Genetic Algorithm Based Service Broker Policy to find Optimal Datacenters in Cloud Services

Shusmoy Chowdhury
Department of Computer Science
Missouri State University
Springfield, USA
sc26s@missouristate.edu

Ajay Katangur
Department of Computer Science
Missouri State University
Springfield, USA
ajaykatangur@missouristate.edu

Alaa Sheta
Computer Science Department
Southern Connecticut State University
New Haven, Connecticut
shetaa1@southernct.edu

Nagesh Rao Psayadala
Department of Computer Science
Missouri State University
Springfield, USA
np58s@missouristate.edu

Siming Liu
Department of Computer Science
Missouri State University
Springfield, USA
simingliu@missouristate.edu

2023 8th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA) | 978-1-6654-5533-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCCBDA56900.2023.10154791

***Abstract*—In modern-day computing, cloud services are widely used in every aspect of life. So, user satisfaction depends on the effectiveness and efficiency of cloud services. Service broker policy of the cloud maintains the effectiveness and efficiency of cloud services. Service broker policy provides the rules and norms based on which a data center is selected for a userbase request. This paper proposes a genetic algorithm-based service broker policy that provides the optimal sequence of data centers for different userbases based on their requirements. This research aims to find an optimal data center for userbases that can achieve user satisfaction by minimizing the cloud service's response time and data processing time. We have experimented with our proposed genetic algorithm-based service broker policy in the CloudAnalyst platform based on different real-world scenarios. Simulation results indicate that our proposed genetic algorithm outperforms existing traditional algorithms.**

***Keywords*—cloud computing, service broker policy, genetic algorithm**

I. Introduction

Modern technology is evolving and changing rapidly to embrace the cloud environment. Cloud computing is one of the emerging computer science exemplars of distributed computing. Cloud computing environment provides application, data, and infrastructure as a service. The services can be used in an omnipresent, flexible, and transparent way [1]. The introduction of cloud-based software has drastically changed many aspects of everyday life [2]. The impact of cloud computing cannot be ignored, especially in terms of high availability and scalability. Many companies have moved towards the cloud environment to increase the efficiency and effectiveness of their work [3]. The shift towards cloud environments has expanded tech companies' services with minimal cost and no need to buy physical hardware and software. Nowadays most of the applications are available over the internet. So there increases the necessity to shift computing resources from the user's location to the service provider. The idea of this kind of shifting is known as cloud computing. It satisfies user requests based on available resources. Cloud computing provides great flexibility with computing resources such as storage, platform, software, power, and bandwidth. Cloud computing ensures better solutions with its service for the users.

Before the invention of cloud computing, the traditional approach for organizations was to buy, deploy and manage computing facilities and tools with high expenses. But these resources need to be utilized most of the time to take advantage of the high expenses. On the other hand, some of these resources have strict contractual conditions with third-party providers. The main advantage of cloud computing is on-demand and pay-per-use services.

The success of cloud computing depends on two crucial factors one is Load Balancing, and another is Service Broker Policy [4]. Service Broker Policy consists of rules and regulations that determine the data center (DC) selection for a particular userbase. The choice of DC needs to be highly effective in cloud computing. The unsuitable selection of the DC might decrease the performance of the cloud. There are already a few traditional approaches for service broker policy, such as the closest DC policy and optimized response time policy. The closest DC policy chooses the nearest DC from the user base; however, optimized response time selects the DC with a better response time. But these approaches may overload a particular DC with many user requests.

This research proposes an evolutionary-based service broker policy that uses a genetic algorithm (GA) for optimal DC selection. Evolutionary algorithms evolve with time and provide better solutions as time progresses. These algorithms are inspired by nature with a motivation of survival of the fittest. Cloud computing is a continuous process, so with time progress, our algorithm can find the optimal DC for the userbase.

We use the GA to design the service broker policy. Our goal

This material is based upon work supported by the National Science Foundation under Grant OAC-1828380

is to minimize the cloud's response time and data processing time so that the quality of the services and user satisfaction increase. Our algorithm will evolve through selection, crossover, and mutation, the basic steps of the GA, and progresses toward the optimal DC selection.

Cloud computing is an overly complex process. It depends on uncontrollable factors like network latency, and servers have different workloads. So, it isn't easy to measure the performance of internet-based applications using real cloud platforms [5]–[7]. Therefore, simulation-based experiments are needed to find a solution virtually and free of charge under stable and controllable environments [8]–[10]. There are several cloud simulators available for cloud research. Our research will use the CloudAnalyst Platform [10] to examine our proposed algorithm. CloudAnalyst is a GUI-based and extended version of CloudSim [9], which is very user-friendly and can be configured according to real-world scenarios. We will examine our algorithm with different real-world scenarios to analyze its performance. The performance of the algorithm is compared with two commonly used service broker policies, Closest Data center Policy (CDC) and Optimized Response Time Policy (ORT) [11].

The rest of the paper is structured as follows. Section 2 provides a summary of previous accomplishments in the area of service broker policy. Section 3 provides the problem encoding of how the GA can be applied in the cloud research domain. Section 4 outlines our proposed GA-based service broker policy and the motivation behind using GA. Section 5 explains the cloud workflow and how the service broker policy is integrated into the cloud environment. Section 6 describes the experimental setup in CloudAnalyst, which is used to test the performance of our proposed GA. Section 7 provides an analysis of the results of our algorithm compared to CDC and ORT policies. Finally, Section 8 concludes this paper with a summary and future extension of this research.

II. Related Work

Service broker policy determines the optimal DC that can handle the request of the users effectively and efficiently. The service broker policy is crucial in achieving user satisfaction and improving cloud performance. Many researchers have worked significantly on service broker policy in cloud computing. Various service broker policies are proposed to select the most appropriate DC to achieve user satisfaction and efficient resource management. The decision factors of the service broker policy vary based on the proposed solution. Some researchers focused on the overall cost to be minimized [12]–[15] on the other hand, some considered minimizing the response time [16]–[20] and execution time [11], [21] to design an effective service broker policy. But the problem is if we focus on a single factor, the other factors may be negatively affected. From [12]–[15], the response time increased when the service broker policy focused on overall cost minimization. Also, the policy should have considered the various workloads, bandwidth, and network latency which may cause both overall cost and response time to be reduced. Again, when the service broker policy aims to reduce the response time, it will increase the overall cost and execution time [16]–[20]. Also, it will cause underutilizing the resources and overloading the selected DC [22]–[24].

The cloud analyst platform has two traditional approaches to service broker policy [25]. The first one is the closest DC or proximity-based service broker policy, which chooses the closest DC for the user request but may overload the DC when requests are from the same region. To solve the problem, Devyaniba Chudasama et al. [26] and Dhaval Limbani et al. [27] proposed an enhanced or extended proximity-based routing policy to direct the selection of the nearest DC based on minimum data transfer cost. So, if the DC is overloaded with requests, it can send them to another DC with minimum transfer cost and improve its overall performance. Sharma et al. [12] proposed a round-robin load-balancing service broker policy where the user requests from a region can be portioned into different DCs of the same region so that no single DC can be overloaded with a huge workload. The simulation results show that the proposed policy enhanced the performance, with DCs having the same configurations. So, the results can only be satisfactory if the DCs have different configurations. Hence Misra et al. [28] designed a priority-based round-robin service broker policy by allocating the workload among available DCs to improve the overall cost and minimize the response time. But the inappropriate selection of DCs can increase overall costs and maximize response time due to unfair resource allocation and network latency. Bhavani and Guruprasad [29] designed an efficient resource allocation model based on network delay between userbase and DCs. This resource allocation model successfully allocates the user's requests to suitable DCs. Nandwani et al. [30] proposed a weight-based DC selection algorithm in a cloud environment for appropriate DC selection based on the weights of the DC. The results of the proposed algorithm reduce the response time, processing time, and overall cost. Kunal Kishor et al. [31] proposed a broker policy that uses a proportion weight to decide which DC is selected for servicing a particular user request. Utilizing a proportion weight ensures an equal workload for each DC.

The second service broker policy is an optimized service broker policy where the DC is selected based on its better response time. The optimized service broker policy can provide satisfactory results, but it may increase the data processing time of the DC, overloading it with more tasks.

Researchers also used meta-heuristic search or evolutionary algorithms to find the optimal DC for the userbase. Manasrah et al. [32] present a variable service broker routing policy (VSBRP) as a heuristic-based technique to minimize response time by selecting a DC based on communication channel bandwidth, latency, and job size. The DC redirects the user requests to the next DC using VSBRP to achieve better response time and processing time for the specific job and avoid DC overloading. Alfonso Quarati et al. [33] proposed a GA-based approach for Cloud Brokering, focusing on resource allocation for applications with diverse Quality of Service

(QoS) requirements. The proposed method may support cloud brokering in allocating batches of jobs to hybrid cloud infrastructures. Ahmad M. Manasrah et al. [34] presented a differential evaluation-based service broker policy to find an optimal DC for the request based on the user's constraints. The proposed service broker policy achieved better response time by minimizing overall cost, which is the sum of a virtual machine (VM) and data transfer costs. Yacine Kessaci et al. [1] used a Pareto-based meta-heuristic approach to design a service broker policy. The proposed algorithm is designed with a Multi-objective Genetic Algorithm Cloud Brokering (MOGA-CB) algorithm where the two objectives are response time and cost of the virtual machine. The results of the algorithm show efficiently practical Pareto sets of solutions. Minhaj Ahmad Khan et al. [35] proposed a normalization-based hybrid service brokering approach integrated with throttled round-robin load balancing to improve resource management through cost and performance-aware provision of cloud services. The proposed method used hybrid (static and dynamic) evaluation criteria using normalization for determining the impact of cost and performance-oriented parameters in a multi-cloud environment.

Najib A. Kofahi et al. [36] proposed an efficient service broker routing policy that improves users' satisfaction and cloud performance. The approach uses the Vector Space Model and multi-objective scalarization function to optimize conflicting objectives. In this paper, the authors also enlisted the crucial factor in selecting a DC in the cloud environment. They are financial cost, DCs capability and availability, communication channels specifications, and users' requirements.

III. Problem Encoding for Genetic Algorithm

A. Chromosome Structure

Chromosomes or individuals are fundamental concepts of any evolutionary algorithm. The chromosome represents the solution structure of the algorithm. The chromosome pattern in this research defines a user base allocated to a particular DC. Table I illustrates an individual chromosome used in this paper. The size of the chromosome will be the size of the user base. Therefore, the index of the chromosome represents the user base number, and the gene in the chromosome represents the DC allocated for the user base.

TABLE I
Chromosome representation of the genetic algorithm

UB1	UB2	UB3	UB4	UB5	UB6	UB7	UB8	UB9
DC2	DC0	DC1	DC5	DC4	DC3	DC2	DC1	DC3

B. Population

Population defines the individuals or chromosomes from which the solutions can be found. Generating the initial population plays a vital role in the progress of the GA. We need to ensure proper diversity in the population. So, we will generate the initial population randomly. The randomness in the population confirms the exploration of different search spaces in cloud computing. We will also inherit the result of the closest DC policy in the initial solution. We assume that if the closest DCs are the fittest in the evolution process, they will survive toward the optimal solution search; otherwise, they will die during the evolution process.

C. Fitness Function

User satisfaction depends on the cloud environment's response time and data processing time. So, the response time and data processing time of the DCs define the efficiency of the service broker policy. In a cloud environment, different DCs have various numbers of virtual machines. Also, the transfer cost of the DC varies with distinct factors like region, number of physical machines, etc. We will consider these factors in our evolutionary algorithms and find an optimal DC for user bases. Our goal will be to optimize the DCs response time and data processing time. Let us assume our optimization or fitness function, as shown in equation 1.

$$F(R_t, D_t) = (\sum R_t + \sum D_t)/N \tag{1}$$

Here,
R_t = Response Time of each datacenter
D_t = Data Processing Time of each datacenter
N = Number of datacenters

Our fitness function is dynamic as the response time and data processing time will change with respect to time. So, our study is divided into multiple episodes, and we will find the optimal DCs for each episode

IV. Proposed Genetic Algorithm

The GA is one of the evolutionary algorithms that use the processes of biological evolution to solve problems and model evolutionary systems. GA searches for an optimal solution for a problem in a search space of chromosomes. A GA uses the following steps: initialization of populations of chromosomes, selection according to fitness, crossover to produce new offspring, and random mutation of new offspring. A GA evolves through crossover and mutation. We will use the GA to find the optimal combination of DCs and userbases. We have already discussed the initialization of the population in problem encoding. The population from every generation will go forward with a GA toward the optimal solution. "Fig. 1" shows the flowchart of the GA used in our work.

1) Selection: The selection step in the GA defines the criteria for how the individuals or chromosomes will be selected for crossover and mutation. We have used CHC elitist selection policy in this work [37]. In CHC, the offspring competes with the parents and other offspring to get a place in the population [38], [39]. We have sorted the population according to the fitness value. The best individuals or chromosomes will go for crossover to produce more fit individuals or chromosomes. Therefore, we will provide 50% of the sorted population for crossover. On the other hand, the less fit individuals will go for mutation so that minor changes can increase their fitness. "Fig. 2" visualizes our selection policy in the population. After the GA operations, the population will double. We then

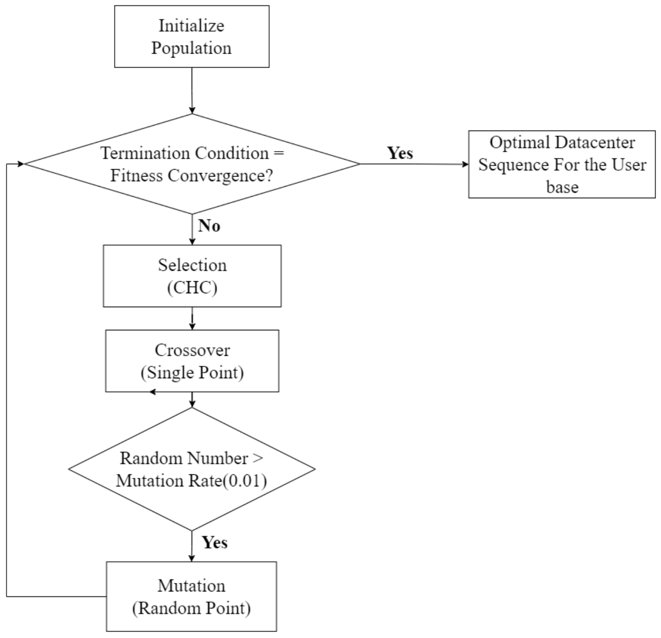
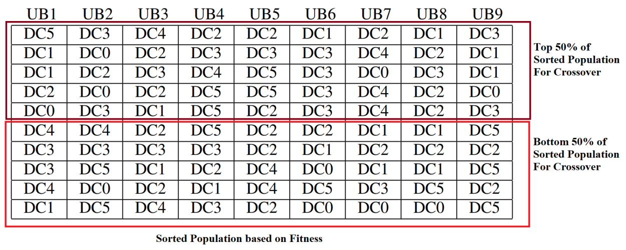

Initialize Population
Termination Condition = Fitness Convergence?
Yes
Optimal Datacenter Sequence For the User base
No
Selection (CHC)
Crossover (Single Point)
Random Number > Mutation Rate(0.01)
Yes
Mutation (Random Point)

Fig. 1. Flow Chart of Genetic Algorithm

sort the population again and remove the bottom half of the population. In this way, we maintain the original population size. CHC selection is a highly elitist selection procedure. The high-fitness individuals are kept in the population during the search.



UB1	UB2	UB3	UB4	UB5	UB6	UB7	UB8	UB9
DC5	DC3	DC4	DC2	DC2	DC1	DC2	DC1	DC3
DC1	DC0	DC2	DC3	DC3	DC3	DC4	DC2	DC1
DC1	DC2	DC3	DC4	DC5	DC3	DC0	DC3	DC1
DC2	DC0	DC2	DC5	DC5	DC3	DC4	DC2	DC0
DC0	DC3	DC1	DC5	DC2	DC3	DC4	DC2	DC3
DC4	DC4	DC2	DC5	DC2	DC2	DC1	DC1	DC5
DC3	DC3	DC3	DC3	DC2	DC1	DC2	DC2	DC2
DC3	DC5	DC1	DC2	DC4	DC0	DC1	DC1	DC5
DC4	DC0	DC2	DC1	DC4	DC5	DC3	DC5	DC2
DC1	DC5	DC4	DC3	DC2	DC0	DC0	DC0	DC5

Top 50% of Sorted Population For Crossover
Bottom 50% of Sorted Population For Crossover
Sorted Population based on Fitness

Fig. 2. Selection bases adopted search algorithms

2) Crossover: Crossover chooses the genes from the parent chromosomes and creates new offsprings. Crossover ensures diversity among the population. The single-point crossover is used for the GA in this work. As we outlined in the selection mechanism, the crossover is performed on the best individuals or chromosomes. Therefore, we will choose two parents randomly from the best individuals in the population. A random crossover point from 0 to gene length is selected. For the first offspring, the genes before the crossover point of the first parent and the genes after the crossover point of the second parent are used. For the second offspring, the genes after the crossover point of the first parent and the genes before the crossover point of the second parent are used. “Fig. 3” illustrates the single-point crossover of the GA in this research.

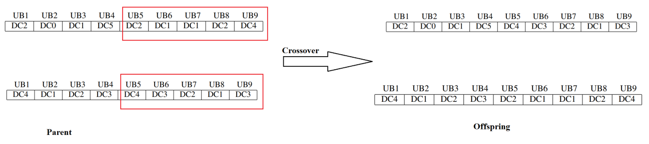

UB1 UB2 UB3 UB4 UB5 UB6 UB7 UB8 UB9
DC2 DC0 DC1 DC5 DC2 DC1 DC1 DC2 DC4
UB1 UB2 UB3 UB4 UB5 UB6 UB7 UB8 UB9
DC4 DC1 DC2 DC3 DC4 DC3 DC2 DC1 DC3
Parent
Crossover
UB1 UB2 UB3 UB4 UB5 UB6 UB7 UB8 UB9
DC2 DC0 DC1 DC5 DC4 DC3 DC2 DC1 DC3
UB1 UB2 UB3 UB4 UB5 UB6 UB7 UB8 UB9
DC4 DC1 DC2 DC3 DC2 DC1 DC1 DC2 DC4
Offspring

Fig. 3. Crossover of the Chromosomes

3) Mutation: After the crossover, mutation takes place in the GA. Mutation randomly changes the genes of the chromosomes. In this research, we will perform mutation on the lesser-fit individuals in the population. As a result, a slight change in the lesser-fit individuals may increase their fitness and make it a better solution. The mutation point is chosen randomly, which lies between 0 to userbase size. The DC at the selected mutation point is randomly replaced with another DC. “Fig. 4” exhibits the mutation process in the GA.

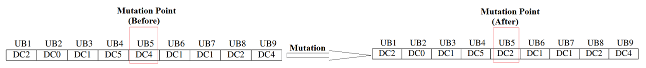

Mutation Point (Before)
UB1 UB2 UB3 UB4 UB5 UB6 UB7 UB8 UB9
DC2 DC0 DC1 DC5 DC4 DC1 DC1 DC2 DC4
Mutation
Mutation Point (After)
UB1 UB2 UB3 UB4 UB5 UB6 UB7 UB8 UB9
DC2 DC0 DC1 DC5 DC2 DC1 DC1 DC2 DC4

Fig. 4. Mutation of the Chromosome

4) Termination: The termination condition defines the number of generations the algorithm should run. In each generation of the GA, selection, crossover, and mutation are performed. The number of generations plays a significant role in the convergence of the GA. Fewer generations may lead to premature convergence of the GA. As a result, the GA will get stuck in the local minimum and unable to produce optimal results. On the other hand, an enormous number of generations will cause high computation time and maximize the algorithm’s time complexity. Therefore, an optimal number of generations is critical to ensure the best performance of the GA. The algorithm will terminate when it detects convergence or consecutively minimal changes in the fitness value.

Table II shows the most suitable value of the GA parameters that we have defined to get the best results.

TABLE II
PARAMETERS FOR GA

Parameter Name	Value
Population	50
Maximum Generation	1000
Selection Policy	CHC
Crossover Policy	Single Point Crossover
Mutation Policy	Random Point Mutation
Mutation Rate	$\frac{5}{7+\text{no of UB}+\text{no of DC}}$

A. Motivation for using Genetic Algorithm

In a cloud environment, the request keeps coming in continuously. The number of requests per hour can vary in size as well as in the number. Therefore, our service broker policy needs to be efficient and effective as time progresses. Evolutionary algorithms or meta-heuristic search comes from the concept of Darwin's evolution. These kinds of algorithms evolve with time and generate optimal solutions. This algorithm uses the natural selection process and survival of the fittest concept to progress toward the optimal solution. The algorithm goes through different generations and progresses with the best results of each generation. As a result, as time progresses, the algorithm gives more optimal solutions than other algorithms. The best example of evolution is the evolution of human beings from monkeys. We will use the concept of evolutionary algorithms in the service broker policy. The service broker policy starts with random solutions and then progresses using evolutionary strategies toward the optimal solution. The GA is one of the world's most famous and most used evolutionary algorithms. So, in this research, the GA is considered for the service broker policy.

V. PROPOSED CLOUD ENVIRONMENT

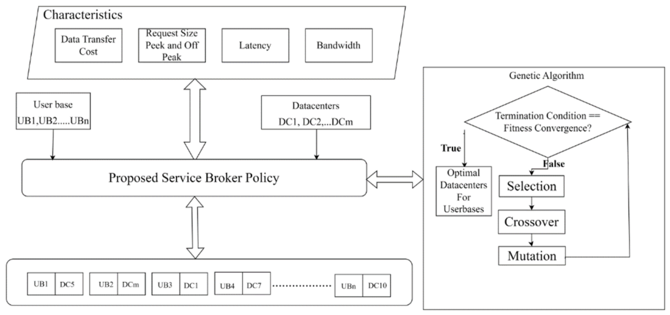


Fig. 5. Workflow of Cloud Environment

A cloud environment is a complex system with attributes such as userbase, DC, load balancing policy, and service broker policy. "Fig. 5" shows the workflow of the proposed cloud environment. The cloud components have several characteristics: data transfer cost, request size in peak and off-peak hours, latency, bandwidth, request number, etc. The service broker policy is responsible for choosing the DC for the userbase based on these attributes of the cloud. In the beginning, the user will configure the environment according to their requirements, such as

- the number of DCs needed for their requests,
- the request size,
- timing of the peak and off-peak hours,
- the number of requests at these hours.
- the DC configuration
- the load balancing policy

The user request and DC are configured in the service broker policy. The service broker policy will decide the selection of DCs for a particular userbase for a specific time based on these inputs. We have used evolutionary-based algorithms, specifically GA, to choose the optimal DCs.

VI. ENVIRONMENT SETUP

We need an appropriate environment to check the proposed algorithm's accuracy, effectiveness, and efficiency. Many cloud computing providers exist, such as Amazon Web Services (AWS), Microsoft Azure, Google, IBM Cloud, Rackspace, Red Hat, Verizon cloud, and VMware [40]. All of them provide the user with access to various configurable computing resources (servers, storage, networks, applications). These providers will only let us integrate our proposed solution once we prove the correctness of our algorithms in the cloud environment. Therefore, we need a simulation environment to simulate real-world cloud scenarios and demonstrate the algorithm's efficiency.

Cloudsim [9] is one of the first cloud simulation modeling platforms. CloudSim allows VMs to be managed by hosts, which DCs manage. The platform does not have any user interface for the users, so it is hard to configure the system according to real-world scenarios. Moreover, the output results are presented in the console, which makes it challenging to generate graphical reports.

Another simulation environment CloudAnalyst [10] platform overcomes these challenges by providing a graphical user interface for the users to configure the cloud environment according to their requirements. "Fig. 6" shows the world view of the CloudAnalyst simulator. The world in the CloudAnalyst platform is divided into six regions defined as R0, R1, R2, R3, R4, and R5. The blue dots in the map represents the userbase, and the red dots represent the DCs of the cloud services around the globe. The four main vital factors of the CloudAnalyst are userbase, DC, Load Balancing Policy, and Service broker policy. We integrated the proposed GA-based algorithm in the service broker policy.

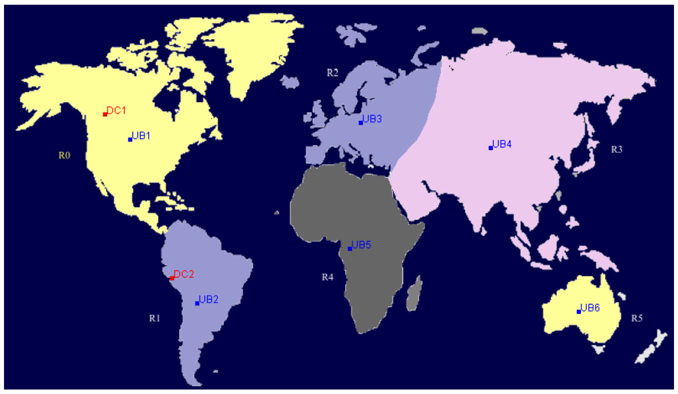


Fig. 6. CloudAnalyst Simulator

A. *Userbase Configuration:*

Userbase defines a group of users from a particular region. The user base has several properties. The user request per hour can be identical or different for each user base. Peak hours are a vital factor for the user base. Peak hours define the time when most users are active and when the cloud is processing an enormous number of requests. The average number of users during peak hours is exceptionally high compared to the average number during off-peak hours. Tables III, IV, IV show different user base configurations used in our experiment.

TABLE III
USERBASE CONFIGURATION 1 (UBC1)

Name	Region	Request Per User per Hra	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, 60,60	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

TABLE IV
USERBASE CONFIGURATION 2 (UBC2)

Name	Region	Request Per User per Hra	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

TABLE V
USERBASE CONFIGURATION 3 (UBC3)

Name	Region	Request Per User per Hra	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

B. DC configuration

The DC tries to process the userbase request effectively and efficiently. DC is a collection of physical servers, where each physical server works like a computer's CPU in the cloud. Cloud computing uses an abstraction over the physical servers named VM. The VMs can have the same configuration as physical servers or be different from the physical servers. The combined configuration of the VMs must be equal to that of the available physical servers. Creating more VMs than the available resources or physical servers is impossible. The tasks are divided among the VMs using load-balancing algorithms. The more significant number of VMs enhances the performance of the cloud environment.

The DC is configured as per the parameters defined in Table VI. In this research, we have used twenty physical servers in every DC. Each physical server is configured, as shown in Table VII. Each physical server is further abstracted with VMs. We have used 25 VMs for our experiments. They have the same memory, 1024 MB, 10000 MB image size, and 1000 Mbps bandwidth.

TABLE VI
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 VII
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

C. Load balancing Policy

In the DC, the load balancing policy plays a vital role in managing the user requests among the VMs. DCs are a collection of physical servers with identical or different configurations. Cloud computing uses abstraction over the DCs, which are called VMs. The tasks are processed on these VMs. Load balancing dynamically distributes the workload arriving at the DC among different VMs so that they are not overwhelmed with the tasks or remain idle because of no tasks. A load-balancing policy ensures the proper resource utilization of the DC. Many load balancing policies are available in the cloud services, such as First Come-First Serve (FCFS), Round Robin, Equally Spread Current Execution Load, and Throttled Load Balancing Policy. Round Robin load balancing policy distributes the tasks among the VM in the rational order of the VMs. FCFS chooses the first VM until it is loaded up with tasks and not responding anymore. The throttled load balancing algorithm marks the VM as busy or available to assign tasks. Equally Spread Current Execution load balancing algorithm equally distributes the tasks among different VMs. In this research, we have only used the round-robin load-balancing policy for all the experiments.

D. Experimental Scenarios

We have used five cloud configurations to examine the correctness of our proposed algorithm. Table VIII shows the cloud configurations used for the experiments.

VII. RESULT ANALYSIS

A. Performance of the Genetic Algorithm:

Table IX and Table X show a comparison of the average response time and data processing time between the proposed GA and traditional algorithms in different configurations. The results show that the CDC policy is the worst among all the service broker policies. The CDC policy chooses the nearest DC of the userbase. As a result, the closest DC gets overcrowded with user bases in the same region. ORT policy chooses the DC with the best response time and yields better

TABLE VIII
CLOUD CONFIGURATIONS

Configuration Name	User base 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, 2 in region 0, 1 in region 2 and 1 in region 3
Cloud Configuration 4 (CC4)	UBC1	4 DCs, 1 in region 0, 1 in region 2, 1 in region 4 and 1 in region 3
Cloud Configuration 5 (CC5)	UBC2	4 DCs, 1 in region 0, 1 in region 2, 1 in region 4 and 1 in region 3

TABLE IX
AVERAGE RESPONSE TIME FOR DIFFERENT ALGORITHMS

Configuration Name	GA	ORT	CDC
CC1	1964.10	2719.48	2752.09
CC2	1406.27	2500.77	4942.96
CC3	871.72	882.01	2429.67
CC4	741.27	812.23	2419.39
CC5	1248.74	1667.57	3901.81

results than CDC, as evidenced by Tables IX and X. But it also overcrowds the DC with the best response time. From Tables

TABLE X
AVERAGE DATA PROCESSING TIME FOR DIFFERENT ALGORITHMS

Configuration Name	GA	ORT	CDC
CC1	1574.66	2151.20	2343.70
CC2	951.24	1977.13	4205.59
CC3	513.38	662.20	2108.89
CC4	518.01	601.34	2100.54
CC5	764.71	1356.07	3514.47

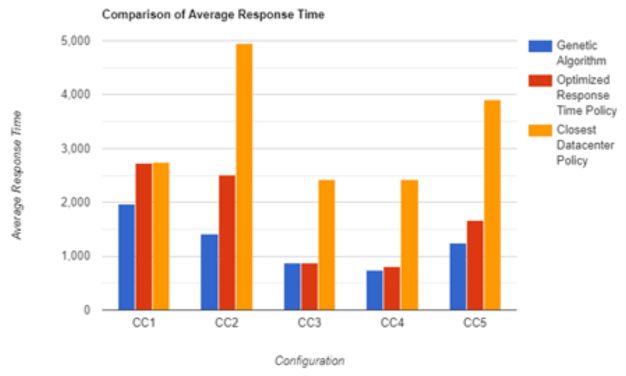


Fig. 7. Comparison of Response time with GA and traditional algorithm

IX and X, we can see the results of the proposed GA. Our proposed GA outperforms the traditional algorithms (CDC, ORT). The proposed GA provides the optimal solution for the userbase. "Fig. 7" and "Fig. 8" show that the proposed GA outperforms the CDC by a fair margin. The proposed GA performance is mostly better than ORT for most experiments. GA is an evolutionary process-based algorithm, so it searches for global optimums. In contrast, other algorithms can be stuck in local optimums, such as best response time or closest distance from the DC to userbase. The GA mutation steps help the algorithm explore different search spaces and find a global optimum. The crossover allows the GA to produce better offspring's/solutions from better parents/previous solutions.

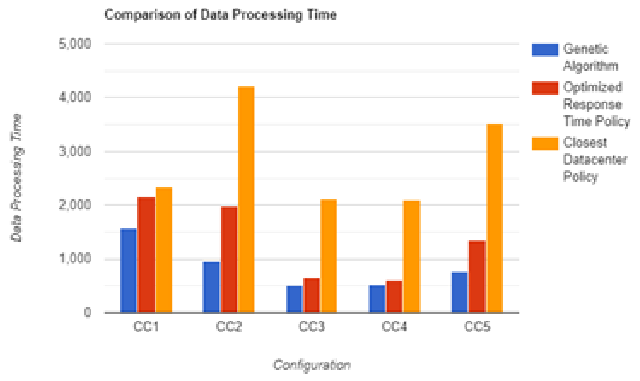


Fig. 8. Comparison of Data processing time with GA and traditional algorithm

B. Convergence of the Genetic Algorithm

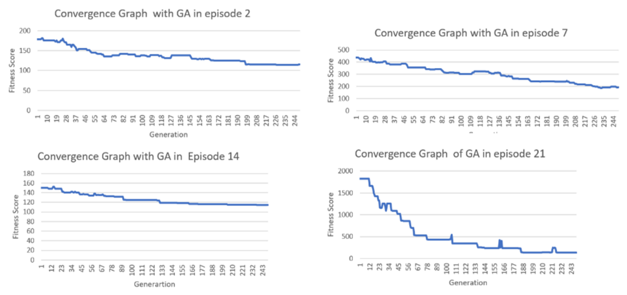


Fig. 9. Convergence of the GA in different hours of the day

The fitness function used in this research is a dynamic fitness function that changes with respect to time. The response time and data processing time are different at every hour of the day because of the difference in the number of users during peak and non-peak hours. During peak hours, the number of users increases rapidly, which overcrowds the DCs with an enormous number of user requests. So, if multiple userbases are allocated to the same DC, it will affect the algorithm's response time and data processing time. In "Fig 9", we can see the convergence graph of the 2nd, 7th, 14th, and 21st hours of the day for the proposed GA. We can see that at different hours of the day, the optimization of the GA starts from different starting points and proceeds further with selection, crossover, and mutation steps to converge the results. The objective of the dynamic fitness function used is to minimize the fitness value. The optimal fitness value would be zero. The cloud receives tasks continuously, and the DC constantly works with user requests. Therefore, the response and data processing time cannot be zero or negative. The proposed GA will terminate when it detects convergence or consecutively minimal changes in the fitness value. From "Fig 9", we can see that we get distinct convergence graphs at different hours of the day. The convergence graph shows that the GA minimizes the fitness value but cannot reach zero.

C. Execution Time of Genetic Algorithm

The time complexity of the GA is exceptionally high. So, executing the GA as soon as a new request arrives can take a considerable amount of time and decrease the performance of the cloud environment. So, we need to be careful when to start our GA for optimization. Initially, we run the proposed GA at the beginning of each hour of the day. The optimal result is stored, and after an hour, we will rerun the GA and try to find another optimal result. If the current optimal result is better than the previous optimal result, we will replace the previous result with the current result; otherwise, we will use the previous optimal result. We experiment with the GA by stopping it at different hours and using the last optimal result for the rest of the day. Table XI shows the performance of our GA and other traditional algorithms like ORT after stopping the GA at the 4th, 8th, 12th, 16th, 20th, 24th, and 48th hours. We have ignored the results of CDC as we can see that CDC cannot produce any better result in any of our cloud configuration setups in Table VIII.

TABLE XI
PERFORMANCE BY STOPPING IT AT A DIFFERENT TIME

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

Figure 10 shows the graphical comparison of the various algorithm's performance at different hours of the day. We can see that initially, the ORT performs better than the GA. With the progress of time, GA catches up with ORT. After that, the GA outperforms the ORT by a fair margin with the same configuration of the user base request. But the cloud continuously receives tasks with changing user base requirements, such as request size, peak hours, number of user requests, etc. From our simulation results, running the GA after every hour of the day is better for learning from the environment's changing parameters and searching for optimal solution

VIII. Conclusion

Many applications that we use daily are connected to the cloud directly or indirectly. The satisfaction of the cloud users relies on the service broker policy of the cloud services.

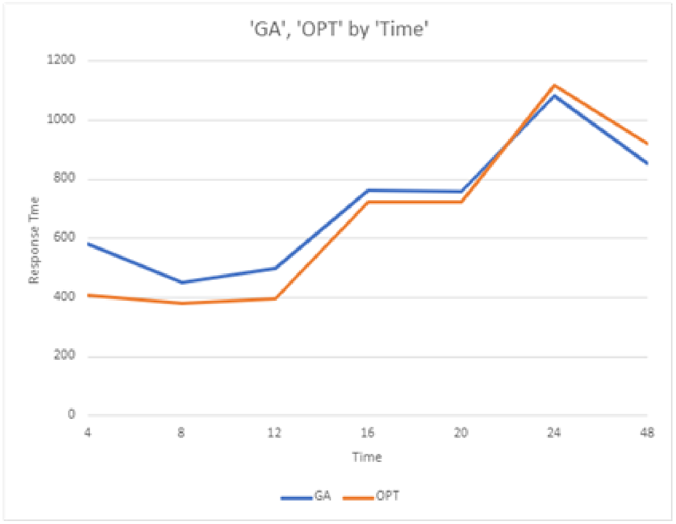


Fig. 10. Performance comparison of the algorithms by stopping it at a different time

Service broker policy ensures user satisfaction by choosing an optimal DC so that the response time and data processing time for the user request are minimal. In this paper, we have proposed a GA-based service broker policy that enhances the cloud environment's performance. GA evolves with time and goes forward to the optimal solution, which is the primary motivation behind using GA for the service broker policy.

In this research, the proposed GA-based service broker policy starts with a random initial population, injecting the solution from CDC policy into the population. The algorithm then evolves through selection, crossover, and mutation to find the optimal sequence of DC for the userbases. The exact sequence is going to be repeated for that hour. The GA will run again in the next hour to find another optimal DC sequence. If the present sequence is better than the previous sequence, the service broker policy will follow the current sequence; otherwise, it will follow the previous one. From the result analysis, the proposed algorithm performs better than the other traditional algorithms, such as ORT and CDC policy.

For the simplicity of the research, we have used only the round-robin load-balancing policy. In the future extension of this work, we will compare our algorithm with other load balancing policies such as Equally Spread Execution Current Load, Throttled, and Machine learning-based load balancing policies. The GA can also be tested with selection policies such as roulette wheel, tournament, elitism, and crossover strategies such as (P, C), (P + C), (P/R), (C), (P/R + C), where P is the parent and C is offspring or children.

References

[1] Y. Kessaci, N. Melab, and E.-G. Talbi, "A pareto-based genetic algorithm for optimized assignment of vm requests on a cloud brokering environment," in *2013 IEEE congress on evolutionary computation*. IEEE, 2013, pp. 2496–2503.

[2] T. Willber, "Impacts of cloud computing in our everyday life," Oct 2020. [Online]. Available: https://cloud-computing.tmcnet.com/breaking-news/articles/446806-impacts-cloud-computing-our-everyday-life.htm

[3] S. Chowdhury and A. Katangur, “Threshold based load balancing algorithm in cloud computing,” in *2022 IEEE International Conference on Joint Cloud Computing (JCC)*. IEEE, 2022, pp. 23–28.

[4] T. Khanom, “Cloud accounting: a theoretical overview,” *IOSR Journal of Business and Management*, vol. 19, no. 06, pp. 31–38, 2017.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[6] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, “Performance analysis of cloud computing services for many-tasks scientific computing,” *IEEE Transactions on Parallel and Distributed systems*, vol. 22, no. 6, pp. 931–945, 2011.

[7] T. Dillon, C. Wu, and E. Chang, “Cloud computing: issues and challenges,” in *2010 24th IEEE international conference on advanced information networking and applications*. Ieee, 2010, pp. 27–33.

[8] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, “A survey of mobile cloud computing: architecture, applications, and approaches,” *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, “Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[10] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, “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, 2010, pp. 446–452.

[11] P. Rekha and M. Dakshayini, “Service broker routing polices in cloud environment: a survey,” *International Journal of Advances in Engineering & Technology*, vol. 6, no. 6, p. 2717, 2014.

[12] V. Sharma, R. Rathi, and S. K. Bola, “Round-robin data center selection in single region for service proximity service broker in cloudanalyst,” *International Journal of Computers & Technology*, vol. 4, no. 2a1, pp. 254–260, 2013.

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

[14] D. Chudasama, N. Trivedi, and R. Sinha, “Cost effective selection of data center by proximity-based routing policy for service brokering in cloud environment,” *International Journal of Computer Technology and Applications*, vol. 3, no. 6, pp. 2057–2059, 2012.

[15] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, “A survey of mobile cloud computing: architecture, applications, and approaches,” *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

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

[17] D. Kapgate, “Improved round robin algorithm for data center selection in cloud computing,” *International Journal of Engineering Sciences and Research Technology*, vol. 3, no. 2, pp. 686–691, 2014.

[18] A. S. Ahmed, “Enhanced proximity-based routing policy for service brokering in cloud computing,” *Int. J. Eng. Res. Appl.(India)*, vol. 2, no. 2, pp. 1453–1455, 2012.

[19] A. Semwal and P. Rawat, “Performance evaluation of cloud application with constant data center configuration and variable service broker policy using cloudsim,” *International Journal of Enhanced Research In Science Technology & Engineering*, vol. 3, no. 1, pp. 1–5, 2014.

[20] D. Kapgate, “Weighted moving average forecast model based prediction service broker algorithm for cloud computing,” *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 2, pp. 71–79, 2014.

[21] O. M. Naha Ranesh Kumar, “Cost-aware service brokering and performance sentient load balancing algorithms in the cloud,” *Journal of Network and Computer Applications*, vol. 75, pp. 47–57, 2016.

[22] D. Arya and M. Dave, “Priority based service broker policy for fog computing environment,” in *International Conference on Advanced Informatics for Computing Research*. Springer, 2017, pp. 84–93.

[23] A. M. Manasrah, T. Smadi, and A. ALmomani, “A variable service broker routing policy for data center selection in cloud analyst,” *Journal of King Saud University-Computer and Information Sciences*, vol. 29, no. 3, pp. 365–377, 2017.

[24] J. Zhang, H. Huang, and X. Wang, “Resource provision algorithms in cloud computing: A survey,” *Journal of network and computer applications*, vol. 64, pp. 23–42, 2016.

[25] H. Patel and R. Patel, “Cloud analyst: an insight of service broker policy,” *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 1, pp. 122–127, 2015.

[26] D. Chudasama, N. Trivedi, and R. Sinha, “Cost effective selection of data center by proximity-based routing policy for service brokering in cloud environment,” *International Journal of Computer Technology and Applications*, vol. 3, no. 6, pp. 2057–2059, 2012.

[27] D. Limbani and B. Oza, “A proposed service broker strategy in cloud-analyst for cost-effective data center selection,” *International Journal of Engineering Research and Applications, India*, vol. 2, no. 1, pp. 793–797, 2012.

[28] R. K. Mishra, S. Kumar, and B. S. Naik, “Priority based round-robin service broker algorithm for cloud-analyst,” in *2014 IEEE International Advance Computing Conference (IACC)*. IEEE, 2014, pp. 878–881.

[29] B. Bhavani and H. Guruprasad, “A comparative study on resource allocation policies in cloud computing environment,” *International Journal of Advanced Computer Technology*, 2015.

[30] S. Nandwani, M. Achhra, R. Shah, A. Tamrakar, K. Joshi, and S. Raksha, “Weight-based data center selection algorithm in cloud computing environment,” in *Artificial Intelligence and Evolutionary Computations in Engineering Systems*. Springer, 2016, pp. 515–525.

[31] K. Kishor and V. Thapar, “An efficient service broker policy for cloud computing environment,” *International Journal of Computer Science Trends and Technology (IJCST)*, vol. 2, no. 4, 2014.

[32] A. M. Manasrah, T. Smadi, and A. ALmomani, “A variable service broker routing policy for data center selection in cloud analyst,” *Journal of King Saud University-Computer and Information Sciences*, vol. 29, no. 3, pp. 365–377, 2017.

[33] A. Quarati and D. D’Agostino, “Moea-based brokering for hybrid clouds,” in *2017 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2017, pp. 611–618.

[34] A. M. Manasrah, A. Aldomi, and B. B. Gupta, “An optimized service broker routing policy based on differential evolution algorithm in fog/cloud environment,” *Cluster Computing*, vol. 22, no. 1, pp. 1639–1653, 2019.

[35] M. A. Khan, “Optimized hybrid service brokering for multi-cloud architectures,” *The Journal of Supercomputing*, vol. 76, no. 1, pp. 666–687, 2020.

[36] N. A. Kofahi, T. Alsmadi, M. Barhoush, and M. A. Al-Shannaq, “Priority-based and optimized data center selection in cloud computing,” *Arabian Journal for Science and Engineering*, vol. 44, no. 11, pp. 9275–9290, 2019.

[37] S. Liu and S. J. Louis, “Comparing two representations for evolving micro in 3d rts games,” in *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2016, pp. 722–729.

[38] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[39] L. J. Eshelman, “The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination,” in *Foundations of genetic algorithms*. Elsevier, 1991, vol. 1, pp. 265–283.

[40] “Top 10 cloud computing service providers in 2017,” 2017. [Online]. Available: https://www.technavio. com/blog/top-10-cloud-computing-service-providers-2017