Using Genetic Algorithms for Load Balancing in Cloud Computing

Lejla Hodžić
University of Sarajevo
Faculty of Electrical Engineering
Sarajevo, Bosnia and Herzegovina
lhodzic1@etf.unsa.ba

Saša Mrdović
University of Sarajevo
Faculty of Electrical Engineering
Sarajevo, Bosnia and Herzegovina
smrdovic@etf.unsa.ba

***Abstract*—The cloud has become an essential part of modern computing, and its popularity continues to rise with each passing day. Currently, cloud computing is faced with certain challenges that are, due to the increasing demands, becoming urgent to address. One such challenge is the problem of load balancing, which involves the proper distribution of user requests within the cloud. This paper proposes a genetic algorithm for load balancing of the received requests across cloud resources. The algorithm is based on the processing of individual requests instantly upon arrival. The conducted test simulations showed that the proposed approach has better response and processing time compared to round robin, ESCE and throttled load balancing algorithms. The algorithm outperformed an existing genetic based load balancing algorithm, DTGA, as well.**

***Index Terms*—load balancing, genetic algorithms, cloud computing**

I. Introduction

Cloud computing is a model for enabling on-demand network access to a shared pool of configurable computing resources, such as servers and storage [1]. Due to the numerous advantages offered by such an approach, its popularity has been growing rapidly. The increasing demand for cloud services has led to certain challenges for cloud service providers. One of these challenges is the problem of load balancing, which involves distribution of the received requests among available resources.

Depending on the circumstances, load balancing can be implemented on various cloud resources, both hardware and software. Regardless of the implementation site, the goal of load balancing is creation of an effective algorithm that targets to avoid any overutilization and underutilization of resources [2]. The existence of such a load balancing algorithm can improve the delivery of cloud services in many ways. Some of these improvements are: reduction of execution cost, increase in stability, and reduction of response time. Ultimately, all the mentioned factors lead to an increase in the quality of provided services. From all of the above, it is clear that the use of a suitable load balancing strategy is extremely important for cloud computing.

There are two types of load balancing algorithms: static and dynamic. Static algorithms perform balancing based on previously known facts about the system. The current state of the system is not taken into account [3]. This makes them suitable for situations where requests are distributed evenly, and there is no excessive load on resources. Some of the frequently used static algorithms are: round robin, FCFS, and threshold algorithm.

Dynamic load balancing algorithms consider the current state of the system when making decisions. This allows them to know which resources are over and underutilized, and dynamically schedule requests accordingly [3]. Load balancing use cases in cloud computing usually imply a huge number of users who generate requests of varying demands. It is clear that these are highly unpredictable situations, in which it is necessary to use dynamic load balancers. Most of the so-called soft computing based methodologies fall under this category [4]. Soft computing refers to all techniques that use some kind of intuition when solving given problems. Some of the numerous soft computing techniques include evolutionary algorithms such as the honey bee algorithm, ant colony optimization, and genetic algorithms.

Soft computing techniques trade low computational complexity for accuracy, which means that such algorithms sometimes find feasible, but not optimal solutions [4]. However, there are many complex, real-life problems where traditional computing methods aren't applicable, and soft computing algorithms are the best choice.

Load balancing in cloud computing is one example of such a problem. Therefore, this paper presents the application of one soft computing approach, genetic algorithms, to the problem of load balancing in cloud computing. More specifically, the paper presents a genetic based algorithm for balancing user requests among virtual machines within the cloud.

II. Related work

Numerous publications have addressed the utilization of soft computing techniques for solving the problem of load balancing in cloud computing. Paper [5] by Dhinesh Babu L. D. and P. Venkata Krishna introduces an approach to solving load balancing using the honey bee algorithm. This approach models the load balancing problem based on the process through which bees locate and reap the food. Another technique that has been applied is the ant colony algorithm, which was described by R. Mishra and A. Jaiswal [6]. The paper introduced an effective load balancing algorithm that

relies on a specific pheromone update mechanism. In the context of load balancing in cloud computing, particle swarm optimization (PSO) algorithms have shown promising results. Notable examples of successful PSO-based approaches include those presented by Liu, Z., Wang, X. [7], A. Pradhan and S. K. Biso [8], and Zavieh et al. [9].

The concept of using genetic algorithms for load balancing problems was originally introduced in [10] by K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal and S. Dam. In this paper, the authors addressed the problem of scheduling N jobs on M processing units. It was assumed that each processing unit can be represented by a Process Unit Vector (PUV), while each job is described using a Job Unit Vector (JUV). Based on these vectors, the authors defined a fitness function that calculates the cost of a given allocation. Throughout the execution, the algorithm aims to minimize the value of the fitness function. Several tests conducted using the CloudAnalyst simulation tool showed that the proposed load balancer outperforms a few existing techniques, while also satisfying Quality of Service (QoS) requirements.

H. A. Makasarwala and P. Hazari [11] proposed a genetic algorithm based load balancer that considers the priority of requests. Here, the priority of the task is calculated according to the amount of time needed for it to be completed. Based on the priorities of upcoming tasks, the initialization of the population is performed. The authors also introduced a fitness function which calculates the cost of executing a single task on a specific virtual machine. The proposed approach was tested using CloudAnalyst. The results showed that it provides a better response time compared to the previously available methods.

A significant number of studies have improved load balancing approaches by combining genetic algorithms with other advanced techniques. S. Eladl, N. Ziedan and T. Gaafar [12] introduced a load balancer called DTGA, which incorporates a genetic algorithm with a throttled load balancer. The main idea proposed in the paper is to use genetic algorithm to ensure that the processing time of individual tasks is taken into account, which cannot be achieved with the throttled algorithm alone. The simulation conducted using the CloudAnalyst simulation tool demonstrated that the implemented algorithm enhances the overall response time, data center processing time and resource utilization.

Similarly, S. Dam et al. in [13] implemented a combination of genetic algorithm and gravitational emulation local search to generate a better initial population. In [14] A. Saadat and E. Masehian proposed a hybrid intelligent approach which utilizes genetic algorithms and fuzzy logic. Another improvement has been demonstrated by M. Kanthimathi and D. Vijayakumar [15], who introduced a load balancer that combines genetic and ant colony-based optimization methods.

Overall, there are numerous implementations of load balancers based on genetic algorithms that improve different aspects of cloud service quality. However, it can be observed that all these approaches operate by handling the requests in batches. In practice, it is impossible to predict the exact number of incoming requests and their arrival times. Therefore, waiting for a batch of requests is not always an optimal strategy. To address this limitation, this paper introduces an approach based on processing each request individually upon arrival.

III. Genetic algorithms

Genetic algorithms are search and optimization algorithms based on the principles of natural selection and genetics. They abstract the problem space as a population of individuals, in which they attempt to identify the best one. [16] The process of searching for the best individual is iterative, and based on continuous production of new generations of solutions. This manner of functioning enables them to be used for solving a wide range of problems.

The basic elements of genetic algorithms are [17]:

A. Population

consists of a group of chromosomes (individuals), where each chromosome represents one possible solution.

B. Chromosome encoding scheme

refers to the method used to represent a potential solution of a problem as a chromosome of a population. Each chromosome is composed of genes that represent some aspect of the problem solution. There are different mechanisms of chromosome encoding available, but the most commonly used are binary and real encoding. Binary encoding represents a potential solution as a vector of binary values, while real encoding uses a vector of real values.

C. Fitness function

the central component of genetic algorithms. It represents the evaluation mechanism of how good an individual, i.e. a potential solution, is. A higher fitness value indicates that the solution is 'fitter for a problem'.

D. Selection mechanism

in each iteration of the algorithm, the best individuals are chosen to pass their genes on to the next generations. Parent individuals are selected based on their fitness value, with those with better fitness having a higher chance of being selected.

E. Crossover mechanism

once a pair of parent individuals is selected, they undergo a crossover operation. As a result, offspring chromosomes that will form a new generation are created. The basic idea behind this operator is that parents with good fitness value will create even better offspring.

F. Mutation

mechanism that randomly changes one element of a population's chromosomes. The primary function of this operator is to maintain the diversity of generated solutions. Usually, the mutation is performed with a low probability, so as not to interfere with the convergence of the algorithm.

G. Shift mechanism

within each iteration of the algorithm, the current population is replaced by a new population generated from the best individuals in the current one. To ensure that each iteration produces a better population, the best few chromosomes of the current population can be transferred to the next population without undergoing any modifications. This strategy is called elitism.

H. Termination conditions

there are usually several termination conditions, with the number of iterations and the stagnation of the fitness value being the most commonly used one. Upon the termination, the algorithm returns the best solution found in the current population, or the best solution found during the entire execution.

It is evident that genetic algorithms are highly configurable and can be made suitable for solving the most diverse types of optimization problems. Thus, load balancing with genetic algorithms can focus on a variety of optimization factors such as energy usage, makespan, and performance, while maintaining the quality of service [18]. All of this makes them very suitable soft computing technique for solving load balancing problems.

IV. Proposed algorithm

This section provides an explanation of the proposed algorithm by describing its components.

A. Chromosome representation

The encoding of chromosomes in the implemented algorithm utilizes the permutation encoding method, as proposed in [11]. This means that each chromosome has the following form:

$$A = \{\text{VM}_{id}\}$$

where VM_{id} denotes a virtual machine identifier.

B. Population initialization

The population initialization mechanism is based on the random selection of chromosomes. As a consequence, the first iteration of the algorithm consists of the evaluation of randomly selected virtual machines.

The size of the population is dependent on the number of virtual machines (VMs) in the datacenter. For large datacenters with more than 50 VMs, 10% of available resources are considered simultaneously in one iteration of the algorithm. In other words, the number of chromosomes is equal to the total number of VMs divided by 10. This ensures that only a smaller percentage of the most suitable resources are taken into account during each iteration. For smaller datacenters with less than 50 VMs, the number of chromosomes is set to 5, which was determined experimentally. In datacenters with less than 5 virtual machines, all available machines are taken into account, meaning the number of chromosomes equals the number of virtual machines.

C. Fitness function

For determining the value of each chromosome, the fitness function proposed in [11] is used. This fitness function evaluates each possible solution based on the following formula:

$$\text{cost} = \frac{N}{MIPS} + \frac{U}{CP}$$

Here, N represents the cloudlet length, MIPS represents the processor speed in Million Instructions per Second, U represents the current load of the considered VM expressed in the number of tasks currently executed on it, and CP represents the capacity of the VM (in terms of RAM). The fitness value of the chromosome is calculated as the inverse of the cost.

D. Selection mechanism

For selection of the chromosomes that will generate the next population, a roulette wheel selection mechanism is used. The main feature of this selection mechanism is that it gives a higher chance of selecting individuals with higher fitness values [17].

E. Crossover operator

The intermediate recombination operator is employed for generating offspring chromosomes. This operator selects the values of child chromosome elements around or between the values of parent chromosome elements. The parameter d, which controls the range of possible offspring values, is set to 0.25 [17].

F. Mutation operator

For the mutation operator, random modification of the chromosome's elements was used. The parameters of this operator are set to allow for the movement of the cloudlet's execution to the two nearest neighboring machines. The mutation operator is applied to a randomly chosen chromosome, with a probability of 0.05.

G. Elitism

To ensure that the best individuals are carried over to the next generation, the elitism of the 2 best individuals is incorporated.

H. Termination conditions

The genetic algorithm terminates when one of the following termination conditions is met:

1) The best fitness value in the current population hasn't changed since the last iteration
2) There is a chromosome in the population representing a VM with a cost of 0 (meaning the cost cannot be further reduced)
3) The number of iterations reaches 10

I. Algorithm flow

The steps of the proposed algorithm are shown in 1:

Algorithm 1: Proposed genetic algorithm

1 Create the initial population by randomly choosing n virtual machines
2 Calculate the fitness value of each chromosome* in the population
3 Check if any of the termination conditions are met. If yes, return the virtual machine with the highest corresponding fitness value (**END**)
4 n - 2 times do:
(I) Select 2 chromosomes using roulette wheel selection
(II) Create a new chromosome by crossing the selected chromosomes
5 Mutate a randomly chosen chromosome
6 Form a new population by combining newly generated chromosomes and 2 best chromosomes from the current population
7 Replace the current population with the new population
8 Calculate the fitness of the new population
9 Return to the step 3

*here, each chromosome represents one virtual machine

The flowchart of the described algorithm is shown in Figure 1.

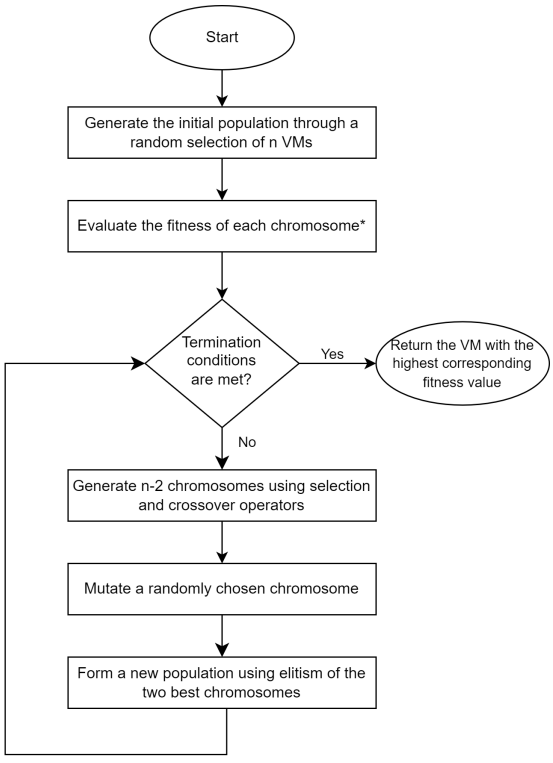

Fig. 1. Flowchart of the proposed algorithm

V. Results

A. Simulation Tool

In order to evaluate the performance of the proposed algorithm, the CloudAnalyst simulation tool was used. CloudAnalyst is an open-source GUI based simulation tool, built on top of the CloudSim framework. This tool allows specifying the detailed descriptions of data centers, user bases and internet characteristics. By doing so, the algorithm can be tested in highly dynamic and demanding contexts.
Some of the main features of this tool include [19]:

- the ability to define highly configurable and flexible simulations
- repeatability of experiments
- graphical output
- use of consolidated technology and ease of extension

CloudAnalyst has three built-in load balancers: round robin, equally spread current execution and throttled load balancer. In order to test the proposed algorithm within this simulation tool, it had to be modified. Specifically, the tool was extended by implementing a new load balancer that works on the basis of the proposed genetic algorithm.

B. Simulation Setup

The performance of the algorithm was evaluated through two different test scenarios. The main objective of this evaluation was to compare the performance of the proposed algorithm with that of one of the existing GA-based load balancers. For that reason, the simulation defined in [12] was taken as the first test scenario.

The scenario involves four data centers and six user bases, each located in a different geographical region. The data centers are equipped with identical physical hardware, with a different number of virtual machines built on top of it. All user bases are set to generate the same amount of requests, with identical request sizes. The routing protocol used in the simulation is 'closest data center', and the simulation time is set to 60 minutes. A more detailed description of the simulation can be found in the original paper.

The proposed algorithm utilizes a fitness function that considers the specifications of available resources. To evaluate the suitability of such a fitness function, the algorithm was tested in a context where data centers have varying resources. This was achieved by modifying the first scenario so that each data center has two types of hardware. These types differ in processor speed, which can be either 10000 or 2000 MIPS. On top of this hardware, two types of virtual machines are built:

TABLE I
VIRTUAL MACHINES SPECIFICATIONS

Type I VM	
Image Size	10000
Bandwidth	1000
Memory	512

Type II VM	
Image Size	100
Bandwidth	100
Memory	64

Each data center is configured to have equal quantities of both hardware and virtual machine types.

C. Results

Each test scenario was carried out using the proposed algorithm and the built-in load balancers.

The performance was measured using two metrics: overall response time, which reflects the time it takes for a request to be completed, and data center processing time, which measures the amount of time the data center spends on processing the requests. The test scenarios were conducted multiple times to ensure robustness and reliability. For each test scenario, the displayed results represent the average metric values.

The results of the first test scenario are shown in the table II, along with the metrics of the DTGA algorithm from [12]. These metrics were obtained from the original paper [12].

TABLE II
SCENARIO I SIMULATION RESULTS

	Round Robin	Throttled	ESCE	DTGA	Proposed GA
Overall response time (ms)	1374.23	1408.75	1384.85	1372.02	**1368.56**
Data center processing time (ms)	1256.76	1291.37	1267.42	1254.49	**1251.10**

The presented results show that the proposed algorithm outperformed other evaluated algorithms in terms of both overall response time and data center processing time. A visual representation of the comparison among these algorithms is show in the figure 2.

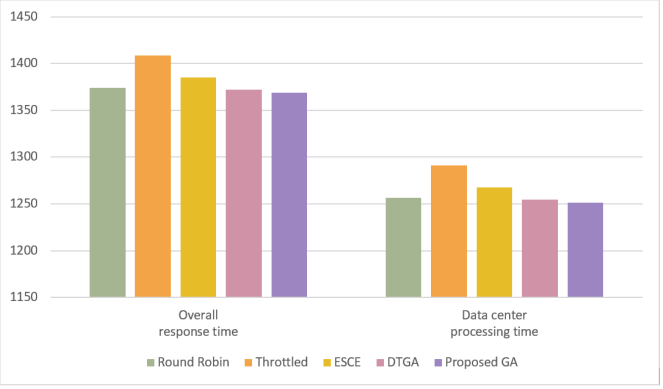

Fig. 2. Scenario I - Comparison of overall response time and data center processing time

Compared to the best built-in load balancer and the DTGA algorithm, the proposed algorithm reduces overall response time by 5.67 ms and 3.46 ms, respectively. At the same time, the datacenter processing time is reduced by 5.66 ms and 3.39 ms, respectively. The results indicate that the implemented algorithm achieves a 2.21 times greater difference in overall response time and a 2.3 times greater difference in data center processing time compared to the DTGA algorithm.

The results of the second test scenario are shown in the table III.

TABLE III
SCENARIO II SIMULATION RESULTS

	Round Robin	Throttled	ESCE	Proposed GA
Overall response time (ms)	2243.12	1538.55	2213.41	**1383.90**
Data center processing time (ms)	2125.44	1420.95	2096.40	**1266.47**

As in the previous scenario, it is possible to observe that the best results are achieved by the algorithm proposed in this paper. However, in this scenario, a considerably larger difference in performance was achieved. The observed difference is shown in the figure 3.

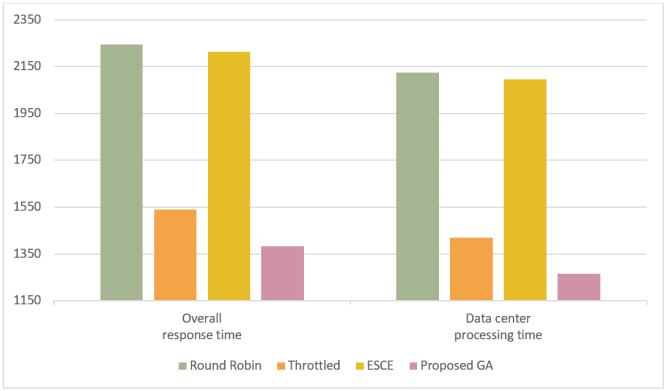

Fig. 3. Scenario II - Comparison of overall response time and data center processing time

It can be seen that the proposed algorithm achieved an overall response time that is 154.65 ms shorter than the response time achieved by the best built-in load balancer. Furthermore, the algorithm shortened the data center processing time by 154.48 ms, compared to the performance achieved by the same built-in load balancer. In conclusion, by using the proposed algorithm, overall response time and data center processing time of the test scenario are reduced by 10.05% and 10.87%, respectively.

VI. FUTURE WORK

Addressing requests individually can introduce additional overhead in situations when requests are arriving simultaneously or very frequently. In such situations, a better approach would be to distribute all the received requests across the available virtual machines simultaneously. This can be achieved by an algorithm that processes all requests that arrive within a short time interval together. In that case, depending on the frequency of requests, some requests would be processed as a group, and some individually. Unfortunately, due to the limitations of currently available simulation tools, it was not possible to implement the described algorithm in this paper. However, the idea remains to be implemented in the future.

VII. Conclusion

This paper proposes a genetic algorithm for load balancing user requests within the cloud. The main idea behind the proposed algorithm is to address requests individually, immediately upon arrival. Experimental results have shown that this approach leads to a reduction in both overall response time and data center processing time. The approach is particularly beneficial in contexts where data centers have varying resources, which was confirmed by the achieved 10% speedup of response and processing time. The proposed algorithm can be used for optimizing other aspects of cloud services as well. This can be achieved by redefining the fitness function of the algorithm.

Acknowledgment

References

[1] P. Mell, "The NIST Definition of Cloud Computing", Commerce Department, National Institute of Standards and Technology (NIST), 2011, 800-145

[2] S. Dalal, S. Kukreja, "Genetic Algorithm based Novel approach for Load Balancing problem in Cloud environment", 2016 International Journal of Computer Science and Information Security (IJCSIS), Pittsburgh, Pennsylvania, USA, 2016

[3] H. Shoja, H. Nahid and R. Azizi, "A comparative survey on load balancing algorithms in cloud computing," Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT), Hefei, China, 2014, pp. 1-5

[4] Sreeraj S., Josna V. R., "Improved genetic algorithm for load balancing in cloud computing", 2019 International Journal of Advances in Computer Science and Cloud Computing, Kerala, India, 2019

[5] Dhinesh Babu L.D., P. Venkata Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments", Applied Soft Computing, Volume 13, Issue 5, 2013, Pages 2292-2303

[6] Mishra, Ratan and Jaiswal, Anant, "Ant colony Optimization: A Solution of Load balancing in Cloud. International journal of Web & Semantic Technology". 3. 10.5121/ijwest.2012.3203., 2012

[7] Liu, Z., Wang, X. "A PSO-Based Algorithm for Load Balancing in Virtual Machines of Cloud Computing Environment". In: Tan, Y., Shi, Y., Ji, Z. (eds) Advances in Swarm Intelligence. ICSI 2012. Lecture Notes in Computer Science, vol 7331. Springer, Berlin, Heidelberg

[8] Arabinda Pradhan, Sukant Kishoro Bisoy, "A novel load balancing technique for cloud computing platform based on PSO", Journal of King Saud University - Computer and Information Sciences, Volume 34, Issue 7, 2022, Pages 3988-3995

[9] Zavieh, H., Javadpour, A., Li, Y. et al. "Task processing optimization using cuckoo particle swarm (CPS) algorithm in cloud computing infrastructure". Cluster Comput 26, 745–769, 2023

[10] Kousik Dasgupta, Brototi Mandal, Paramartha Dutta, Jyotsna Kumar Mandal, Santanu Dam, "A Genetic Algorithm (GA) based Load Balancing Strategy for Cloud Computing", Procedia Technology, Volume 10, 2013, Pages 340-347

[11] H. A. Makasarwala and P. Hazari, "Using genetic algorithm for load balancing in cloud computing," 2016 8th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Ploiesti, Romania, 2016, pp. 1-6

[12] Eladl, Shymaa and Ziedan, Nesreen and Gaafar, Tamer. (2019). Cloud Computing Load Balancing using Genetic and Throttled Hybrid Algorithm. International Journal of Engineering and Technology. 11. 606-626

[13] S. Dam, G. Mandal, K. Dasgupta, and P. Dutta, "Genetic algorithm and gravitational emulation based hybrid load balancing strategy in cloud computing," in Computer, Communication, Control and Information Technology (C3IT), 2015 Third International Conference on, 2015, pp. 1-7

[14] A. Saadat and E. Masehian, "Load Balancing in Cloud Computing Using Genetic Algorithm and Fuzzy Logic," 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2019, pp. 1435-1440

[15] M. Kanthimathi and D. Vijayakumar, "An Enhanced Approach of Genetic and Ant colony based Load Balancing in Cloud Environment," 2018 International Conference on Soft-computing and Network Security (ICSNS), Coimbatore, India, 2018, pp. 1-5

[16] Lingaraj, Haldurai. (2016). A Study on Genetic Algorithm and its Applications. International Journal of Computer Sciences and Engineering. 4. 139-143.

[17] S. Konjicija, "Heuristički algoritmi", Elektrotehnički fakultet Univerziteta u Sarajevu, ISBN 978-9958-629-54-9, Sarajevo, 2013

[18] Arshiya and J. Singh, "Genetic Approach based Optimized Load Balancing in Cloud Computing: A Performance Perspective," 2022 9th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2022, pp. 814-819

[19] B. Wickremasinghe, R. N. Calheiros and R. Buyya, "CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications," 2010 24th IEEE International Conference on Advanced Information Networking and Applications, Perth, WA, Australia, 2010, pp. 446-452