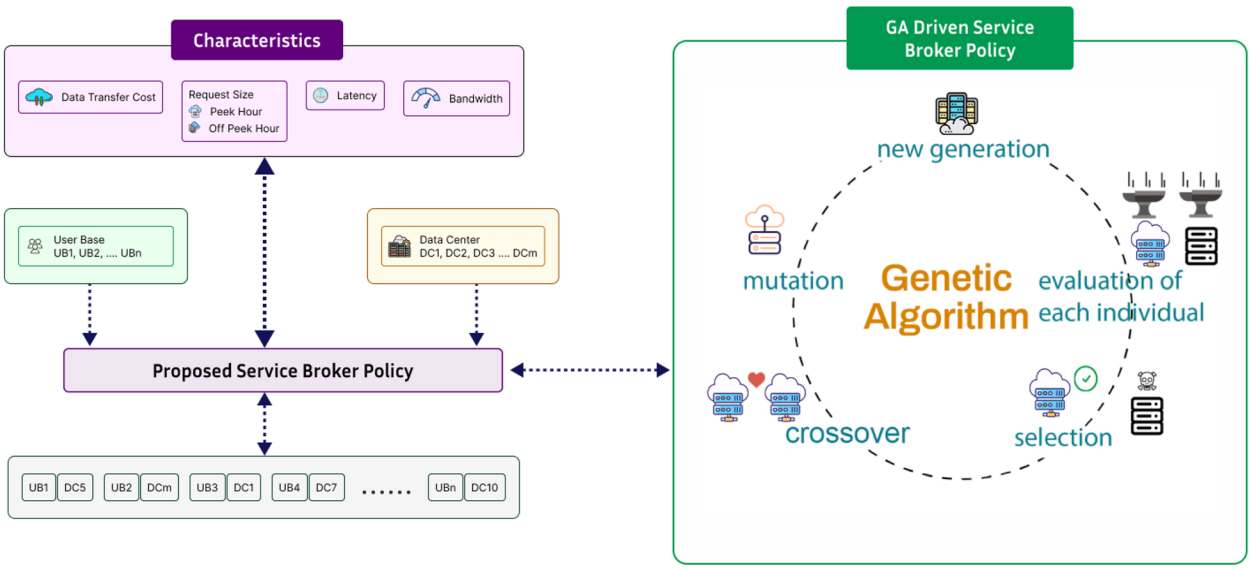

Fig. 8 Workflow of CloudAnalyst with GA based service broker policy

Table 8 Cloud configurations

Configuration name	Userbase configuration	DC configuration
Cloud Configuration 1 (CC1)	UBC1	4 DCs, 2 in region 0, 1 in region 2 and 1 in region 3
Cloud Configuration 2 (CC2)	UBC3	4 DCs, 2 in region 0, 1 in region 2 and 1 in region 3
Cloud Configuration 3 (CC3)	UBC2	4 DCs, 1 in region 0, 1 in region 2, 1 in region 4 and 1 in region 3
Cloud Configuration 4 (CC4)	UBC2	4 DCs, 2 in region 0, 1 in region 2 and 1 in region 3
Cloud Configuration 5 (CC5)	UBC1	4 DCs, 1 in region 0, 1 in region 2, 1 in region 4 and 1 in region 3

Table 9 Response time (milliseconds) for different population

Population	Response time in simulation 1	Response time in simulation 2	Response time in simulation 3	Response time in simulation 4	Standard deviation of response time
40	1881.06	1181.5	2238.21	1577.26	449.013
50	800.58	756.83	827.96	634.78	85.356
75	2689.16	3844.16	1391.14	1033.25	1284.767
100	1364.81	2301.38	2486.81	1483.58	567.1423

Optimal parameters for GA

The optimal parameter value for the GA has been discerned by examining the experimental outcomes presented in the preceding sections. The following delineates the most fitting values assigned to the GA parameters, strategically defined to achieve optimal results in the conducted experiments.

$$\begin{cases} Population = 50 \\ Maximum\ Generation = 1000 \\ Mutation\ Rate\ = 0.028 \end{cases} \tag{24}$$

Genetic algorithm convergence analysis

The fitness function employed in our research is characterized as dynamic, reflecting temporal variations. Notably, response and data processing times exhibit hourly fluctuations due to user activity disparities between peak and non-peak hours. The surge in user numbers during peak hours results in a substantial influx of user requests, consequently leading to congestion in DCs. The impact on the algorithm's response and data processing times becomes apparent when multiple userbases are allocated to a single DC.

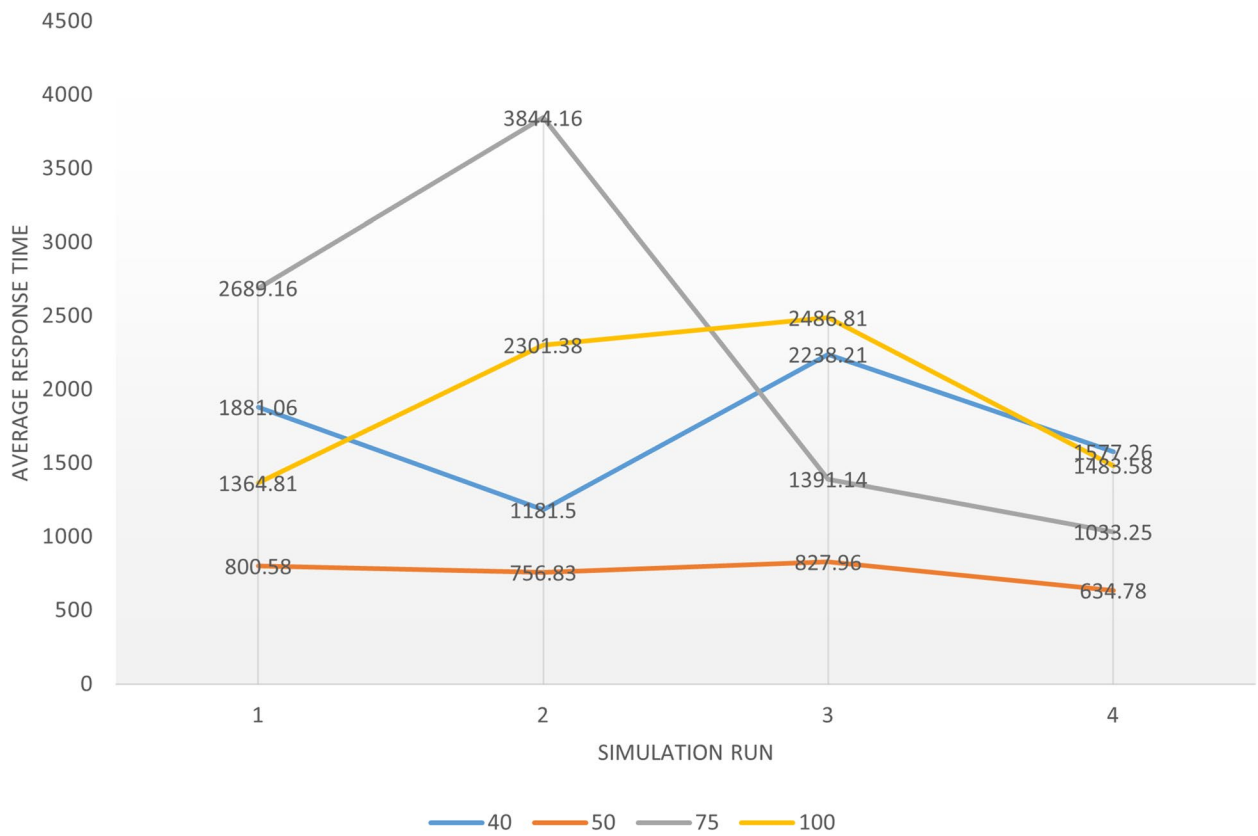

Fig. 9 Average response time (milliseconds) for different population size

Table 10 Response time (milliseconds) for different population

Mutation rate	Simulation 1	Simulation 2	Simulation 3	Simulation 4	Standard deviation of response time
0.006	718.24	878.4	1220.89	1578.02	382.0819369
0.011	1222.51	1150.92	1282.53	2413.28	599.7312068
0.017	2135.39	1646.69	1972.42	1656.84	241.5558606
0.023	3047.16	1277.94	1374.48	2542.19	873.4122342
0.028	800.58	756.83	827.96	634.78	85.35590133
0.034	1741.88	1580.35	2593.43	1705.97	463.8799767
0.04	1083.61	1956.72	1535.32	1517.83	356.5363816

The graphical representation in "Fig. 10" elucidates the convergence patterns observed during the 2nd, 7th, 14th, and 21st hours of the day for the proposed GA. The primary objective of the dynamic fitness function is the minimization of the fitness value, with the ideal fitness value being zero. However, given the continuous influx of tasks to the cloud and the perpetual engagement of the DC with user requests, achieving zero or negative response and data processing times is unattainable.

The termination criterion for the proposed GA is predicated on detecting convergence in the fitness value or reaching the predetermined maximum number of generations. "Figure 10" underscores the distinctiveness of convergence graphs during various hours of the day, thereby substantiating the assertion of a dynamic fitness function. This adaptability is crucial for addressing the varying demands imposed on the cloud environment throughout different temporal intervals.

Table 11 Generation needed in every hour for GA Generation needed in every hour for GA

Hours	Generation in run 1	Generation in run 2	Generation in run 3	Generation in run 4	Generation in run 4	Generation in run 5	Maximum generation needed each hour
0	411	226	211	34	35	35	411
1	14	120	499	123	147	142	499
2	27	49	320	173	182	546	546
3	19	26	499	434	457	36	499
4	61	18	499	23	264	61	499
5	88	22	101	184	188	286	286
6	41	56	499	46	407	250	499
7	111	27	87	654	477	163	654
8	48	19	203	162	524	288	524
9	12	18	186	411	58	224	411
10	26	30	499	342	427	342	499
11	43	69	249	451	210	279	451
12	23	43	499	487	36	355	499
13	28	63	309	69	125	435	435
14	14	10	403	147	419	579	579
15	33	10	318	535	351	317	535
16	26	10	76	74	461	545	545
17	16	137	287	96	477	105	477
18	82	38	177	20	457	87	457
19	23	11	20	199	263	217	263
20	22	32	433	212	53	69	433
21	46	18	109	264	879	38	879
22	9	41	303	55	135	121	303
23	9	10	128	453	152	168	453
24	42	40	178	496	20	83	496
Max Generation Needed					879		

Execution time of genetic algorithm

The GA-based service broker policy manifests a discernibly high time complexity, resulting in extended execution times that pose potential challenges to the overall efficiency of the cloud environment with each incoming request. Given these considerations, a reasonable approach is critical when initiating the GA for optimization to reduce its impact. Initially, we orchestrated the proposed GA to execute at the onset of each hour throughout the day. The resultant optimal solution is persistently stored, and after the completion of each hour, the GA is re-invoked to explore additional optimal outcomes. If the present optimal result surpasses its predecessor, it supplants the former outcome; otherwise, the prior optimal result is retained.

To evaluate the repercussions of selectively suspending the GA at various intervals, we conducted tests involving halting its execution at the 4th, 8th, 12th, 16th, 20th, 24th, and 48th hours. After each interruption, a meticulous performance comparison was conducted between our GA and conventional algorithms, precisely the ORT, as delineated in Table 12. Notably, the results of the CDC algorithm were excluded from the analysis, as indicated in Table 8, highlighting its inability to surpass any predefined configurations within our cloud environment setups.

"Figure 11" provides a visual representation that assesses the performance dynamics of various algorithms across distinct time intervals. Initially, ORT exhibited superior performance compared to GA. However, as time elapses, the GA progressively converges and ultimately surpasses the performance of the ORT algorithm. The GA demonstrates a substantial performance advantage over the ORT algorithm under identical userbase request configurations.

It is imperative to recognize that the cloud environment continuously receives tasks characterized by evolving user requirements, including variations in request size, peak hours, and the overall volume of user requests. Based on our simulation results, the strategic

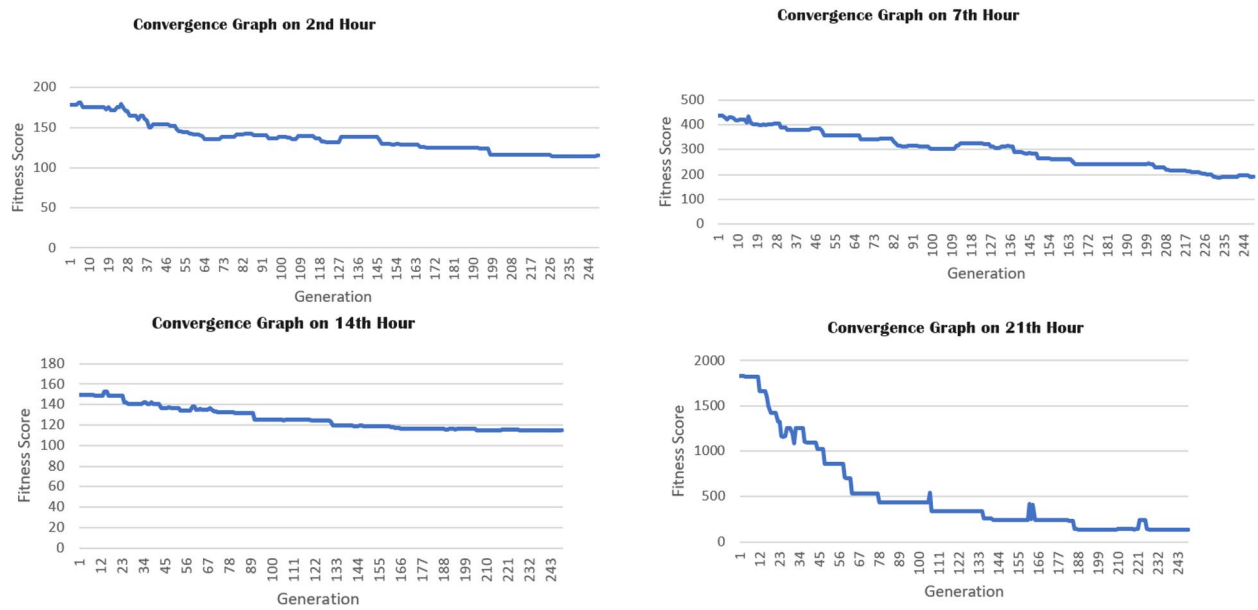

Fig. 10 Temporal variations in the convergence patterns of genetic algorithms

deployment of the GA at hourly intervals emerges as a more productive approach. This periodic execution pattern enhances the adaptability of the GA to the changing parameters of the environment, facilitating a continuous search for optimal solutions amid the dynamic nature of the cloud environment.

Performance comparison varying the userbases

The average response and data processing times for the proposed GA and traditional service broker algorithms in various configurations are compared in Tables 13 and 14. The findings indicate that the CDC policy is the worst of all the service broker policies. The CDC policy selects the userbase's closest DC. Consequently, the closest DC becomes overrun with userbases from the same area. Tables 13 and 14 show that ORT policy chooses the DC with the best response time and produces better results than CDC. This also crowds the DC with the fastest response time.

Tables 13 and 14 show the outcomes of the suggested GA. Our suggested GA performs better than the established algorithms (CDC, ORT). The suggested GA offers the userbase the best option. "Figures 12" and "13" demonstrate that the proposed GA performs substantially better than the CDC. For most experiments, the proposed GA performs generally better than ORT. Because GA is an algorithm based on evolutionary processes, it looks for global optimums. The best response time or the closest distance from the DC to the userbase are two examples of local optimum algorithms where the service broker policies can be stuck.

Table 12 Average response time (milliseconds) of GA and ORT throughout two days

Time	GA	ORT
4	582.77	407.97
8	449.97	380.77
12	500.41	397.35
16	763.22	723.43
20	758.25	723.9
24	1083.8	1118.24
48	852.5	922.85

However, the proposed GA-based service broker policy can explore various search spaces and locate a global optimum with the GA mutation steps. Thanks to crossover, the GA can produce better offspring/solutions from better parents/previous solutions.

The time needed to process the service request by a DC is called DC Request Servicing Times. Table 15 shows the DC request servicing time of ORT and GA service broker policy for each DC. From Table 15, we can say that for DC1 and DC2, the GA-based service broker policy provides better DC request servicing time than ORT. On the other hand, from the first look, for DC3 and DC4, the situation is the opposite. As a result, we have computed the improvement or deterioration of the GA-based service broker policy for each DC concerning ORT. Table 16 shows the individual DCs improvement or deterioration of DC service requesting time compared to ORT. From Table 16, we see that the DC request servicing time is

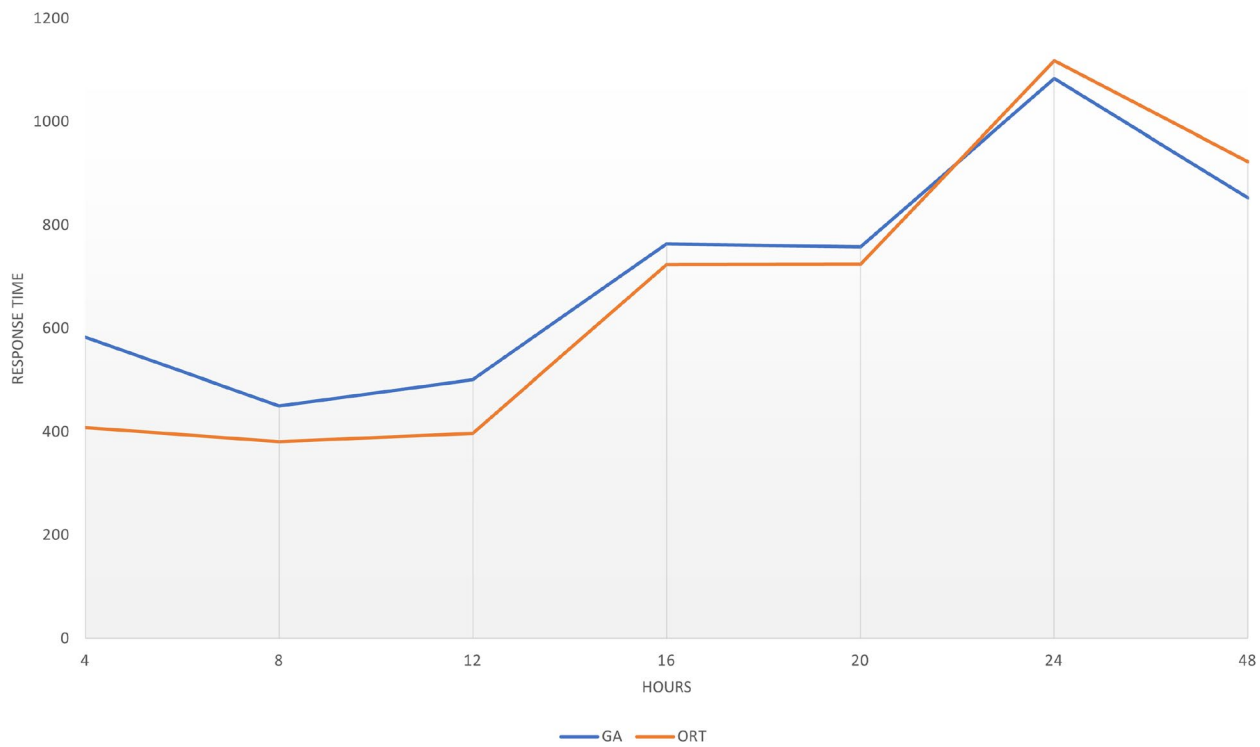

Fig. 11 Comparative analysis of performance through varied termination times

Table 13 Average response time (milliseconds) for different service broker policies

Configuration name	CDC	ORT	GA
CC1	2752.09	2719.48	1964.1
CC2	4942.96	2500.77	1406.27
CC3	3901.81	1667.57	1248.74
CC4	2429.67	882.01	871.72
CC5	2419.39	812.23	741.27

Table 14 Average data processing time (milliseconds) for service broker policies

Configuration name	CDC	ORT	GA
CC1	2343.7	2151.2	1574.66
CC2	4205.59	1977.13	951.24
CC3	3514.47	1356.07	764.71
CC4	2108.89	662.2	513.38
CC5	2100.54	601.34	518.01

improved for DC1 and DC2 and deteriorated for DC3 and DC4. We have plotted the results of Table 15 in the graph. "Figure 14" shows us the results of the improvement and deterioration of each DC. "Figure 14" shows that the worsening of DC3 and DC4 is negligible compared to the improvement in DC1 and DC2. Moreover, the further deterioration of DC3 and DC4 happened because they are more used in GA-based service broker policy than ORT. This ensures our GA-based service broker policy maintains more resource utilization than ORT. As a result, some DCs might yield more request servicing time; however, the overall request servicing time will be better than ORT. Table 14 and "Fig. 13" prove that the overall request servicing time, also called the data processing, is better.

Performance comparison varying the number of datacenters

Tables 17 and 18 present a comparative analysis of the average response and data processing times for both the proposed GA and baseline service broker policies across different DC configurations with UBC3 as userbase configuration shown in Table 5. From Table 5, the sum of the average users in peak hours is 1370000, which is only 0.05% of the real Facebook users (3.05 billion) as of 2023 [75]. The DC configuration reflects the free-tier configuration of Amazon Elastic Compute Cloud (AWS EC2) [76]. As expected, the findings reveal that an increase in DCs correlates with enhanced performance, whereas a reduction in DCs leads to diminished performance. It is noteworthy that expanding the quantity of DCs could

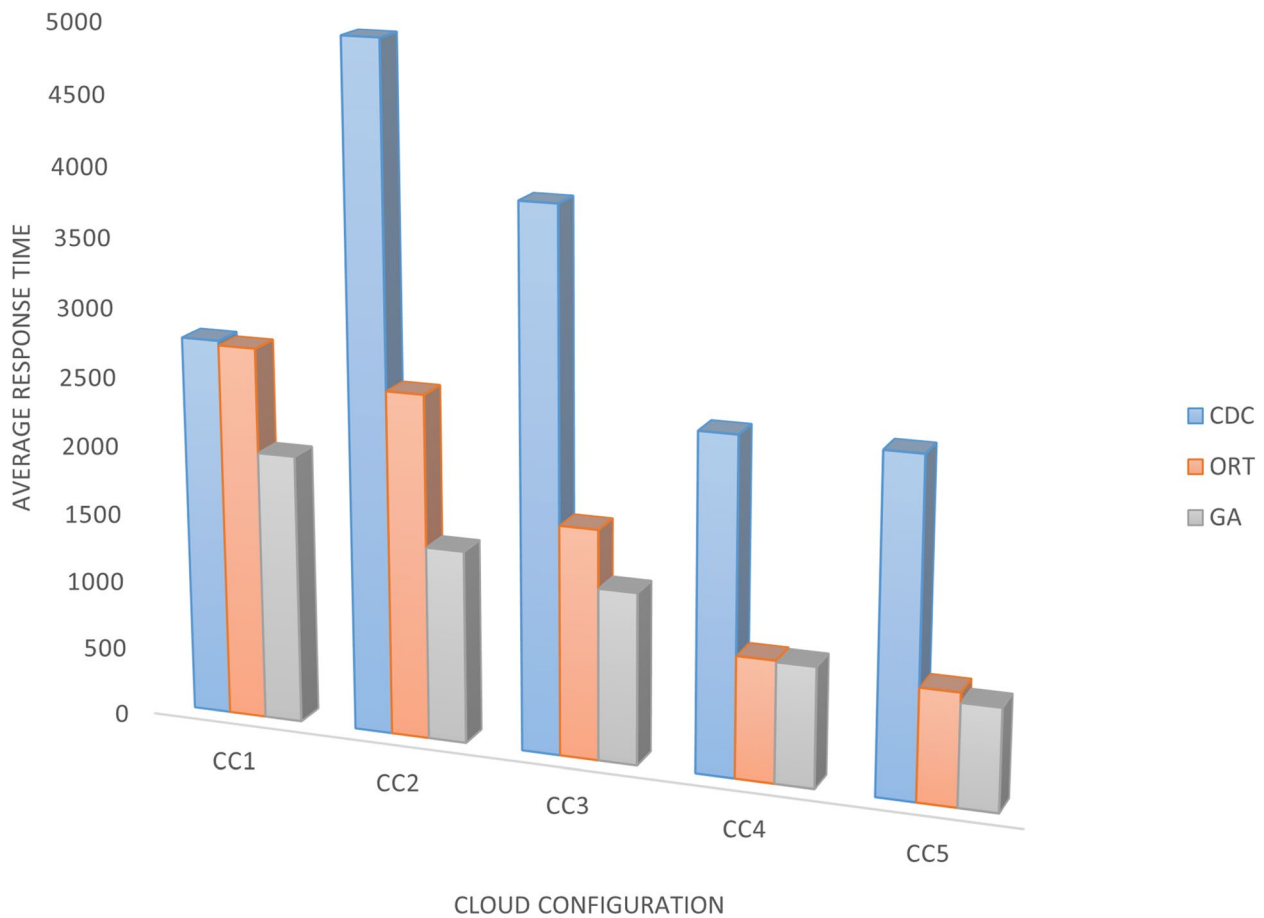

Fig. 12 Comparative analysis of response time between genetic algorithms and traditional algorithms

potentially increase the overall expenditure within the cloud environment.

As illustrated in "Figs. 15" and "16", our research outcomes showcase the response and data processing times. "Figures 15" and "16" demonstrate that the GA-based service broker policy outperforms the ORT and CDC approaches. The proposed GA-based service broker policy exhibits nearly identical response times, with only marginal differences compared to ORT, and superior response times compared to CDC, especially when utilizing fewer DCs. Additionally, the GA-based service broker policy proves to be more efficient in terms of data processing time when compared to both CDC and ORT, particularly in scenarios with fewer DCs.

The augmentation of DCs may result in an escalation of the overall cost within the cloud environment. This relationship between the number of DCs and the associated costs is elucidated in Table 19, where we systematically vary the number of DCs in our experiments. As illustrated in "Fig. 17", a discernible trend emerges, indicating that a reasonable reduction in DCs can effectively mitigate the expenses incurred within the cloud environment. Conversely, our proposed GA-based service broker policy strategy has demonstrated comparable levels of performance even when employing a reduced number of DCs, as shown in "Figs. 15" and "16". This highlights how our proposed service broker policy effectively ensures optimal resource utilization within the DCs and thus reduces the necessity of building new DCs. Notably, the number of DCs reduction is intricately linked to a tangible reduction in costs within the cloud environment. Furthermore optimizing the DC selection, there will have a significant positive impact on the environment, as it will help reduce the emission of global warming gases. Our strategic DC selection can lead to more energy-efficient operations, contributing to the global effort to mitigate climate change. In conclusion, our proposed GA-based service broker not only achieves superior performance but also does so at a reduced cost.

Moreover, it is noteworthy that our userbase configuration constitutes a mere 0.05% of the overall Facebook userbase [75]. In contrast, each DC configuration aligns

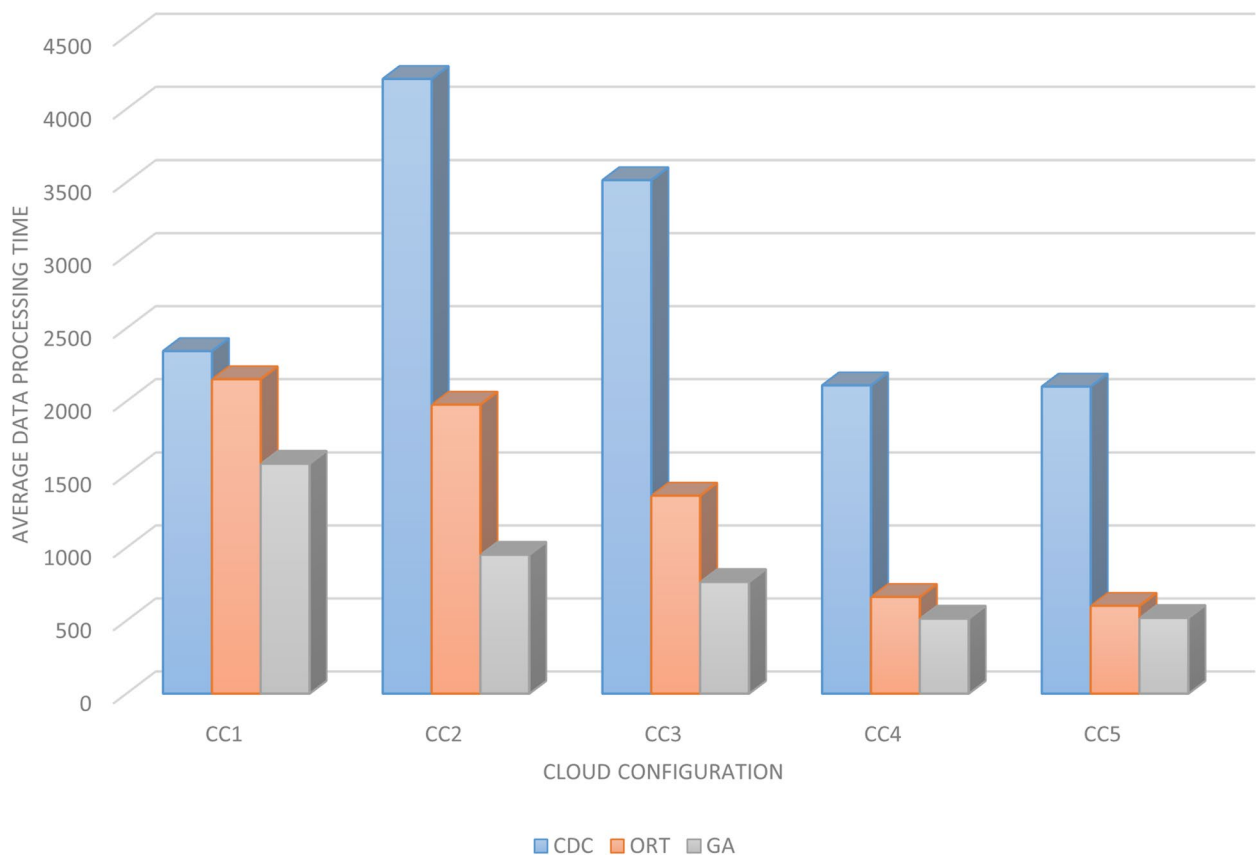

Fig. 13 Comparative analysis of data processing time (milliseconds) between genetic algorithm and traditional algorithms

Table 15 Datacenter request servicing times (milliseconds) for each DC

Cloud configuration	DC1		DC2		DC3		DC4	
	ORT	GA	ORT	GA	ORT	GA	ORT	GA
CC1	48710.51	45181.08	49691.03	40804.25	174.64	289.6	507.69	2953.28
CC2	46283.08	22747.09	33151.03	13974.83	174.34	406.41	570.36	2349.4
CC3	20270.11	15093.09	29316.58	7046.62	173.76	329.95	2265.03	2982.64
CC4	17344.01	14333.09	31204.29	6590.79	175.54	334.35	2202.3	3652.73
CC5	45622.1	20245.01	28433.79	11311.51	175.54	398.46	533.71	2336.69

Table 16 Improvement or deterioration of datacenter request servicing times (milliseconds) for each datacenter

Cloud configuration	DC1	DC2	DC3	DC4
CC1	3529.43	8886.78	-114.96	-2445.59
CC2	23535.99	19176.2	-232.07	-1779.04
CC3	5177.02	22269.96	-156.19	-717.61
CC4	3010.92	24613.5	-158.81	-1450.43
CC5	25377.09	17122.28	-222.92	-1802.98
Overall Result	**Improved**	**Improved**	**Deteriorated**	**Deteriorated**

with the Amazon EC2 free tier [76]. Remarkably, our GA-based approach results in a cost savings of 60 USD without compromising performance. This observation leads us to assert that adopting our proposed GA-based service broker methodology in expansive cloud service infrastructures holds the potential for financial gains while concurrently ensuring the sustained efficacy of the cloud environment.

In Table 20, we conducted an analysis of individual userbase response times using GA and ORT for varying DC configurations. The study involved 46 distinct

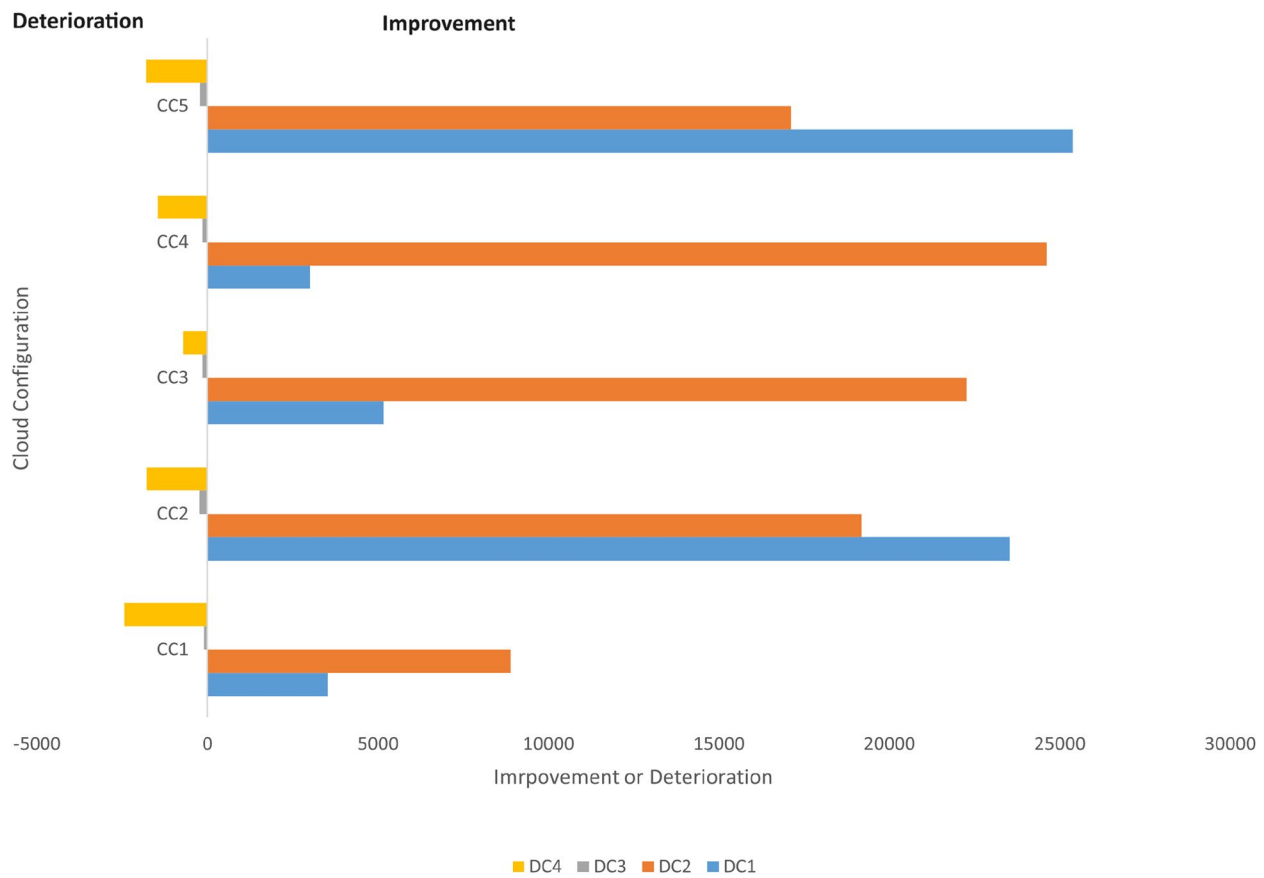

Fig. 14 Comparison of improvement or deterioration of datacenter request servicing times (milliseconds) for each DC

Table 17 Comparison of improvement or deterioration of datacenter request servicing times (milliseconds) for each DC

Number of DC	CDC	ORT	GA
3 DCs	4050.51	2001.92	1584.45
4 DCs	2419.39	812.23	741.27
5 DCs	1946.05	694.86	576.48

Table 18 Average data processing time (milliseconds) varying the number of DCs

Number of DC	CDC	ORT	GA
3 DCs	4050.51	1629.74	1131.4
4 DCs	2100.54	601.34	518.01
5 DCs	1702.05	527.29	293.69

userbases with varying user counts. The comparative results of userbase response times are graphically represented in 'Figs. 18", "19", and "20", corresponding to 3 DCs, 4 DCs, and 5 DCs, respectively.

Upon scrutinizing "Figs. 18", "19," and "20", it becomes apparent that, for the majority of userbases, our proposed GA-based service broker policy either outperforms ORT in terms of response time or achieves comparable performance. This observation allows us to assert that our proposed GA-based service broker policy method can maintain superior response times for individual userbases.

To further substantiate this claim, Table 16 reinforces our findings by illustrating that the average userbase response time for GA-based service broker policy is consistently superior to that of ORT across all DC configurations. Consequently, our proposed GA-based approach demonstrates efficacy in enhancing the system's overall responsiveness for diverse userbases and DC setups.

Performance comparison with other evolutionary algorithms

The performance of our proposed GA-based DC selection strategy is assessed by comparing it with basic GA and Particle Swarm Optimization (PSO). Initial tests with basic GA, which used random selection and

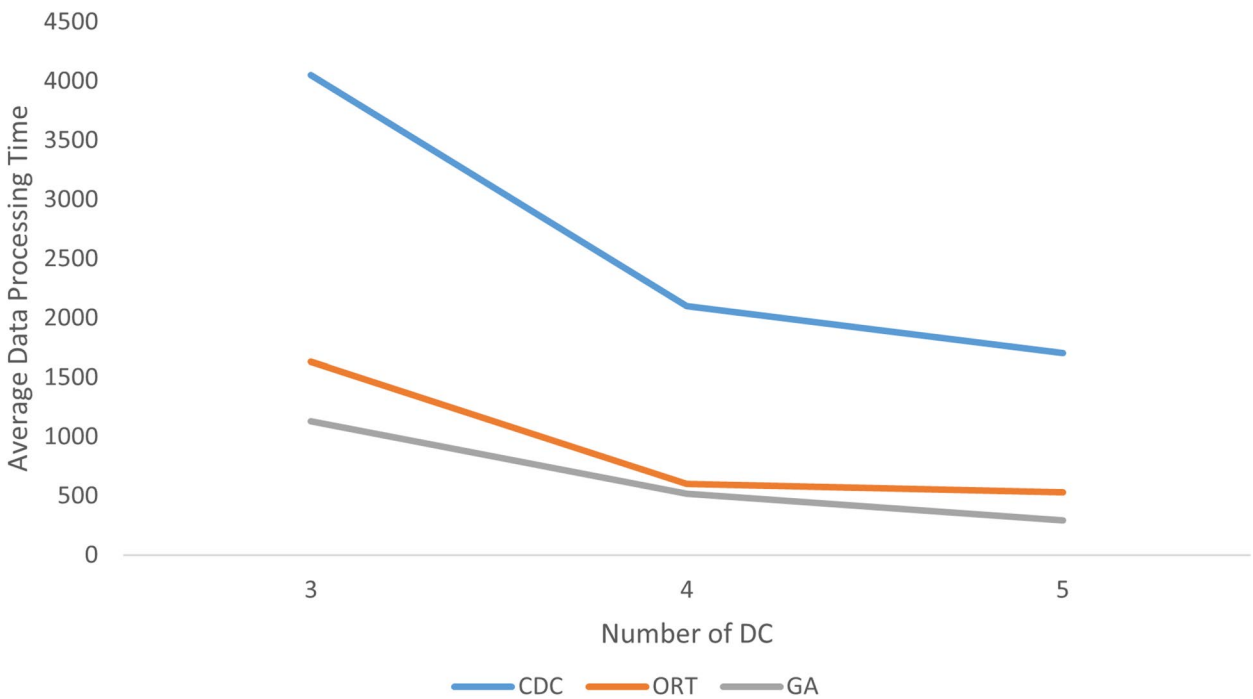

Fig. 15 Comparison of response time (milliseconds) varying the number of DCs

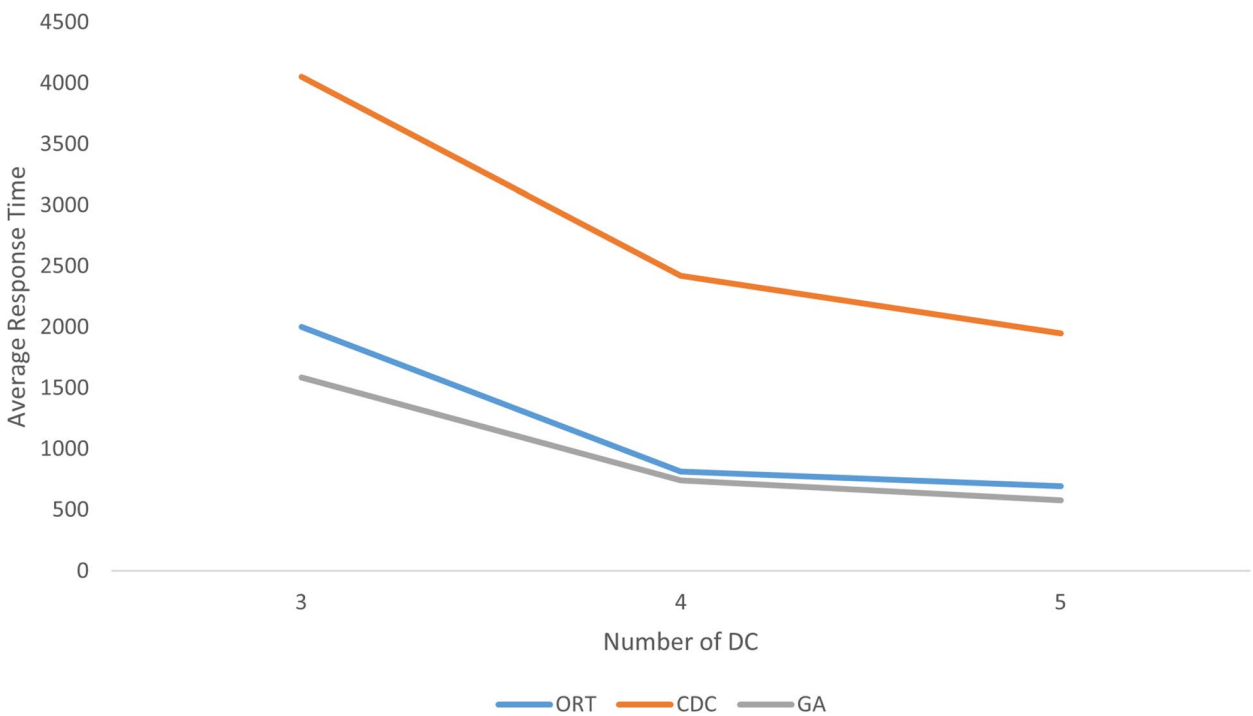

Fig. 16 Comparison of data processing time (milliseconds) varying the number of DCs

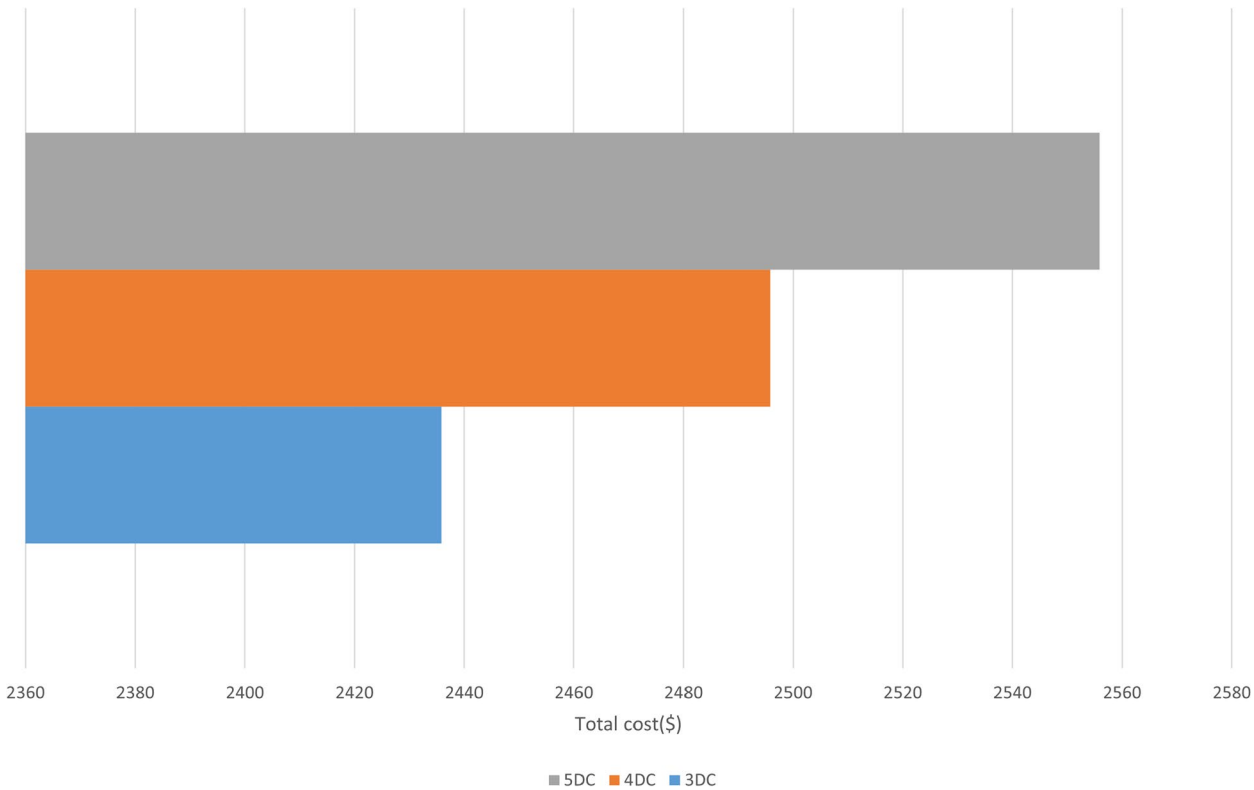

Fig. 17 Comparison of data processing time (milliseconds) varying the number of DCs

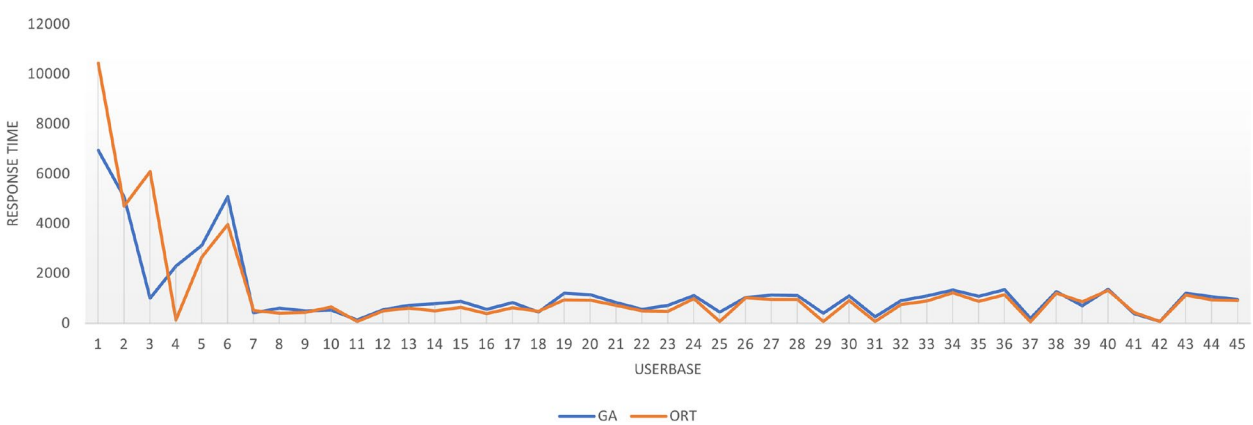

Fig. 18 Comparison of userbase response time (milliseconds) with 3 DCs

Table 19 Total cost($) varying the number of DCs

Number of DC	Total cost ($)
3	2435.85
4	2495.85
5	2555.85

single-point crossover and mutation strategies, showed poor results, as seen in Tables 21 and 22. The random chromosome selection in basic GA led to sub optimal crossover and mutation operations, reducing its efficiency in finding optimal solutions for DC selection. In contrast, PSO exhibited competitive results, especially in CC_1 and CC_4 configurations, where it outperformed

Table 20 Individual userbase response time (milliseconds) with GA and ORT for different DC configuration

Userbase	3DC		4 DC		5 DC	
	GA	ORT	GA	ORT	GA	ORT
UB1	6936.07	10429.62	1640.97	2272.75	613.65	1624.64
UB2	5066.86	4702.05	3010.16	787.94	438.91	677.67
UB3	987.05	6094.03	1135.86	5044.3	512.29	5604.1
UB4	2289.37	119.8	179.42	120.26	2344.72	120.18
UB5	3127.04	2648.62	2212.36	1353.11	914.73	72.83
UB6	5075.06	3963.39	1493.46	701.94	2436.3	513.46
UB7	402.13	496.51	629.22	497.3	615.09	455.13
UB8	597.56	398.3	635.12	393.21	384.71	364.13
UB9	492.85	432.39	457.92	409.47	441.86	399.78
UB11	526.02	652.14	466.53	167.42	584.88	132.1
UB12	130.93	54.31	99.63	54.09	348.06	54.16
UB13	527.66	490.63	731.2	475.21	542.82	440.2
UB14	708.74	602.12	443.99	157.1	451.53	126.65
UB15	772.77	481.92	540.94	500.33	551.29	439.41
UB16	871.18	634.42	359.05	161.72	372.07	133.8
UB17	553.64	375.32	470.74	321.44	416.2	359.46
UB18	814.21	620.55	534.09	157.31	328.2	137.14
UB19	446.18	473.28	565.01	476.51	502.18	465.77
UB20	1202.19	932.12	819.36	304.15	819.87	284.45
UB21	1138.63	920.75	962.94	681.53	263.3	55.11
UB22	823.47	715.46	322.09	272.53	504.73	261.33
UB23	544.75	484.52	750.32	474.23	429.87	448.37
UB24	703.7	477.6	671.49	474.99	424.33	449.1
UB25	1108.36	980.58	994.76	698.08	763.42	55.81
UB26	442.67	54.84	304.31	54.92	753.81	54.89
UB27	1023.82	1012.37	359.35	307.8	552.72	273.96
UB28	1116.45	949.18	627.95	302.93	668.57	283.21
UB29	1106.61	946.23	270.97	298.26	638.25	281.19
UB30	387.22	54.69	75.77	54.88	537.02	54.65
UB31	1085.7	902.67	973.72	630.58	114.5	54.81
UB32	255.89	54.69	120.53	54.71	478.43	54.76
UB33	899.59	732.87	545.77	577.27	376.12	53.08
UB34	1091	883.22	781.32	633.26	442.87	54.7
UB35	1330.78	1211.94	572.67	332.93	505.64	301.52
UB36	1065.9	861.63	572.73	295.65	455.58	275.63
UB37	1341.07	1134.09	594.58	329.42	524.92	294.86
UB38	186.95	54.1	158.78	54.16	231.62	54.04
UB39	1261.1	1202.68	715.89	326.78	565.4	293.42
UB40	693.95	851.88	231.68	177.21	422.22	151.65
UB41	1359.75	1303.76	595.4	343.53	302.3	311.53
UB42	384.63	432.84	381.41	417.46	417.67	423.42
UB43	59.76	54.78	54.72	54.78	266.14	54.8
UB44	1203.25	1112.23	972.98	730.35	192.04	56.6
UB45	1060.25	931.65	449.57	294.03	666.67	283.76
UB46	944.08	899.81	369.67	188.4	371.46	156.01

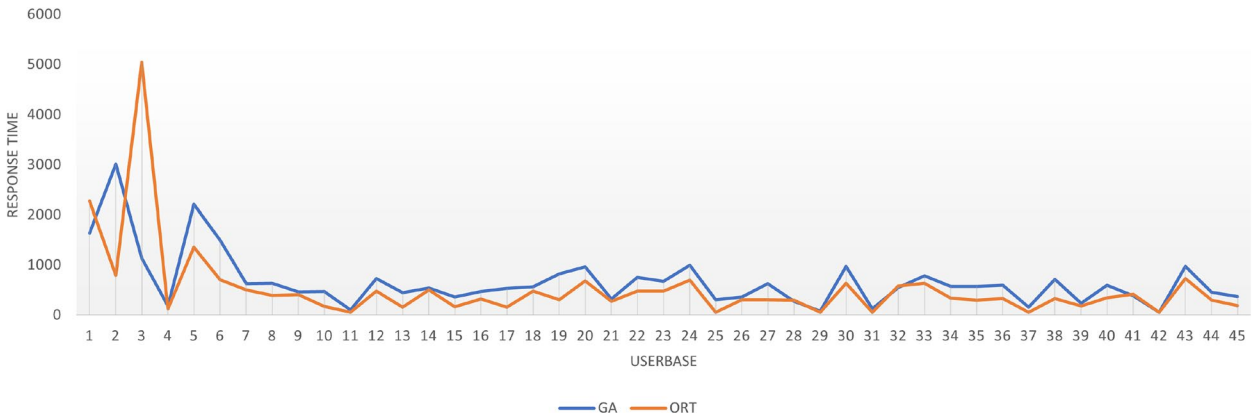

Fig. 19 Comparison of userbase response time (milliseconds) with 4 DCs

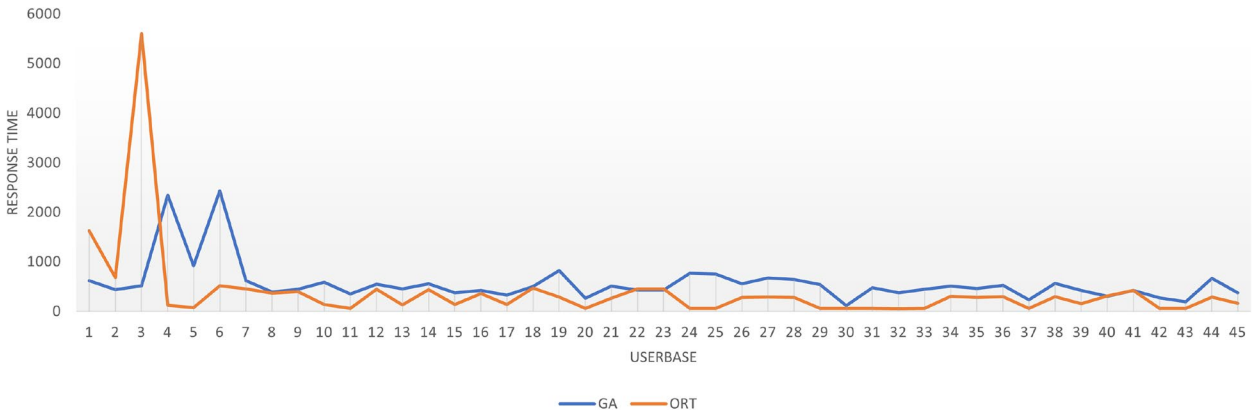

Fig. 20 Comparison of userbase response time (milliseconds) with 5 DCs

the basic GA due to its ability to quickly follow the best-performing particles in the swarm.

However, PSO's exploration capability is limited as it does not generate new individuals but focuses on altering the position and velocity of the existing population. This limited exploration frequently resulted in the algorithm being stuck in local optima, as seen in its inconsistent performance across the configurations. Our novel GA approach, by employing elitist selection, targeted mutation on less-fit individuals, and adaptive population reduction, significantly improves exploration and exploitation of the global search space. This allowed our proposed GA to consistently converge toward global optima, avoiding the stagnation issues seen in PSO.

Tables 21 and 22 clearly demonstrate the superior performance of the proposed GA. For instance, the average response time for CC2 and CC3 improved by 18% and 23% respectively compared to PSO. Moreover, the average data processing time across all configurations saw notable reductions, with the proposed GA achieving a reduction of 32.10%, which surpasses the improvements seen with both PSO and basic GA. "Figures 21" and "22" further illustrate the consistency of these improvements.

In conclusion, while both basic GA and PSO show their strengths in certain configurations, the overall robustness and adaptability of our proposed GA make it the superior choice for optimizing cloud service broker policies. By effectively balancing exploration and exploitation, it ensures better performance across a variety of userbase and DC configurations.

Performance comparison with other research works

Based on the above results and discussion, it is evident that our proposed GA significantly outperforms the baseline service broker policies available in CloudAnalyst. While previous researchers have implemented evolutionary algorithms in service broker policies [45–47], a direct comparison with existing evolutionary algorithm-based service broker policies is necessary.

Table 21 Average response time(milliseconds) of genetic algorithms and other evolutionary algorithms

Configuration name	Basic GA	PSO	Proposed GA
CC1	2045.91	937.44	1964.1
CC2	2890.05	1758.34	1406.27
CC3	2039.54	1407.91	1248.74
CC4	2121.31	768.55	871.72
CC5	1182.51	904.08	741.27

Table 22 Average data processing time(milliseconds) of genetic algorithms and other evolutionary algorithms

Configuration name	Basic GA	PSO	Proposed GA
CC1	1545.93	568.82	1574.66
CC2	2204.75	1288.32	951.24
CC3	1523.33	943.16	764.71
CC4	2121.31	460.72	513.38
CC5	815.23	575.65	518.01

Among the two baseline service broker policies in CloudAnalyst, ORT policy performs better than the CDC policy. Therefore, we have calculated the improvements of both the existing evolutionary algorithm research and our proposed GA in comparison to ORT. Table 23 illustrates this comparison.

From Table 23, it is apparent that previous research has achieved improvements of 23.27% and 31.18% in terms of userbase response time in the cloud environment. In contrast, our proposed GA demonstrates a 24.73% improvement in userbase response time. Additionally, considering the data processing time of the DCs, previous research has shown improvements of 31.18% and 8.10%, while our proposed GA has achieved a 32.10% improvement. "Figure 23" visualizes the comparison of all evolutionary algorithms, highlighting the superior performance of our proposed GA compared to existing research works on evolutionary algorithms.

These results illustrate the efficacy of our proposed GA in optimizing service broker policies, achieving substantial improvements in both userbase response time and data processing time.

Real-world adaptation of our proposed GA

Global e-commerce platforms serve millions of users daily across multiple regions as depicted in "Fig. 6". As userbases grow, these platforms face increasing challenges with their cloud infrastructure. During high-traffic periods, such as sales events, the platforms experience slow response times, overloaded datacenters, and rising operational costs. Traditional service broker policies, such as the CDC and ORT, are unable to meet the dynamic nature of user demand and

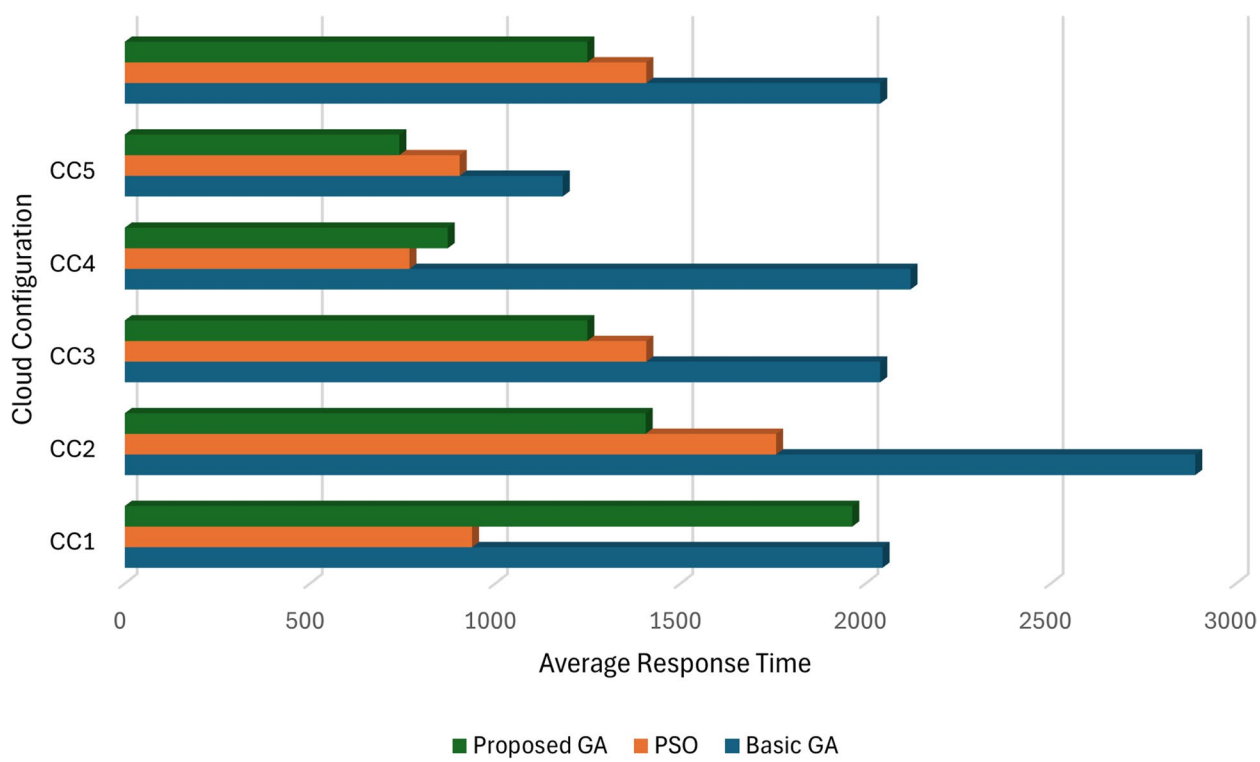

Fig. 21 Comparative analysis of average response time between genetic algorithms and evolutionary algorithms

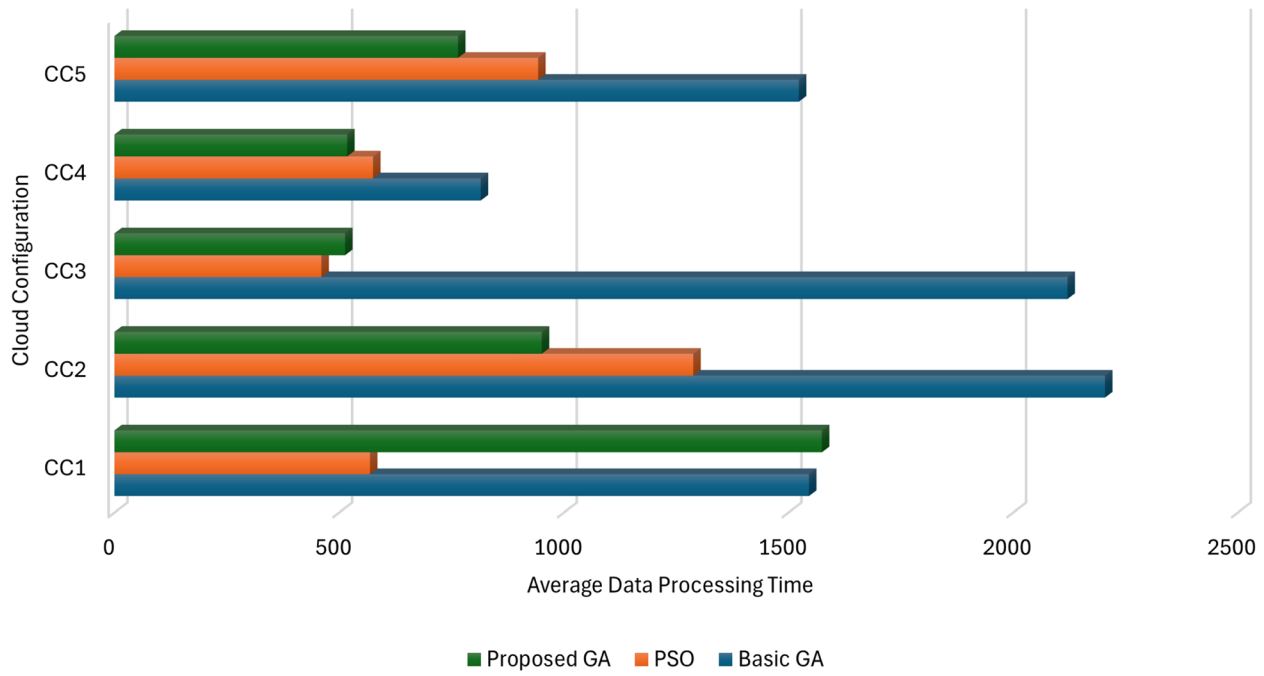

Fig. 22 Comparative analysis of data processing time between genetic algorithms and evolutionary algorithms

fluctuating network conditions. This results in uneven traffic distribution and inefficient resource utilization, leading to performance degradation.

To address these challenges, the platform needed to balance high user demand across regions, reduce response times during peak hours, and minimize the costs of using premium datacenters. Additionally, it required a solution that could adapt in real-time to the changing cloud environment, where network latency and DC performance vary frequently.

Our proposed GA-driven service broker policy offers a solution to these issues by intelligently selecting the most suitable datacenters for user requests. This policy optimizes cloud performance by reducing response times, minimizing data processing delays, and lowering operational costs, all while adapting to real-time fluctuations in network conditions and user demand.

The GA-driven service broker policy models each DC selection as a chromosome, representing different configurations for various userbases. The fitness function aims to minimize response time, data processing time, and operational costs. Through genetic operations such as selection, crossover, and mutation, the algorithm refines these configurations iteratively, continuously improving the cloud's performance.

Table 23 Improvement in response and data processing times between existing and proposed evolutionary algorithms in comparison to ORT

DC selection algorithms	Response time	Data processing time
VSBRP [45]	23.27%	31.18%
OSBRP/DE [46]	19.28%	8.10%
GA	24.73%	32.10%

Our experimental results, as outlined in "Result analysis" section, demonstrate that the proposed GA-driven service broker policy significantly enhances cloud performance, reducing both response and processing times in real-world scenarios as shown in "Experimental scenarios" section. The algorithm outperforms traditional methods and exhibits consistent improvements compared to other evolutionary algorithms. Additionally, the GA-driven approach achieves better results with fewer datacenters, reducing the need for building new infrastructure, which lowers overall cloud costs. This reduction in datacenter expansion also contributes to environmental sustainability by decreasing emissions from datacenter operations.

In conclusion, the GA-driven service broker policy provides a dynamic, scalable, and cost-effective solution to the challenges faced by global e-commerce platforms. By optimizing datacenter selection in real-time and reducing operational costs, this approach improves cloud performance while supporting environmental sustainability. These benefits make the proposed solution highly applicable to large-scale cloud environments, offering a significant improvement over traditional methods.

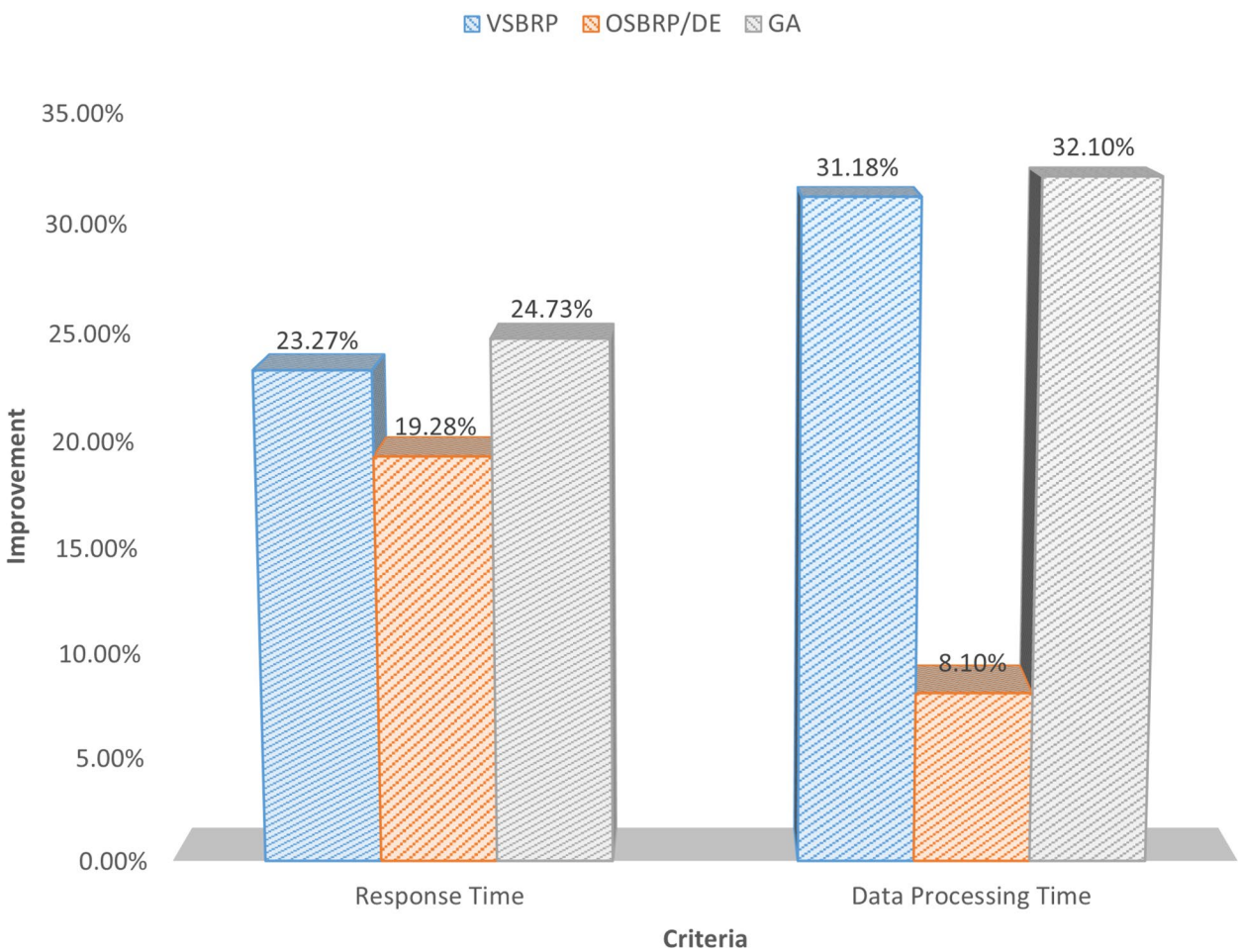

Fig. 23 Comparison of various evolutionary algorithms against ORT

Conclusion

Numerous applications integral to our daily routines are intricately connected to the cloud, either directly or indirectly. The satisfaction of cloud users hinges on the service broker policy employed by cloud services. This policy ensures user contentment by choosing the optimal DC to minimize response and data processing times for user requests. In our research, we introduce a service broker policy grounded in GA to enhance the overall performance of the cloud environment. The principal rationale behind employing GA for the service broker policy lies in its capacity to evolve and converge toward an optimal solution.

Based on GA, the service broker policy proposed in our research initiates with a randomly generated initial population. Subsequently, the solution from the CDC policy is introduced into this population. The algorithm iteratively undergoes evolution through selection, crossover, and mutation processes to identify the most optimal DC sequence for each userbase. This determined sequence is then employed for the ensuing hour. In the subsequent hour, the GA is rerun to ascertain another optimal DC sequence. If the current sequence proves superior to the preceding one, the service broker policy adopts it; otherwise, it reverts to the previous sequence.

The results analysis findings demonstrate the proposed algorithm's superior performance to traditional counterparts such as the ORT and CDC policies. This evaluation comprehensively examined average response and data processing times under varying userbase configurations. Furthermore, a detailed investigation into the servicing time of individual DCs revealed that the GA-based service broker policy, as proposed, exhibited enhanced resource utilization compared to the ORT policy. Moreover, it was observed that the GA-based service broker policy could achieve comparable or superior performance with a reduced number of DCs when contrasted with both ORT and CDC policies.

A cost analysis was also conducted, affirming the financial viability of the proposed GA-based service broker policy, particularly in the context of large-scale cloud services. Importantly, this financial benefit was demonstrated without compromising the overall performance of the cloud environment. The outcomes of this study underscore the potential practical and economic advantages of adopting the proposed GA-based approach for service broker policy in large-scale cloud service infrastructures.

We conducted a comprehensive evaluation of the performance of our proposed GA in comparison with other evolutionary algorithms. The results indicate that our proposed GA consistently outperforms alternative evolutionary approaches, demonstrating superior and more reliable performance. Additionally, we compared our GA-based service broker policy with existing research on evolutionary algorithm-based service broker policies. In these comparisons, our proposed GA achieved notable improvements in performance, particularly in minimizing response time and data processing time, outperforming all other evolutionary algorithms considered.

To maintain simplicity, we exclusively employed the round-robin load-balancing policy. However, our algorithm will be evaluated against various load balancing policies in future comparisons, including Equally Spread Execution Current Load, Throttled, and Machine Learning-based load balancing policies. The GA can also undergo testing with different selection policies, such as a roulette wheel, a tournament, and elitism. Moreover, diverse crossover strategies such as (P, C), (P + C), (P/R), (C), and (P/R + C) will be explored, where P denotes the parent, and C signifies offspring or children.

Acknowledgements
This material is based upon work supported by a grant from the O'Reilly Auto Parts.

Authors' contributions
Research Work and Documents is primarily written by Shusmoy Chowdhury under the supervision of Ajay Katangur. All authors reviewed the manuscript.

Funding
O'Reilly Auto Parts.

Data availability
No datasets were generated or analysed during the current study.

Declarations

Ethics approval and consent to participate
Yes.

Consent for publication
Yes.

Competing interests
The authors declare no competing interests.

Received: 18 July 2024 Accepted: 31 October 2024
Published online: 14 November 2024

References

1. Bello SA, Oyedele LO, Akinade OO, Bilal M, Delgado JMD, Akanbi LA, Ajayi AO, Owolabi HA (2021) Cloud computing in construction industry: Use cases, benefits and challenges. Autom Constr 122:103441
2. Al Masarweh M, Alwada'n T, Afandi W (2022) Fog computing, cloud computing and iot environment: advanced broker management system. J Sens Actuator Netw 11(4):84
3. Katal A, Dahiya S, Choudhury T (2023) Energy efficiency in cloud computing data centers: a survey on software technologies. Clust Comput 26(3):1845–1875
4. Tari M, Ghobaei-Arani M, Pouramini J, Ghorbian M (2024) Auto-scaling mechanisms in serverless computing: A comprehensive review. Comput Sci Rev 53:100650
5. Ebrahimi A, Ghobaei-Arani M, Saboohi H (2024) Cold start latency mitigation mechanisms in serverless computing: taxonomy, review, and future directions. J Syst Archit 151:103115
6. Stergiou C, Psannis KE, Gupta BB, Ishibashi Y (2018) Security, privacy & efficiency of sustainable cloud computing for big data & iot. Sustain Comput Inform Syst 19:174–184
7. Alouffi B, Hasnain M, Alharbi A, Alosaimi W, Alyami H, Ayaz M (2021) A systematic literature review on cloud computing security: threats and mitigation strategies. IEEE Access 9:57792–57807
8. Alwada T, Al-Zitawi O, Khawaldeh S, Almasarweh M (2015) Privacy and control in mobile cloud systems. Int J Comput Appl 113(1):12–15
9. Mbarek N (2021) Service level management in the cloud. Serv Level Manag Emerg Environ 20:45–81
10. Washizaki H, Xia T, Kamata N, Fukazawa Y, Kanuka H, Kato T, Yoshino M, Okubo T, Ogata S, Kaiya H et al (2021) Systematic literature review of security pattern research. Information 12(1):36
11. Butpheng C, Yeh KH, Xiong H (2020) Security and privacy in iot-cloud-based e-health systems—a comprehensive review. Symmetry 12(7):1191
12. De Sousa NFS, Perez DAL, Rosa RV, Santos MA, Rothenberg CE (2019) Network service orchestration: A survey. Comput Commun 142:69–94
13. Voorsluys W, Broberg J, Buyya R (2011) Introduction to cloud computing. Cloud Comput Princ paradigms 1:1–41
14. Varghese B, Leitner P, Ray S, Chard K, Barker A, Elkhatib Y, Herry H, Hong CH, Singer J, Tso FP et al (2019) Cloud futurology. Computer 52(9):68–77
15. Fernández-Cerero D, Jakóbik A, Grzonka D, Kołodziej J, Fernández-Montes A (2018) Security supportive energy-aware scheduling and energy policies for cloud environments. J Parallel Distrib Comput 119:191–202
16. Ruse M (2009) Charles darwin on human evolution. J Econ Behav Org 71(1):10–19
17. Bartz-Beielstein T, Branke J, Mehnen J, Mersmann O (2014) Evolutionary algorithms. Wiley Interdiscip Rev Data Min Knowl Disc 4(3):178–195
18. Ghorbian M, Ghobaei-Arani M, Esmaeili L (2024) A survey on the scheduling mechanisms in serverless computing: a taxonomy, challenges, and trends. Clust Comput 27:1–40
19. Aghazadeh R, Shahidinejad A, Ghobaei-Arani M (2023) Proactive content caching in edge computing environment: A review. Softw Pract Experience 53(3):811–855
20. Rathi DR, Sharma V, Bole SK (2013) Round robin data center selection in single region for service proximity service broker in cloud analyst. Int J Comput Technol 4(2):254–260
21. Jaikar A, Kim GR, Noh SY (2013) Effective data center selection algorithm for a federated cloud. Adv Sci Technol Lett 35:66–69
22. Chudasama D, Trivedi N, Sinha R (2012) Cost effective selection of data center by proximity-based routing policy for service brokering in cloud environment. Int J Comput Technol Appl 3(6):2057–2059
23. Dinh HT, Lee C, Niyato D, Wang P (2013) A survey of mobile cloud computing: architecture, applications, and approaches. Wirel Commun Mob Comput 13(18):1587–1611
24. Radi M (2014) Weighted round robin policy for service brokers in a cloud environment. The International Arab Conference on Information Technology (ACIT2014). Nizwa, Oman, pp 45–49
25. Kapgate D (2014) Improved round robin algorithm for data center selection in cloud computing. Int J Eng Sci Res Technol 3(2):686–691

26. Ahmed AS (2012) Enhanced proximity-based routing policy for service brokering in cloud computing. Int J Eng Res Appl(India) 2(2):1453–1455
27. Kapgate D (2014) Weighted moving average forecast model based prediction service broker algorithm for cloud computing. Int J Comput Sci Mob Comput 3(2):71–79
28. Naha RK, Othman M (2016) Cost-aware service brokering and performance sentient load balancing algorithms in the cloud. J Netw Comput Appl 75:47–57
29. Rekha P, Dakshayini M (2014) Service broker routing polices in cloud environment: a survey. Int J Adv Eng Technol 6(6):2717
30. Shi T, Ma H, Chen G (2019) A seeding-based ga for location-aware workflow deployment in multi-cloud environment. In: 2019 IEEE Congress on Evolutionary Computation (CEC), IEEE, pp 3364–3371
31. Shi T, Ma H, Chen G, Hartmann S (2020) Location-aware and budget-constrained service deployment for composite applications in multi-cloud environment. IEEE Trans Parallel Distrib Syst 31(8):1954–1969
32. Sharma V (2014) Efficient data center selection policy for service proximity service broker in cloudanalyst. Int J Innov Comp Sci Eng(IJICSE) 1(1):21–28
33. Shi T, Ma H, Chen G (2019) A genetic-based approach to location-aware cloud service brokering in multi-cloud environment. In: 2019 IEEE International Conference on Services Computing (SCC), IEEE, pp 146–153
34. Heilig L, Buyya R, Voß S (2017) Location-aware brokering for consumers in multi-cloud computing environments. J Netw Comput Appl 95:79–93
35. Shi T, Ma H, Chen G, Hartmann S (2021) Location-aware and budget-constrained service brokering in multi-cloud via deep reinforcement learning. In: Service-Oriented Computing: 19th International Conference, ICSOC 2021, Virtual Event, November 22–25, 2021, Proceedings 19, Springer, pp 756–764
36. Mishra RK, Kumar S, Naik BS (2014) Priority based round-robin service broker algorithm for cloud-analyst. In: 2014 IEEE International Advance Computing Conference (IACC), IEEE, pp 878–881
37. Tordsson J, Montero RS, Moreno-Vozmediano R, Llorente IM (2012) Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Futur Gener Comput Syst 28(2):358–367
38. Zhang Q, Cheng L, Boutaba R (2010) Cloud computing: state-of-the-art and research challenges. J Internet Serv Appl 1:7–18
39. Nayak B, Bisoyi B, Pattnaik PK (2023) Data center selection through service broker policy in cloud computing environment. Mater Today Proc 80:2218–2223
40. Nandwani S, Achhra M, Shah R, Tamrakar A, Joshi K, Raksha S (2016) Weight-based data center selection algorithm in cloud computing environment. In: Artificial Intelligence and Evolutionary Computations in Engineering Systems: Proceedings of ICAIECES 2015, Springer, pp 515–525
41. Al-Tarawneh M (2015) A fuzzy logic based proximity-aware cloud service broker algorithm. Int Rev Comput Softw 10(10):1027–1036
42. Arya D, Dave M (2017) Priority based service broker policy for fog computing environment. Advanced Informatics for Computing Research: First International Conference, ICAICR 2017, Jalandhar, India, March 17–18, 2017. Springer, Revised Selected Papers, pp 84–93
43. Manasrah AM, Smadi T, ALmomani A (2017) A variable service broker routing policy for data center selection in cloud analyst. J King Saud Univ-Comput Inf Sci 29(3):365–377
44. Zhang J, Huang H, Wang X (2016) Resource provision algorithms in cloud computing: A survey. J Netw Comput Appl 64:23–42
45. Limbani D, Oza B (2012) A proposed service broker strategy in cloud-analyst for cost-effective data center selection. Int J Eng Res Appl India 2(1):793–797
46. Manasrah AM, Aldomi A, Gupta BB (2019) An optimized service broker routing policy based on differential evolution algorithm in fog/cloud environment. Clust Comput 22:1639–1653
47. Kessaci Y, Melab N, Talbi EG (2013) A pareto-based genetic algorithm for optimized assignment of vm requests on a cloud brokering environment. In: 2013 IEEE congress on evolutionary computation, IEEE, pp 2496–2503
48. Quarati A, D'Agostino D (2017) Moea-based brokering for hybrid clouds. In: 2017 International Conference on High Performance Computing & Simulation (HPCS), IEEE, pp 611–618
49. Raghuwanshi S, Kapoor S (2018) The new service brokering policy for cloud computing based on optimization techniques. Int J Eng Tech 4:481–488
50. Salimian M, Ghobaei-Arani M, Shahidinejad A (2022) An evolutionary multi-objective optimization technique to deploy the iot services in fog-enabled networks: an autonomous approach. Appl Artif Intell 36(1):2008149
51. Chowdhury S, Katangur A (2023) Optimization of datacenter selection policy in cloud computing using differential evolution algorithm. In: 2023 IEEE Cloud Summit, IEEE, pp 33–40
52. Tiwari S, Parwar P (2018) A modified service broker policy based on cuckoo optimization. Int J Sci Adv Res Technol 4:202–206
53. Cui Z, Zhao T, Wu L, Qin A, Li J (2023) Multi-objective cloud task scheduling optimization based on evolutionary multi-factor algorithm. IEEE Trans Cloud Comput 11(4):3685–3699
54. Vikhar PA (2016) Evolutionary algorithms: A critical review and its future prospects. In: 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), pp 261–265. https://doi.org/10.1109/ICGTSPICC.2016.7955308
55. Reeves CR (2010) Genetic Algorithms. In: Gendreau M, Potvin JY (eds) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol 146. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-1665-5_
56. Liu S, Louis SJ (2016) Comparing two representations for evolving micro in 3d rts games. In: 2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, pp 722–729
57. Holland JH (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT Press
58. Eshelman LJ (1991) The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In: Foundations of genetic algorithms, vol 1, Elsevier, pp 265–283
59. Engelbrecht AP (2007) Computational intelligence: an introduction. John Wiley & Sons
60. Hasançebi O, Erbatur F (2000) Evaluation of crossover techniques in genetic algorithm based optimum structural design. Comput Struct 78(1):435–448. https://doi.org/10.1016/S0045-7949(00)00089-4
61. Zainuddin FA, Abd Samad MF, Tunggal D (2020) A review of crossover methods and problem representation of genetic algorithm in recent engineering applications. Int J Adv Sci Technol 29(6s):759–769
62. Patil S, Bhende M (2014) Comparison and analysis of different mutation strategies to improve the performance of genetic algorithm. Int J Comput Sci Inf Technol 5(3):4669–4673
63. De Falco I, Della Cioppa A, Tarantino E (2002) Mutation-based genetic algorithm: performance evaluation. Appl Soft Comput 1(4):285–299
64. Technavio (2017) Top 10 cloud computing service providers in 2017. https://www.technavio.com/blog/top-10-cloud-computing-service-providers-2017. Accessed 20 Feb 2018
65. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R (2011) Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Experience 41(1):23–50
66. Wickremasinghe B, Calheiros RN, Buyya R (2010) 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, IEEE, pp 446–452
67. Shahid MA, Alam MM, Su'ud MM (2023) Performance evaluation of load-balancing algorithms with different service broker policies for cloud computing. Appl Sci 13(3):1586
68. Mulla BP, Krishna CR, Tickoo RK (2020) Load balancing algorithm for efficient vm allocation in heterogeneous cloud. Int J Comput Netw Commun 12:83–96
69. Chowdhury S, Katangur A (2022) Threshold based load balancing algorithm in cloud computing. In: 2022 IEEE International Conference on Joint Cloud Computing (JCC), IEEE, pp 23–28
70. Katangur A, Akkaladevi S, Vivekanandhan S (2022) Priority weighted round robin algorithm for load balancing in the cloud. In: 2022 IEEE 7th International Conference on Smart Cloud (SmartCloud), IEEE, pp 230–235
71. Pattanaik PA, Roy S, Pattnaik PK (2015) Performance study of some dynamic load balancing algorithms in cloud computing environment. In: 2015 2nd International Conference on Signal Processing and Integrated Networks (SPIN), IEEE, pp 619–624
72. Domanal SG, Reddy GRM (2013) Load balancing in cloud computingusing modified throttled algorithm. In: 2013 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), IEEE, pp 1–5
73. Islam T, Hasan MS (2017) A performance comparison of load balancing algorithms for cloud computing. In: 2017 International Conference on the Frontiers and Advances in Data Science (FADS), IEEE, pp 130–135

74. Mandal G, Dam S, Dasgupta K, Dutta P (2019) Load balancing strategy in cloud computing using simulated annealing. In: Computational Intelligence, Communications, and Business Analytics: Second International Conference, CICBA 2018, Kalyani, India, July 27–28, 2018, Revised Selected Papers, Part I 2, Springer, pp 67–81
75. Oberlo (2023) How many users does facebook have? https://www.oberlo.com/statistics/how-many-users-does-facebook-have. Accessed 30 Nov 2023
76. Xiong J, Shi SH, Zhang S (2017) Build and evaluate a free virtual cluster on amazon elastic compute cloud for scientific computing. Int J Online Eng 13(8):121–132
77. Chaiyaratana N, Zalzala AM (1999) Hybridisation of neural networks and genetic algorithms for time-optimal control. In: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406), IEEE, vol 1, pp 389–396
78. Jareanpon C, Pensuwon W, Frank R, Davey N (2004) An adaptive RBF network optimised using a genetic algorithm applied to rainfall forecasting. In: IEEE International Symposium on Communications and Information Technology, 2004. ISCIT 2004, IEEE, vol 2, pp 1005–1010
79. Hamdan M (2008) A heterogeneous framework for the global parallelisation of genetic algorithms. Int Arab J Inf Technol 5(2):192–199

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.