

Artificial Bee Colonies Solution for Load Sharing in a Cloud RAN

DEUSSOM DJOMADJI Eric Michel

College of Technology, University of Buea, Buea, Cameroon

NDJE BIHOLONG Evelyne Noelle

National Advanced School of Posts

Telecommunications and Information and

Communication Technologies, University of Yaoundé I, Yaoundé, Cameroon

TONYE Emmanuel

National Advanced School of Engineering of Yaoundé

University of Yaoundé I, Yaoundé, Cameroon

ABSTRACT

The integration of the cloud in the access network architecture, through the Cloud RAN, allows to increase the network capacity thanks to the virtualization of the base station and the sharing of cloud resources. The great innovation of this concept is the BBU Pool present in the architecture. It is the pool that allows the term "centralized" to be used. The processing that is carried out there is critical for the operation of the entire network. Indeed, in a cloud based environment such as the C-RAN, it is important that the resources are used optimally. This is where load-balancing comes in, which is defined as a technique used to evenly distribute workloads across multiple servers or other computing resources to optimize network performance, reliability and capacity. This paper proposes a load balancing algorithm in the BBU pool of a C-RAN. It is based on the artificial bee colony metaheuristic (ABC algorithm) and on the Load Balanced Min-Min Algorithm (LBMM). All this is done to improve network performance by guaranteeing high availability of centralized and shared resources. Its effectiveness in terms of performance and execution time against Round Robin and MIN MIN algorithms is studied and the results are appreciable.

Keywords: Cloud RAN, BBU pool, load-balancing, ABC (Artificial Bees Colony) algorithm, LBMM algorithm.

INTRODUCTION

With the continuous evolution of networks, the various players in the electronic communications sector are concerned about having networks capable of providing access to more elaborate services such as videoconferencing, e-learning, e-Commerce, e-banking, video-on-demand and embryonic but already highly demanded services such as virtual reality, augmented reality, v2x service, industry 4.0. The figures predict a 10 to 100 times increase in traffic between 2020 and 2030; the multiplication of the number of devices and services; the demand for improved user experience. The number of devices connected to the Internet is expected to reach 50 billion by 2025. [1]

To cope with this increase, mobile network operators need to become more resourceful in increasing the capacity of their networks. Densification of network equipment has been proposed as a solution to this problem. But this alternative is very costly as it requires the implementation of new access points, and a relatively complex reconfiguration in the network operation (power adjustment, cluster frequency scheme planning, traffic aggregation equipment upgrade...).

With the integration of technological concepts such as cloud-RAN, operators are working to solve the problem of increasing network capacity in order to contain the exponential growth of data traffic.

The C-RAN architecture, illustrated in Figure 1, redefines radio access by separating the LTE interface into two units: the Remote Radio Head (RRH), which is the cell, and the BaseBand Unit (BBU), which is the "intelligent" part of the access network. The different BBUs are centralized and virtualized in a "cloud" called the BBU pool, which is located at a certain distance from the RRH. The two C-RAN units are then interconnected through a Fronthaul Network (FH). [2]

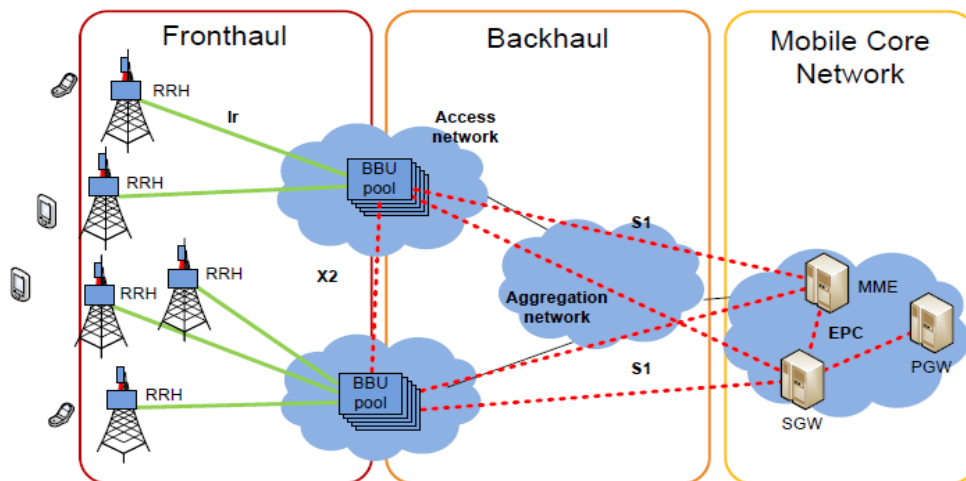


Figure 1 : 4G network with Cloud RAN access

However, this innovative vision of access networks presents technical implementation challenges such as the limited capacity of the fronthaul network, the allocation of resources, the grouping of cells, security, etc. It is in this context that our work is integrated, with the objective of proposing an approach to improve the performance of the C-RAN network, particularly in terms of resource sharing within the centralized BBU. This approach is the implementation of a load balancing algorithm at the component level of the BBU Pool.

A load balancing algorithm aims to increase the utilization of low-load or idle resources, thereby freeing up high-load resources. The algorithm tries to distribute the load among all available resources. At the same time, it aims to minimize the makespan with the efficient use of resources. [3]

MATERIALS AND METHOD

The core concern of telecom operators is to efficiently respond to the continuous growth of user data traffic. To answer this concern, it is necessary to increase the capacity of the network. The immediate solution is to increase the density of the equipment in the network, but this raises an obvious problem: the current architecture of access networks wastes processing resources and their densification generates high capital and operating costs.

Indeed, the standard architecture of a base station consists of a radio module statically connected to a processing module or baseband unit (BBU), which makes it difficult to offload traffic from the busiest stations to less busy stations. Hence the idea of virtualizing these processing modules and centralizing them within the BBU pool in order to manage the sharing of processing loads between stations more easily.

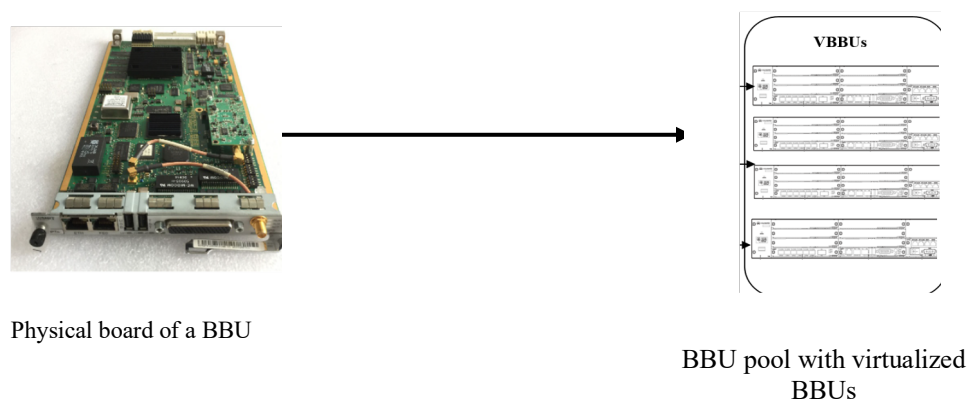


Figure 2 : migration to centralized RAN

MATERIALS

Metaheuristics are iterative algorithms, having a random component and traversing the search space via different solution generation techniques. They will iteratively move the solutions according to different strategies in order to find an approximate value of the global solution. Generally a metaheuristic uses information provided by its own neighborhood (it will move in the direction that improves the result of the function), by a neighborhood of nearby solutions or by other good solutions already found.

It is these abilities that motivated our decision to turn to the artificial bee colony (ABC) metaheuristic in order to adapt it to the load-balancing problem; giving birth to Load Balancing based on Artificial Bees Colony (LB_ABC).

As this work is not the first to turn to load sharing solutions, it is appropriate to present some previous works among which are those that inspired our approach

LITERATURE REVIEW

- Tareghian and Bornae [4] proposed a static scheduling algorithm based on the PSO (Particle Swarm Optimization) technique. It starts with a swarm containing some particles whose

number is equal to the number of tasks. Each particle is given a random position and a random velocity. In the next step of the procedure, a fitness function is evaluated for each particle. This function is the sum of the processing time of the particles on the virtual machine, the computing power and the amount of storage required. Considering the capacity of the virtual machines available in the cloud environment, the proposed approach not only reduces the amount of memory required, but also minimizes the maximum makespan of the tasks.

- Sowmya Suryadevevra et al [5] proposed an ant colony optimization algorithm for load balancing in Grid Computing, which will decide the best resource to allocate jobs based on the resource capacity.

- Manish Gupta et al [6] describes the Artificial Bee Colony Algorithm (ABC) optimization algorithm based on the intelligent foraging behavior of the bee swarm. To assign tasks in the system in a way that optimizes the overall performance of the application. The authors proposed an efficient artificial bee colony algorithm (ABC), in which they used additional genetic algorithm (GA) mutation and crossover operators in the classical ABC algorithm to solve the task scheduling problem with the criterion of decreasing the maximum completion time.

- T.Kokilavani et al [7] proposed a new approach based on the behavior of the Argentine ant. Traditional methods try to reduce the overall response time by giving an optimized schedule. But they fail to produce a balanced schedule. Moreover, they do not share the load between the available resources. Thus, the proposed method uses the ant colony system approach to share the load among the resources for the meta-tasks. The proposed work uses one of the QoS parameters, memory, to subdivide the task and schedule it in parallel among the available resources.

- Katyal and Mishra [8] proposed a selective algorithm that uses some heuristics to decide between the two Min-Min and Max-Min algorithms so that the total task execution time is minimized. At first the execution time of each task on each machine is calculated. Then the completion time is determined by summing the execution time and the waiting time in the queue. Finally, the choice between Min-Min and Max-Min is made thanks to the calculation of the standard deviation. The comparison of this algorithm with the FIFO algorithm has shown that it is more efficient in minimizing the execution time.

- Santhosh et al [9] modified the improved Max-Min algorithm to define two new algorithms based on the average execution time. In contrast to Max-Min, the task with an execution time just above the average is selected and assigned to the resource that yields the minimum execution time. The average execution time is calculated using the arithmetic mean for independent tasks and the geometric mean for dependent tasks. Experimental comparison of these two algorithms with Max-Min and improved Max-Min showed that the proposed algorithms are more efficient in minimizing the makespan.

- NoroozOliaee et al [10] proposed a mechanism for online scheduling of tasks requiring multiple resources such as memory and CPU. The proposed approach aims at minimizing the average queuing time of the tasks while taking into account their execution times. The scheduling is performed in two phases. The first phase is triggered by the arrival of new tasks. When a new task arrives, the scheduler tries to find an available server to assign and execute this task. If the scheduler does not find any server for the task, it puts it in the queue. The second phase is triggered when a task is completed. The scheduler tries to find the task with the shortest execution time among the tasks that are already in the queue and associates it with the newly started server. Then, these tasks will be assigned to this server after allocating the necessary resources to all previous tasks with shorter execution times.

- Patel and Bhoi [11] proposed a priority-based algorithm for scheduling tasks in a cloud computing environment. It consists of three priority levels: scheduling level, resource level, and task level. The scheduling level is the goal to be achieved by the scheduler, the resource level represents the attributes available to achieve the desired goal and the last level is the task level which are the available alternatives among which the best task should be ordered first. Therefore, each task requires resources with a certain priority and the priorities of different tasks are compared with each other when scheduling them.
- Thomas et al [12] proposed a credit-based algorithm. Indeed, Cloud providers have a limited amount of resources whose utilization they want to maximize. In this context, the proposed approach is based on two parameters: user priority and task duration. A credit is assigned to each task based on its duration and priority. The task with the highest credit value is executed first.
- Himani and Sidhu [13] proposed a scheduling algorithm based on cost and deadline CBD (Cost Deadline Based). The cost is calculated according to the length of the task, its deadline and the number of processing elements required. Then, a sorting mechanism is used to decide the order of execution of the tasks. Their mapping to virtual machines is given by the Min-Min heuristic algorithm. Therefore, the proposed approach is used to minimize the missed deadlines and help define the cost in the user and provider perspectives.
- Zhao et al [14] proposed a task scheduling algorithm between the local and remote cloud based on a traffic load threshold. Indeed, when the traffic load of the local cloud exceeds a certain threshold, delay-tolerant applications should be scheduled on the remote cloud to leave more computational resources for delay-sensitive applications. To ensure better QoS, they model the local cloud as a queuing system with priority. For delay sensitive applications, they are labeled with high priority and will be executed before delay tolerant applications. The proposed approach aims to improve the QoS of MCC systems by simultaneously taking advantage of the low latency of the local cloud and the abundant resources of the remote cloud.

a) LBMM (Load Balanced Min-Min) Algorithm

The makespan is the total time required to complete all the tasks that present themselves to the BBU pool.

The LBMM algorithm is an improvement of the MIN-MIN algorithm, which relies on the execution time and completion time of each task on each available resource in the pool.

It works in two steps; first, it finds the minimum execution time of all tasks. It then schedules the task with the lowest execution time on the resource with which this time is associated. In a second step, (and this is where its improvement compared to the classical MIN-MIN comes from) it chooses the resources with high load and reallocates them to the least loaded resources.

Its interest for our solution is its task scheduling methodology. Unlike FIFO, which is based on the order of arrival of tasks, the LBMM algorithm ranks the tasks in a queue based on the makespan calculation of each of them. The work of [3] details its operation.

b) ABC (Artificial Bees Colony) Algorithm

The Artificial Bee Colony (ABC) algorithm was introduced by Dervis Karaboga in 2005 [15] and developed since 2008 by Karaboga and Basturk [16] for continuous optimization problems. It is a naturalistic population-based algorithm based on the foraging of bees.

Bees collect nectar from flower patches as a food source for the hive. The hive sends out scouts (or workers) who locate flower patches, which then return to the hive and inform other bees about the fitness and location of a food source via a dance. The scout returns to the flower patch with follower (or spectator) bees. [17]

More specifically, in this algorithm, a candidate solution to the optimization problem is represented by a food source. Each food source has a quantity of nectar that characterizes its quality (fitness).

The population of the colony is divided into three groups of bees, roaming the search space in search of food sources. These groups of bees do not represent the solutions but a set of iterations. Each group of iterations corresponds to the flight of one of the three types of bees: worker bees, spectators and explorers.

The worker bees scour the vicinity of the food sources to find a better source than the one they visited. They then share the quality of the source with the spectators. The spectators focus mainly on the best quality food sources. When a food source has been 'sufficiently explored', it is abandoned and the explorers randomly go in search of a new source. [18].

Here is the general structure of an ABC algorithm [19]:

1. Initialization

2. Repeat

3. Worker bee phase: Send worker bees to food sources and update their positions.

4. Spectator bee phase: Select nectar-rich sources based on a matching function and update the solutions.

5. Explorer bees phase: Locate the most inactive solution (site exhaustion), and replace it with a randomly generated one.

6. Until the stopping criterion is reached

METHODS

Our algorithm analogizes to the three groups of bees in their search for a food source:

- The workers or scouts (employed) are assimilated to the load estimator,
- The onlookers are represented by the scheduler,
- The explorers (scout) are replaced by the module that we have named "Explorer".

Bees measure the quality of a food source by defining the fitness value. It is defined here as the attractiveness of a source compared to another.

Table 1 similarities between ABC and CRAN entities

ABC algorithm	Correspondent in the BBU pool
Food sources	VBBUs
Workers	Load estimator
Onlookers	Scheduler
Explorer	« Explorer »
Fitness	Load of the VBBU

Flowchart

The operating logic that should lead to a load balance is fulfilled by a program that communicates in parallel with the user data and the VBBUs (Virtual Base Band Unit). It is integrated into a virtual machine dedicated to fulfilling the load balancing function: we have called it the Pool Manager. The Pool Manager fulfills three main roles:

- Efficiently distribute customers requests or network load over several servers;
- Guarantee high availability and reliability by sending requests only to available VBBUs;
- Provide the flexibility to add or subtract VBBUs based on demand.

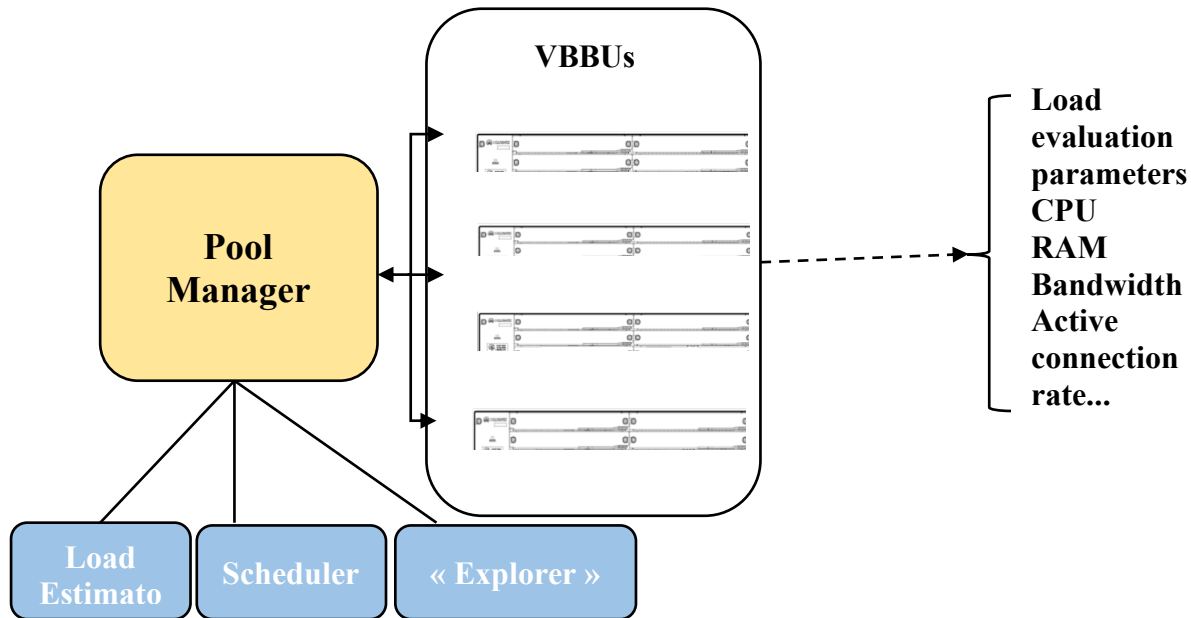


Figure 3 : Load balancer block diagram

We have identified four algorithmic steps that should lead to a load balancing between the different virtual machines.

a) Evaluation of the VBBU load

Here, the operating information of the VBBUs is collected and assembled to calculate the fitness value of each of them. Then the result is compared to a threshold value that determines the status of the container ("loaded", "eligible"); if no VBBU is active, then the choice is made randomly from the pool. This operation is called "Explorer" in reminder of the behavior of the explorer bee.

We recall that VBBUs are in fact virtual containers that logically share the elements (CPU, memory, connection ports...) of a single physical entity with very large capacities.

In this section we want to document how the load of a VBBU is evaluated.

The level of CPU load indicates the intensity with which the processor handles the programs and processes being executed. The value in question indicates the percentage of the total time used by the processor or processor core for data processing. The calculation is based on the total running time in relation to the performance time. The CPU load is therefore an indicator of how much the processor is being used and what capacity is still available. Too high a CPU load results in slow loading and saving processes. In the worst case, all programs can even come to a standstill due to an overloaded CPU. [20]

$$L_{CPU} = \frac{\text{total processing time}}{\text{operating time}} \quad (1)$$

Processors perform processing using data that they temporarily store in random access memory (RAM). Extremely fast compared to the computer's hard disk, RAM is indispensable for the smooth operation of any PC. Its load is evaluated by the following formula:

$$L_{RAM} = \frac{\text{space used}}{\text{total available space}} \quad (2)$$

The last parameter used to calculate the fitness value of a VBBU is the bandwidth usage rate. Indeed, each virtual machine in the pool is allocated a fraction of the maximum bandwidth available for the whole pool. When it executes a task, the latter consumes bandwidth; the load evaluator can thus estimate the fraction still available.

$$L_{BW} = \frac{\text{Bandwidth used}}{\text{Maximum bandwidth available}} \quad (3)$$

These criteria will be evaluated uniformly for the calculation of the fitness value in order to establish a single ranking on the score obtained and not on three distinct levels. The work of [21] admits the following formula for the calculation of the fitness value; and this is what we will also admit in the rest of the present work.

$$\text{Fitness} = \frac{1}{1 - L_{CPU}} \times \frac{1}{1 - L_{RAM}} \times \frac{1}{1 - L_{WL}} \quad (4)$$

Based on this calculation, if a parameter is heavily loaded the fitness value attached to its VBBU will also increase.

b) Costing of the task

When a user request from the network arrives at the BBU pool, information is collected to evaluate its average execution time on a VBBU. This information includes its requirements in terms of bandwidth, processor time and memory space required... This is where the MIN-MIN method comes into play. To evaluate the cost of a task is to estimate by anticipation the impact that the execution of this task will have on the fitness value of a VBBU.

c) Classification of VBBUS

Based on the previous calculations (task cost and container fitness), the scheduler makes a ranking in descending order of unloading priority and returns its results.

d) Assigning the task to a VBBU in the pool

The list of elected VBBUs is consulted to allow the Pool Manager to assign the execution of this task to one of the priority VBBUs. The assignment is made in this list by calculating the fitness value, after assignment of the task.

We orient the programming on this algorithmic basis whose flowchart is established in Figure 4

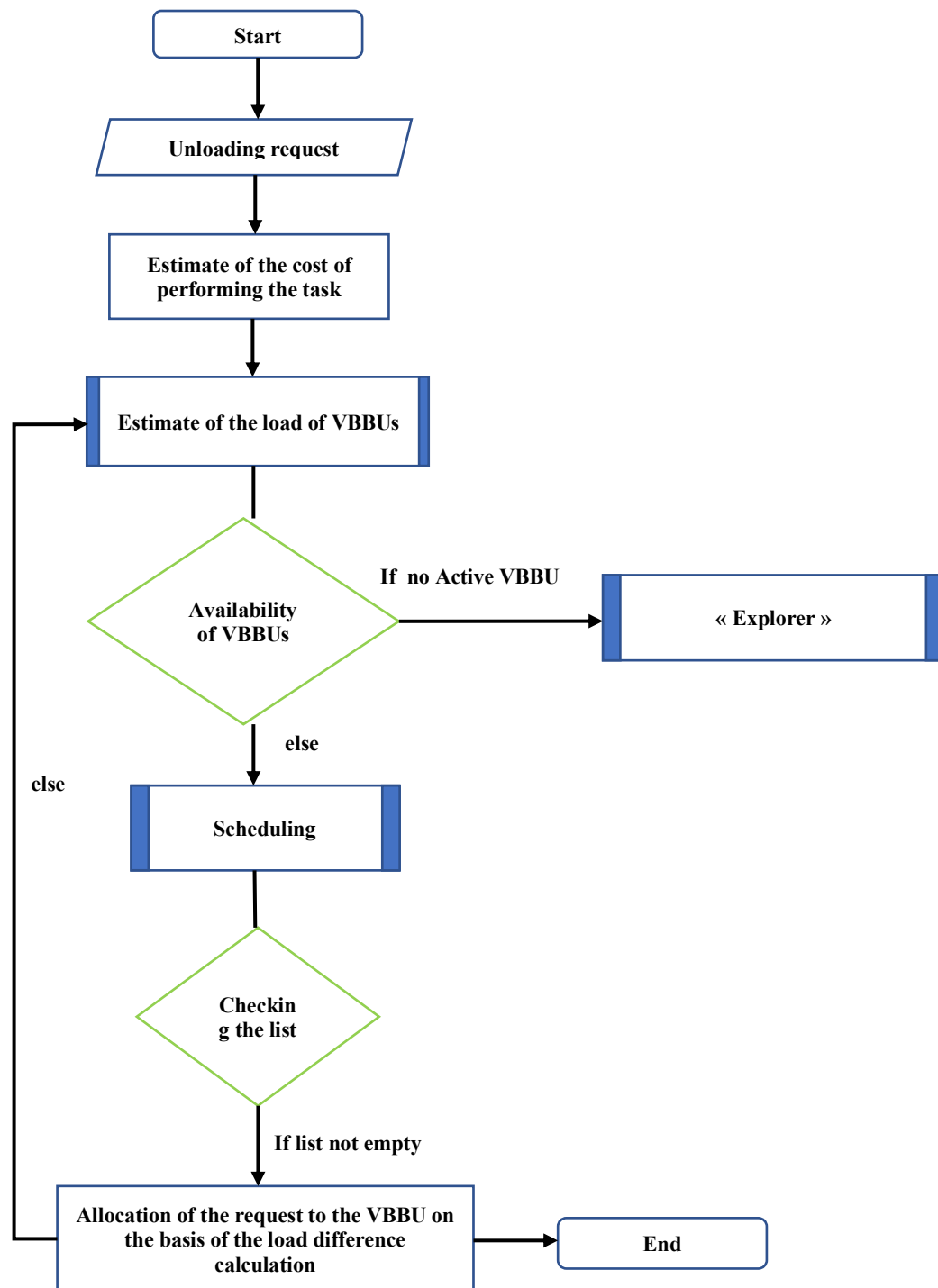


Figure 4 : Flowchart of the operation of LB_ABC

Global algorithm

Inputs:

- VBBU usage parameters: *infoCharge*
- List of VBBUs in the pool: *ListVBBU*
- Load threshold value: *Fitseuil*

Output:

- *VBBU best suited to handle the request.*

Start

1 : As long as (waiting list is not empty) do

1st step: evaluation of the load of the VBBUs (flight of the workers)

2 : $i \leftarrow 0$

3: for each VBBU(i) in ListVBBU

4: ListVBBUstatus(i) = infoCharge

5: VBBU (i).fit = CalculFit(ListVBBUstatus(i))

6: if VBBU(i).fit $\leq$ Fitseuil

7: ListVBBUwork $\leftarrow$ VBBU(i)

8: endif

9: endfor (repeat from 1 every 60 processor time units)

2nd step: generation of the best solutions (spectator theft)

10: if ListVBBUavail $\neq 0$

11: for each VBBU(i) in ListVBBUavail

12: compute the execution time of the task on the VBBU and compute the estimated fitness value after unloading

13 : $i=0$

14: bestVBBU = VBBU(i)

15: bestfit = ListVBBUavail(i).fit

16: if VBBU(i).fit $\leq$ bestfit

17: bestVBBU $\leftarrow$ VBBU(i)

18: bestfit $\leftarrow$ VBBU(i).fit

19: endif

20: endfor

21 : allocate the task to VBBU(i)

22 : update the status of the VBBUs

23 : otherwise

Flight of the explorers

24 : load the request in the load manager's queue and resume at 1

25 : endfor

26 : while

End

Summary of operation: The algorithm collects the execution parameters that we have specified for each VBBU. Then it calculates the degree of attractiveness of each of them and makes a ranking on this basis. It then evaluates the cost of executing the request on each of the VBBUs in the list of eligible ones, and chooses to assign the processing to the one that will be the least loaded once the request is executed.

RESULTS

Illustrative case of operations

Note: The value attributed to the load parameter is a kind of score that characterizes the eligibility of a VBBU and its priority compared to another. For this simulation, the maximum value is obtained for a fitness of 133.

This value corresponds to L_CPU=80% , L_RAM=85% and L_BW= 75%.

As a basis for our test and to simplify the understanding we will admit that our BBU pool is composed of 2 VBBUs (V1 and V2) with similar technical parameters (250MB of RAM and 500 Kbps of bandwidth)

We also consider that four requests form the meta task and reach the Pool Manager (R1, R2, R3 and R4).

We assume the following parameters:

For the requests, we group the expected execution times in the following table:

Table 2 requests parameters

Requests	Execution time (in units of processor time)		RAM requirement (in MB)	Bandwidth requirement (in Kbps)
	V1	V2		
R1	9	3	15	25
R2	10	5	30	50
R3	8	2	45	80
R4	12	7	75	120

For the VBBUs we group the initial charges in the following table:

Table 3 VBBUs parameters

VBBUs	Initial CPU load	Initial RAM load	bandwidth	Associated Fitness
V1	25%	35%	15%	2.41
V2	5%	17%	10%	1.41

Results of the test

For a confirmation of the improvement produced by the presented algorithm, we compared it in simulation to the Round Robin scheduling algorithm and to MIN-MIN.

Load balancing algorithms include different metrics to evaluate their efficiency [22].

- Throughput: this metric calculates the number of completed processes per unit time.
- Makespan: it measures the total time required for the system to execute a submitted meta task.
- Fault tolerance: determines the ability of the algorithm to balance the load in the event of certain failures in certain nodes or links.
- Migration time: time required to transfer a task from an overloaded node to an underloaded one.

- Modularity: Determines the ability of the system to perform a load balancing algorithm with a limited number of processors or machines.
- Performance: Measures the efficiency of the system after a load balancing algorithm has been executed.
- Energy Consumption: Calculates the amount of energy consumed by all nodes. Load balancing prevents overheating and therefore reduces energy consumption by balancing the load on all nodes.

In the context of our simulation, we will base ourselves only on two criteria to establish our comparisons:

- The response time or makespan which we recall is the maximum time that the system takes to execute the requests that it receives. Its expression is explained in the formula given by the formula II.1 of chapter 2.
- The performance which is the distribution of the CPU load between the different VBBUs once the load balancing algorithm is executed. To evaluate it, we calculate the standard deviation of the CPU load after allocation and execution of the requests. We recall its formula:

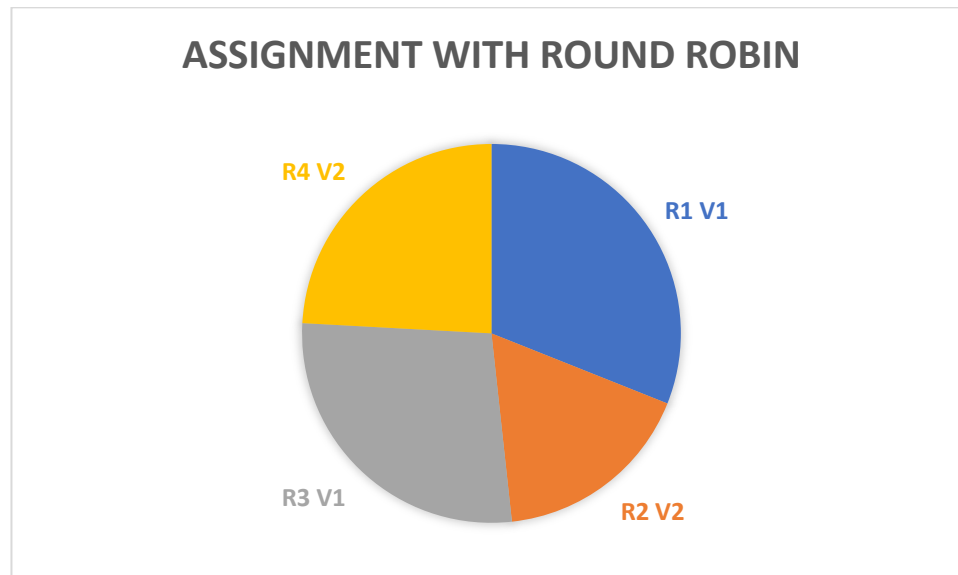
$$\sigma = \sqrt{\frac{\sum_{i=1}^n |x_i - \mu|^2}{n}}$$

With x_i a value of the series; μ the arithmetic mean of the series and n its number.

a) Results of the sharing with Round Robin

i. Assignment of tasks :

As a reminder, the allocation is done cyclically on the FCFS (First Come First Served) logic regardless of the score obtained.



ii. Results of the sharing

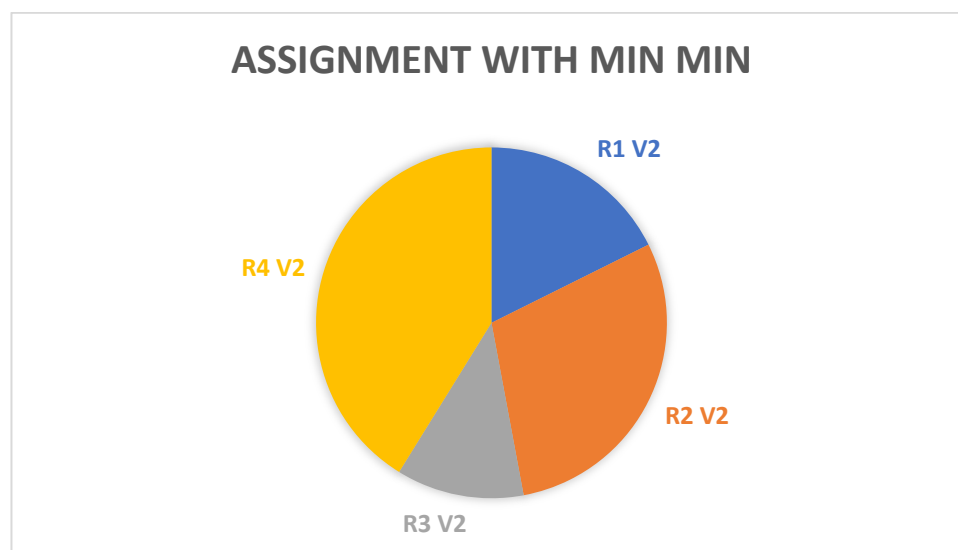
Table 4 Results of the sharing with round robin

VBBUs	CPU _i	CPU _f	RAM _i	RAM _f	BW _i	BW _f	Executed requests	fitness associated
V1	0.25	0.54	0.35	0.59	0.15	0.36	R1, R3	8.28
V2	0.05	0.25	0.17	0.83	0.1	0.44	R2, R4	5.80

b) Results of the sharing with MIN MIN

i. Assignment of tasks

As a reminder, the allocation is based on the SJF (Short Job First) logic regardless of the score obtained.



ii. Results of the sharing

Table 5 Results of the sharing with MIN-MIN

VBBUs	CPU _i	CPU _f	RAM _i	RAM _f	BW _i	BW _f	Executed requests	Fitness associated
V1	0.25	0.25	0.35	0.35	0.15	0.15	/	2.41
V2	0.05	0.34	0.17	0.59	0.1	0.65	R1, R2, R3 R4	10.56

c) Results of the sharing with LB_ABC

i. Assignement of tasks

We have seen fit to detail the algorithmic steps here in order to make explicit the execution times returned by the LB-ABC application.

- **Worker flight:** The parameters of V1 and V2 as listed in Table 3 are collected and their respective fitness values are found to be well below the threshold. They are therefore both eligible.

- **Spectator theft:** for the task in the queue we will estimate its execution cost on each of the VBBUs knowing that the LBMM algorithm has ranked the queries in increasing order of execution time. The Pool Manager will distribute the requests in the following order **R3 → R1 → R2 → R4**

1st iteration: R3 assignment

	CPU _i	CPU _f	RAM _i	RAM _f	BW _i	BW _f	fitness	Choice
V1	0.25	0.39	0.35	0.53	0.15	0.31	5.05	V2
V2	0.05	0.09	0.17	0.35	0.1	0.26	2.28	

2nd iteration : R1 assignment

	CPU _i	CPU _f	RAM _i	RAM _f	BW _i	BW _f	fitness	Choice
V1	0.25	0.4	0.35	0.41	0.15	0.2	3.53	V2
V2	0.09	0.14	0.35	0.41	0.26	0.31	2.85	

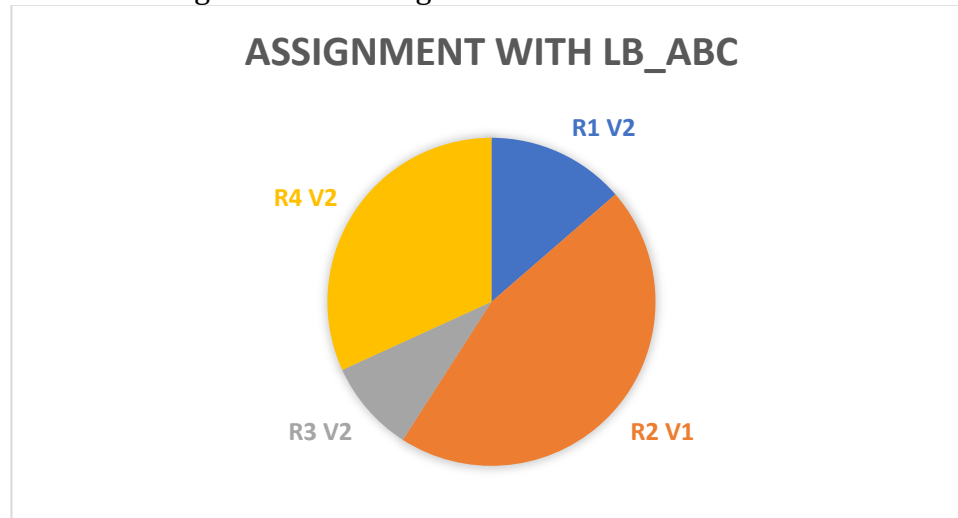
3rd iteration : R2 assignment

	CPU _i	CPU _f	RAM _i	RAM _f	BW _i	BW _f	fitness	Choice
V1	0.25	0.42	0.35	0.47	0.15	0.25	4.34	V1
V2	0.14	0.23	0.41	0.53	0.31	0.41	4.68	

4th iteration : R4 assignment

	CPU _i	CPU _f	RAM _i	RAM _f	BW _i	BW _f	fitness	Choice
V1	0.42	0.62	0.47	0.77	0.25	0.49	22.43	V2
V2	0.14	0.26	0.41	0.71	0.31	0.55	10.42	

This gives us the following attribution diagram:



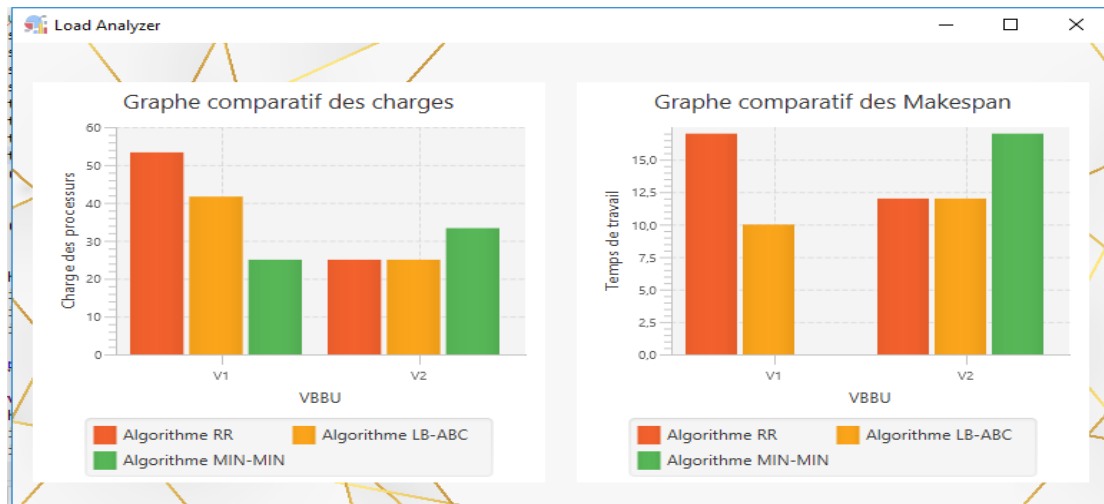
- **Flight of the explorers:** here no VBBU is loaded at the beginning of the execution so the flight of the explorers does not take place.

ii. Results of the sharing:

Table 6 Results of the sharing with LB-ABC

VBBUs	CPU _i	CPU _f	RAM _i	RAM _f	BW _i	BW _f	Executed requests	Fitness associated
V1	0.25	0.42	0.35	0.47	0.15	0.25	R2	4.34
V2	0.14	0.26	0.41	0.71	0.31	0.55	R1, R3, R4	10.42

These measurements are shown in the following graph for interpretation.



Discussion

Here are the results obtained for the load sharing of four tasks on two VBBUs V1 and V2 via the different algorithms.

algorithms	Load V1 (%)	Load V2 (%)	Performance	Makespan (in units of processor time)
Round-robin	54	25	14.5%	17
MIN-MIN	25	34	6.4%	17
LB-ABC	42	26	8%	14

	Makespan	Performance
Round Robin		
MIN-MIN		
LB-ABC		

This means that after load-balancing in Round Robin, the processors in our VBBUs have load values that are more dispersed around their average, reflecting a certain imbalance in the work of each of them. On the other hand, after a MIN-MIN or LB-ABC sharing, the processors work

relatively load-balanced, despite the fact that MIN-MIN achieves a better performance. We can therefore conclude that the LB-ABC algorithm has counterbalanced the performance with the response time, which is not the case for Round Robin and for MIN-MIN.

CONCLUSION

In the case that we have established for our study, the results obtained clearly show that our process obtains better results both for the improvement of the makespan and for the distribution of the loads on the processors. It establishes a compromise between performance and makespan minimization, where the other two are either very good in one criterion and poor in the other.

Contrary to Round Robin which shares the loads only on the basis of the order of arrival of the requests or even MIN-MIN which allocates only on the basis of the response times, the behavior of the bees simulated in LB-ABC allows to evaluate on the basis of additional criteria (CPU load, RAM load, bandwidth use)

Finally, the results obtained show the interest of our approach and its ability to improve the response time and performance of a BBU Pool during the allocation of tasks; this with the aim of optimizing the use of resources, compared to approaches such as the Round Robin algorithm and the MIN-MIN algorithm. These results are therefore satisfactory for our objectives sought.

References

- [1] «5g fifth generation of mobile technologies,» [En ligne]. Available: www.itu.int/fr/mediacentre/backgrounders/Pages/5G-fifth-generation-of-mobile-technologies.aspx. [Accès le 10 October 2021].
- [2] H. TOUATI, Allocation de Ressources et Partage de Charge adaptés aux architectures de réseaux d'accès basées sur le cloud, Evry, 2020.
- [3] K. T et A. Dr. D.I. George , «Load Balanced Min-Min Algorithm for Static Meta-Task Scheduling in Grid Computing,» *International Journal of Computer Applications*, pp. 45-47, 2011.
- [4] T. S. et B. Z, «A new approach for scheduling jobs in cloud computing environment,» *Cumhur. Sci. J.*, p. 2499-2506 , 2015.
- [5] S. Sowmya, C. Jaishri Chourasia et R. Sonam, «Load Balancing in Computational Grids Using Ant Colony Optimization Algorithm,» *International Journal of Computer & Communication Technology (IJCCT)*, 2012.
- [6] G. Manish et S. Govind, «An Efficient Modified Artificial Bee Colony Algorithm for Job Scheduling Problem,» *International Journal of Soft Computing and Engineering (IJSCE)*, pp. 2231-2307, 2012.
- [7] K. T et A. Dr. D.I. George, «Memory Constrained Ant Colony System For Task Scheduling In Grid Computing,» *International Journal of Grid Computing & Applications (IJGCA)*, 2012.
- [8] K. M et M. A, «Application of Selective Algorithm for Effective Resource Provisioning in Cloud Computing Environment,» *Int. J. Cloud Comput. Serv. Archit*, pp. 1-10, 2014.
- [9] B. S et M. D, «An Improved Task Scheduling Algorithm based on Max-min for Cloud Computing,» *Int. J. Innov. Res.*, pp. 84-88, 2015.

-
- [10] N. M, H. B, G. M et B. M, «2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS),» pp. 375-379, 2014.
- [11] P. S. J. et B. U. R, «Improved Priority Based Job Scheduling Algorithm in Cloud Computing Using Iterative Method,» *2014 Fourth International Conference on Advances in Computing and Communications*, pp. 199-202, 2014.
- [12] T. A, K. G et J. R. V, «Credit Based Scheduling Algorithm in Cloud Computing Environment,» *Procedia Comput. Sci*, pp. 913-920, 2015.
- [13] Himani et S. H. S, «Cost-Deadline Based Task Scheduling in Cloud Computing,» *2015 Second International Conference on Advances in Computing and Communication Engineering*, pp. 273-279, 2015.
- [14] Z. T, Z. S, G. X, Z. Y et N. Z, «A Cooperative Scheduling Scheme of Local Cloud and Internet Cloud for Delay-Aware Mobile Cloud Computing,» *2015 IEEE Globecom Workshops (GC Wkshps)*, pp. 1-6, 2015.
- [15] D. Karaboga, An idea based on honey bee swarm for numerical optimization, 2005.
- [16] D. Karaboga et Basturk, «On the Performance of Artificial Bee Colony (ABC) algorithm,» *Applied soft computing*, pp. 687-697, 2008.
- [17] Microsoft, «Natural algorithms use bee colony algorithms to solve impossible problems,» [En ligne]. Available: <https://docs.microsoft.com/fr-fr/archive/msdn-magazine/2011/april/msdn-magazine-natural-algorithms-use-bee-colony-algorithms-to-solve-impossible-problems#synth%C3%A8se>. [Accès le 9 August 2021].
- [18] L. Peio, Amélioration des métaheuristiques d'optimisation à l'aide de l'analyse de sensibilité, 2016.
- [19] F. HARFOUCHI, Contribution à l'optimisation par colonies d'abeilles: Application à la conception des systèmes intelligents, Boumerdes, 2019.
- [20] «Load Balancing,» [En ligne]. Available: <https://www.lemagit.fr/definition/Load-balancer-repartition-de-charge>. [Accès le 18 October 2021].
- [21] O. Chabbouh, Gestion Des Ressources Des Réseaux Cloud RAN Dans Un Contexte 5G, Tunis, 2017.
- [22] R. Saumendu, H. Dr. Md. Alam, K. Sujit, H. Nazmul et R. Md. Al Asif, «Measuring the Performance on Load Balancing Algorithms,» *Global Journal of Computer Science and Technology: B*, vol. 19, p. 47, 2019.