Abstract

1 Introduction

2 State of the Art

2.1 Scheduling Algorithm types

The key purpose of scheduling algorithms is the appropriate allocation of task or a job to the appropriate resource. [1]

The Task Scheduling algorithms can be classified as follows:

In immediate scheduling, when new tasks arrive, they are scheduled to VMs directly. In batch scheduling, tasks are grouped into a batch before being sent, known as mapping events. In static scheduling, it is based on the prior information of the global state of the system and does not consider the current state of VMs and then divides all traffic equivalently among all VMs using round robin and random scheduling algorithms. In dynamic scheduling, it considers the current state of the VMs and does not require prior information of the global state of the system and distribute the tasks according to the capacity of all available VMs. In preemptive scheduling, each task is interrupted during execution and can be moved to another resource to complete execution. In non-preemptive scheduling, VMs are not re-allocated to new tasks until finishing execution of the scheduled task. [2]

There exist various scheduling algorithms for scheduling, allocating and scaling the resources in cloud computing. These algorithms are first come first serve, round robin, min-min, max-min, earliest deadline first and greedy. In cloud computing process scheduling algorithms can be classified into two groups; Batch Mode Heuristic Scheduling Algorithm (BMHSA) and online mode heuristic scheduling algorithms. In BMHSA tasks are queued and further collected into a set as they arrive in the system and the scheduling algorithms are begins after a time slot. Example of BMHSA based algorithms are; First Come First Served (FCFS) scheduling algorithm, Round Robin (RR) scheduling algorithm, Min - Min algorithm and Max - Min algorithm. Example of online-based heuristic scheduling algorithm is Most Fit Task scheduling algorithm. [3]

In the cloud computing environment, the task scheduling algorithms are categorized into two main groups; Batch Mode Heuristic Scheduling Algorithms (BMHA) and online mode heuristic algorithms. In BMHA, Jobs are queued and collected into a set when they arrive in the system. The scheduling algorithm will start after a fixed period of time. The main examples of BMHA based algorithms are; First Come First Served scheduling algorithm (FCFS), Round Robin scheduling algorithm (RR), Min–Min algorithm and Max–Min algorithm. By On-line mode heuristic scheduling algorithm, Jobs are scheduled when they arrive in the system. Since the cloud environment is a heterogeneous system and the speed of each processor varies quickly, the on-line mode heuristic scheduling algorithms are more appropriate for a cloud environment. Most Fit Task scheduling algorithm is suitable example of On-line mode heuristic scheduling algorithm. [4]

The scheduling algorithms are divided into two basic categories; immediate mode scheduling and batch mode scheduling. In the immediate mode. task is mapped onto a resource as soon as it arrives at the scheduler. In the batch mode, tasks are not mapped onto the resources as they arrive; instead they are collected into a set that is examined for mapping at pre-scheduled times called mapping events. The independent set of tasks which is considered for mapping at the mapping events is called a meta-task. [5]

2.2 Scheduling Policies

There exist two levels of scheduling algorithms in the cloud computing environment. The First level corresponds to host level where a set of policies exist to distribute VMs in host. The Second level corresponds to the VM level where a set of policies exist to distribute tasks to VM. [2]

CloudSim proposes two levels of resource allocation policies based on two basic scheduling policies which are the time-shared and space-shared allocation policies. The space-shared and time-shared policies are depictions of the First Come First Serve and Round Robin algorithms respectively.

When space-shared policies are used for both virtual machines and cloudlets, only one task get executed at a specific time and the second one will wait for the first task to end in order to get executed. When space-shared policy is used at the virtual machines and time-shared policy is used at the task level, this policy algorithm will give each task a slice of the processing time until both tasks are finished.

When time-shared policy is used for virtual machines and space-shared policy is used for tasks, both first tasks of both virtual machines get to run simultaneously. The second task of both virtual machines will hold until the first task is executed for both VMs.

When time-shared policies are used for both virtual machines and tasks, the time of both processing units of the host will be shared simultaneously by the tasks on the virtual machines. [1]

2.3 Scheduling Algorithms

2.3.1 First Come First Serve

It is also known as First in First out. The first job will be processed first without other preferences. The author suggests that, with regards to complexity, this algorithm is the simplest one and suitable for batch systems and has relatively more waiting time [6, p. 382]

It is the simplest scheduling algorithm which serves in FIFO manner. FCFS algorithm serves those jobs which are first arrived in the queue. After completing one task, it is assigned to the next task that is at the queue. The executing task is then to eliminate from the queue. All the tasks which are queued behind need to wait for a long time for the long task to complete. [3]

It is a static task scheduling algorithm. The order of tasks in task list is based on their arriving time. The tasks are assigned to VMs as per this order. It is the most popular, simplest scheduling algorithm which depend on FIFO rule. It is also relatively less complex. On the other hand, tasks can have high waiting time because when we have large tasks in the begin task list, all tasks must wait a long time for the large tasks to finish first. [2]

Job in the queue which comes first is served. This algorithm is simple and fast. [4]

The simplest algorithm for resource scheduling is the FCFS algorithm. It is also known as First In First Out. This algorithm is based on the arrival time of the resource request. Different tasks have different burst time. The burst time is the time required by a task for its execution. The waiting times for each task are computed based on the order of tasks and their respective burst time. Thus, the average waiting time is calculated. It is observed that the tasks order changes the average waiting time for task execution. Hence, if a heavy tasks takes on the lead of the queue list, all the other small tasks will have to wait until the execution end for the leading tasks. [1]

2.3.2 Round Robin Scheduling

It is one of the oldest, simplest, fairest and most widely used scheduling algorithm. designed mainly for time sharing systems. A small unit of time known as time quantum or time slice is used. The processes or tasks or jobs are kept in a circular queue. The CPU scheduler allocates the CPU to each process for one-time quantum. The new tasks or processes are being added to the tail of the queue. In this algorithm, the CPU scheduler picks the first process from the queue, sets a timer to interrupt the process after one quantum, the process is then dispatched after one quantum and is added to the tail of the queue if the process has not been finished. If the process finishes before the time quantum, the process releases itself from the CPU voluntarily. The author suggests that the complexity and performance of the algorithm depends on the size of time quantum, takes the highest waiting time among FCFS, SJF, priority algorithm and is suitable for time sharing systems. The CPU gets allocated and preempted based on the time quantum. [6, p. 383]

It is also one of the simplest scheduling algorithms and is constructed mainly for time sharing systems. It is similar to FCFS algorithm but there exist preemption or substitution between the processes. It shares the load uniformly to all the resources. It works in a circular queue with a small unit of time called time quantum. It executes all the tasks in a sequential manner and waits until the previous task to complete and hence, there exist no starvation issue. When the queue is fully loaded and the workload is heavy, then it takes a huge amount of time to complete all the tasks and is difficult to decide on a suitable time quantum. [3]

In the round robin scheduling, the processes are dispatched in an FIFO manner but given a limited amount of CPU time called a time-slice or a quantum. If a process does not complete before its CPU-time expires, the CPU is preempted and given to the next process waiting in a queue. The preempted process is then placed at the back of the ready list. [4]

The Round Robin algorithm scheduling algorithms distributes the selected job across all available VM in a round order where each job is equally handled. It basically attempts to send the selected jobs to the available VMs in a round form. The algorithm does not require any preprocessing, overhead or scanning of the VMs to nominate the cloudlet executor. The selected VM for the current cloudlet is computed by a round robin fashion. [7, p. 255]

2.3.3 Shortest Job First

It is a scheduling technique that selects the job with the smallest execution time. The jobs are queued with the smallest execution time placed first and the job with the longest execution time placed last and given the lowest priority. The CPU is allocated to the process with the least burst time. The author suggests that, with regards to complexity, this algorithm is relatively difficult, takes less waiting time than FCFS and suitable for batch systems. The CPU is allocated to the process with least CPU burst time. [6, p. 382]

The tasks are sorted based on their priority. Priority is given to tasks based on task length and begins from smallest task to highest task. The waiting time is less when compared to that of FCFS. It has the minimum average waiting time among all task scheduling algorithms. However, this leads to situations wherein the long tasks are left waiting while small tasks are assigned first to VM for execution. This may result in long execution time and total finish time. [2]

The concept of this algorithm is almost similar to that of FCFS algorithm. The Shortest Job first chooses the task with the shortest execution time in order to take the lead of the queue which in turn reduces the average waiting time. [1]

2.3.4 Priority Based Scheduling algorithm

In this algorithm, the scheduling happens based on the priority of the processes. The highest priority jobs run first and the lowest priority jobs waits. This may lead to the starvation of the processes. The author suggests that, with regards to complexity, this algorithm is relatively difficult as the CPU gets allocated based on the priority with higher priority jobs running first. The author implies that it takes less waiting time and is suitable for both batch and time sharing systems. [6, p. 383]

In this algorithm, a priority is assigned with each process. The task with the highest priority can serve first and then the lowest priority tasks are served. For a task with the equal priority, it follows FCFS rules. This leads to the starvation of the processes. [3]

2.3.5 Min-Min

This algorithm chooses small tasks to be executed first, which in turn delays large tasks for long time. [4]

Min-Min algorithms schedule tasks by considering the execution time of the tasks on the resources. Then, the set of minimum completion times for each of the tasks exiting in *U* is found. Next, the task with the overall minimum completion time from unscheduled tasks is selected and assigned to the corresponding resource (hence the name Min-min). Last, the newly scheduled task is removed from *U* and the process repeats until all tasks are scheduled.

In Min-Min algorithm, *rj* denotes the expected time which resource *Rj* will become ready to execute a task after finishing the execution of all tasks assigned to it. First, the *Cij* entries are computed using the *Eij* (the estimated execution time of task *Ti* on resource Rj) and *rj* values. For each task Ti, the resource that gives the earliest expected completion time is determined. The task *Tk* that has the minimum earliest expected completion time is determined and then assigned to the corresponding resource. The matrix *C* and vector *r* are updated and the above process is repeated for tasks that have not yet been assigned to a resource. [5]

2.3.6 Max-Min

In MAX-MIN tasks are sorted based on the completion time of tasks; long tasks that take more completion time have the highest priority. Then assigned to the VM with minimum overall execution time in VMs list.

This algorithm exploits the available resources in an efficient manner and has better performance than FCFS, SJF and MIN-MIN algorithm. However, this increases waiting time for small and medium tasks [2]

This algorithm chooses large tasks to be executed first, which in turn delays small tasks for long time. [4]

In Max-Min algorithm, *rj* denotes the expected time which resource *Rj* will become ready to execute a task after finishing the execution of all tasks assigned to it. First, the *Cij* entries are computed using the *Eij* (the estimated execution time of task *Ti* on resource Rj) and *rj* values. For each task Ti, the resource that gives the earliest expected completion time is determined. the task *Tk* that has the maximum earliest completion time is determined and then assigned to the corresponding resource. [5]

2.3.7 Resource Aware Scheduling algorithm

RASA is a hybrid scheduling algorithm which consist of two techniques Min-Min and Max-Min. Min-Min technique is utilized to execute little tasks before huge tasks and Max-Min is used to escape the delays in vast tasks execution. [3]

In the scheduling algorithm, RASA, If the number of available resources is odd, the Min-min strategy is applied to assign the first task, otherwise the Max-min strategy is applied. The remaining tasks are assigned to their appropriate resources by one of the two strategies, alternatively. For instance, if the first task is assigned to a resource by the Min-min strategy, the next task will be assigned by the Max-min strategy. In the next round the task assignment begins with a strategy different from the last round. For instance, if the first round begins with the Max-min strategy, the second round will begin with the Min-min strategy. Alternative exchange of the Min-min and Max-min strategies results in consecutive execution of a small and a large task on different resources and hereby, the waiting time of the small tasks in Max-min algorithm and the waiting time of the large tasks in Min-min algorithm are ignored .[5]

2.3.8 Improved Max-Min algorithm

Max-min algorithm allocates task Ti on the resource Rjwhere large tasks have highest priority rather than smaller tasks. For example, if we have one long task, the Max-min could execute many short tasks concurrently while executing large one. In Max-Min algorithm, the task with maximum expected completion time is chosen to be assigned for corresponding resource that gives minimum execution time. The algorithm calculates the expected completion time of the submitted tasks on each resource. Then the task with the overall maximum expected execution time is assigned to a resource that has the minimum overall completion time. Finally, this scheduled task is removed from meta-tasks and all calculated times are updated and the processing is repeated until all submitted tasks are executed. [8]

2.3.9 Random Algorithm

The idea of random algorithm is to randomly assign the selected jobs to the available Virtual Machines (VM). The algorithm does not consider, the status of the VM if the VM is under heavy load or low load. The algorithm requires cloudlet list and VM list. The selection of VM for a cloudlet is decided using a random function. [7, p. 254]

2.3.10 Minimum Completion Time algorithm

The Minimum Completion Time job scheduling algorithm attempts to allocate the selected job to the avail-able VM that can offer the minimum completion time considering its current load. The algorithm first scans the available VMs in order to determine the most appropriate machine to perform the job. Subsequently, it dispatches the job to the most suitable VM and starts execution. The main criterion to determine the VM in the minimum completion time scheduling algorithm is the processor speed and the cur-rent load on each VM. [7, p. 259]

The minimum completion time (MCT) algorithms assign each task to the resource which results in that task's earliest completion time. [5]

2.3.11 Minimum Execution Time algorithm

The algorithms which estimate the execution time of the tasks existing in the meta-task on the resources; then assign each task to the resource with the minimum expected execution time for that task are known as Minimum Execution Time algorithm. [5]

2.3.12 Opportunistic Load Balancing algorithm

This algorithm attempts to dispatch the selected job to the available VMs which has the lowest load compared to the other VMs. The idea is to scale the current loads for each VM before sending the job. Then, the VM that has the minimum load is selected to run the job. The meaning of load here is indicated by the level of VM preoccupation with current jobs. [7, p. 259]

Opportunistic load balancing algorithm do not use the execution or completion time of the tasks and schedules the tasks in the arbitrary order. As OLB does not consider expected task execution times, the mappings it finds can result in very poor make-span. [5]

2.3.13 Longest Cloudlet Fastest Processing Element

In this algorithm the computational complexity of the cloudlets is considered while making scheduling decisions. The lengthier cloudlets are mapped to Processing Elements (PEs) having high computational power so as to minimize the makespan. In this algorithm the longer jobs finish quickly when compared with the FCFS where processing requirement of jobs are not considered while making scheduling decisions.
1. Sort the cloudlets in descending order of length.
2. Sort the PEs across all the hosts in descending order of processing power. 3. Create virtual machines in the sorted list of PEs by packing as many VMs as possible in the fastest PE.
4. Map the cloudlets from the sorted list to the created VMs. [9]

2.3.14 Shortest Cloudlet Fastest Processing Element

In this algorithm the shorter cloudlets are mapped to PEs having high computational power
1. Sort the cloudlets in ascending order of length.
2. Sort the PEs across all the hosts in descending order of processing power. 3. Create virtual machines in the sorted list of PEs by packing as many VMs as possible in the fastest PE.
4. Map the cloudlets from the sorted list to the created VMs. [9]

2.4 Meta-Heuristics techniques

The author categorizes the task scheduling algorithm into four broad categories which are traditional heuristics, meta-heuristics, hybrid heuristics and hyper-heuristics. The author grouped First Come First Serve, Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Min-Min, Max-Min and Improved Max-Min algorithms as traditional heuristics. [10]

The paper presents an extensive review of various scheduling algorithms based on five meta-heuristic techniques namely Ant Colony Optimization (ACO), Genetic algorithm (GA), Particle Swarm Optimization (PSO), League Championship Algorithm (LCA) and BAT algorithm. Different algorithms have focused on diverse optimization criteria. The paper finds that most authors focused on the reduction of make span and execution time while others focused on response time, throughput, flow time and average resource utilization. [11, p. 277]

2.4.1 Ant Colony Optimization

Ant Colony Optimization (ACO) metaheuristic is inspired by the behavior of real ants finding the shortest path between their colonies and a source of food. While walking amid their colony and the food source, ants leave pheromones on the ways they move. The pheromone intensity on the passages increases with the number of ants passing through and drops with the evaporation of pheromone. As the time goes on, smaller paths draw more pheromone and thus, pheromone intensity helps ants to recognize smaller paths to the food source. ACO methods are useful for solving discrete optimization problems that need to find paths to goals. It has been successfully applied for solving traveling salesman problem, multidimensional knapsack problem, job shop scheduling, quadratic assignment problem, scheduling of tasks in grid and cloud environments.

For scheduling of independent tasks in grid [8] or cloud, the number of ants taken is less than or equal to number of tasks. Each ant starts with an arbitrary task ti and resource Rj for processing this task. Next, the task to be executed and the resource on which it is performed are calculated by the a probable function.

Each ant builds the whole solution of assigning all the tasks to the resources. Initially, the pheromone value is set to be a positive constant and then ants change this value at the end of every iteration. The ant's solution that gives minimum value or maximum value for considered objective function is taken as the best solution of that iteration. The final optimal solution is the one which is best out of all iterations' best solution.

The algorithm is simulated in Cloudsim with the number of tasks varying from 100 to 1000. ACO is compared with Round Robin and FCFS algorithms and experimental results prove that with the increase in number of tasks, ACO takes less time than RR and FCFS. For 1000 tasks, there is approximately 29–32% reduction in makespan in comparison with RR and FCFS. [11, p. 278]

2.4.3 Genetic Algorithm

GA was first introduced by Holland in 1975 and represents a population-based optimization method based on a metaphor of the evolution process observed in nature. In GA, each chromosome (individual in the population) represents a possible solution to a problem and is composed of a string of genes. The initial population is taken randomly to serve as the starting point for the algorithm. A fitness function is defined to check the suitability of the chromosome for the environment. On the basis of fitness value, chromosomes are selected and crossover and mutation operations are performed on them to produce offspring for the new population. The fitness function evaluates the quality of each offspring. The process is repeated until sufficient offspring are created. [11, p. 281]

2.4.4 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an evolutionary computational technique introduced by Kennedy and Eberhart [64] in 1995 motivated by social behavior of the particles. Each particle is allied with position and velocity and moves through a multi-dimensional search space. In each iteration, each particle adjusts its velocity based on its best position and the position of the best particle of the whole population. PSO combines local search methods with global search methods trying to balance exploration and exploitation. PSO has gained popularity due to its simplicity and its usefulness in broad range of applications with low computational cost. [11, p. 284]

2.5 Comparative simulation analysis of task scheduling algorithms

In research paper [2] , 15 tasks of different length are taken and six VMs of different MIPS capacity are considered as follows {500, 500, 1500, 1500, 2500, 2500}. The FCFS, SJF and MAX-MIN algorithms are run on these configurations and their respective total waiting time and total finish time has been calculated for comparison. It has been observed that the total waiting time and total finish time is highest for FCFS and lowest for SJF. Hence, SJF is best in terms of total waiting time and total finish time.

In research paper [3], 5 VMs were configured with MIPS 1000, 500, 250, 250, 250 respectively and RAM size of all VM as 512 MB. The experiment is conducted for a varying number of tasks like 20, 40, 60, 80 and 100 respectively. The performance of algorithms namely First Come First Serve, Round-Robin, Resource-Aware Scheduling algorithm and priority scheduling algorithm were analyzed with respect to the following parameters which are completion time, resource utilization and throughput. It was observed that the completion time of RR algorithm is better than other algorithms. It was observed that resource utilization of genetic algorithm is better than others.

In research paper [1], The simulation configuration consisted of 1 data center with 1 host, with two PEs. Each PE has 4000 MIPS and 4 GB of RAM. 1 VM which requires 1 PE of 1000 MIPS and 1GB RAM has been initiated. The simulation aimed to evaluate the performance of FCFS and RR by varying the number of cloudlets, resources for each cloudlet, cloudlet length, cloudlet order and calculating the respective execution time and average waiting time. The average waiting time of cloudlets has been compared against the number of cloudlets for Cloudsim FCFS and SJF proposed FCFS algorithm respectively. It was observed that SJF enabled FCFS performed much better as it only took relatively less average waiting time when the number of cloudlets is increased. The average waiting time for cloudlets has been compared against the number of cloudlets for CloudSim RR and dynamic RR algorithm. It was observed that dynamic

RR algorithm performed much better as it only took relatively less average waiting time when the number of cloudlets is increased.

In research paper [7, p. 252], the author conducts a practical comparison study among four common job scheduling algorithms namely Round Robin, Random Resource selection, Opportunistic Load Balancing and Minimum Completion Time. The author used three evaluation metrics for comparing these algorithms which are namely throughput, make span and the total execution cost. The makespan represent the maximum finishing time among all received jobs. The throughput is the number of executed jobs, which is calculated to study its efficiency in satisfying the jobs dead-lines where the deadlines represents the respective finish time. The total cost is calculated based on the processing time and the amount of data transferred. The simulation was conducted for all these scheduling algorithms by analyzing their performance metrics (throughput, make span and total cost) with respect to the various number of jobs. It was observed that the throughput deteriorates for all cases when the number of jobs were increased for all scheduling algorithms. This is mainly due to the increasing number of jobs, resulting in a high load for each VM that further leads to the failure to execute some jobs. It was also noted that the minimum completion time steadily outperforms the other scheduling algorithms in all cases. The opportunistic load balancing algorithm performed better than the Round Rubin and the random algorithms in most of the cases. This is because the opportunistic load balancing algorithm attempts to distribute the jobs to the avail-able VM which had the lowest load compared to the other VMs. However, the Round Robin algorithm performance is low because it does not take into account the specific VM‘s load and handled the jobs in sequence by giving the same time portion for each job. Finally, the Random algorithm performed the worst in most of the cases compared to the other scheduling algorithms. This is because the random algorithm randomly assigns the selected jobs to the available (VM). The algorithm does not take into considerations the VM status whether it was under high or low load. It was observed that the minimum completion time has achieved the lowest value of make span in all cases compared to the other algorithms. This is mainly due to the fact that the mini-mum completion time attempts to select the most suitable VM that can rapidly respond and execute the given job. The opportunistic load balancing algorithm achieved better compared to the Round Rubin and Random algorithms. the Round Rubin algorithm performed the worse compared to the minimum completion time and the opportunistic load balancing algorithms. This is because the Round Rubin algorithm is not concerned with the given VM specifications and loads the job in a circulatory form. The Random algorithm is the worst among the other algorithms for achieving the highest makespan time. This is because the random algorithm attempts to randomly distribute the set of jobs over the VM and the job constraints is not taken into consideration. It was observed that the minimum completion time produces the highest cost in all cases compared to the other scheduling algorithms. This is mainly due to the minimum completion time accomplishing the largest number of received jobs. Thus, the total cost will be more compared with the other algorithms. The opportunistic load balancing scheduling algorithm incurred higher cost compared with the Random and Round Rubin algorithms. The Round Rubin algorithm produced less cost compared to the minimum completion time and the opportunistic load balancing algorithms. The Random algorithm is the superior in all cases in terms of the total cost compared with the other algorithms.

In research paper [12], The authors implemented the FCFS, SJF and RR task scheduling policies on the CloudSim using both the time-shared and space-shared modes of cloudlet scheduling policies. It was observed that the waiting time in RR scheduling is much less in comparison of FCFS and SJF task scheduling policy in case of time-shared policy of cloudlet. It was noted that turnaround time (time interval from the time of submission of a process to the time of the completion of the process) is less in RR in comparison of FCFS and SJF in case of space shared policy of cloudlet. In both the cloudlet policies RR task scheduling algorithm is much better than the FCFS and SJF.

In research paper [9] ,The algorithms are tested by varying the number of cloudlets from 10 to 50 and also randomly varying the length of cloudlets. Also, the number of VMs used to execute the cloudlets, are varied accordingly. The overall makespan to execute the cloudlets is used as the metric to evaluate the performance of the proposed algorithms. It has been observed that, for smaller number of tasks, all the three algorithms exhibit more or less similar performance since the length of the queued cloudlets are less. But, as the number of tasks increase, LCFP exhibits better performance when compared to SCFP and FCFS since longer tasks complete faster thereby reducing the makespan.

3. Design and Implementation of the Hybrid Model

3.1 Scheduling Heuristics

3.2 Meta-Heuristics

3.3 Optimization Model

4. Experiments and Simulation Results

4.1 Experimental Configuration

4.2 Simulation Evaluations and Results

5. Conclusion and Future work

References

[1] H. GIBET TANI and C. EL AMRANI, “Cloud Computing CPU Allocation and Scheduling Algorithms using CloudSim Simulator,” *IJECE*, vol. 6, no. 4, p. 1866, 2016, doi: 10.11591/ijece.v6i4.10144.

[2] T. Aladwani, “Types of Task Scheduling Algorithms in Cloud Computing Environment,” in *Scheduling Problems - New Applications and Trends*, R. da Rosa Righi, Ed.: IntechOpen, 2020.

[3] Nobo Chowdhury1*, K. M. Aslam Uddin1, Sadia Afrin1, Apurba Adhikary1 and Fazly Rabbi2, “Performance Analysis of Scheduling Algorithms in Cloud Computing,” 2(1): 1-6, 2018; Article no.AJRCOS.44634.

[4] Athokpam Bikramjit Singh, Sathyendra Bhat J.1, “A Comparative Study of Various Scheduling Algorithms in Cloud Computing,”

[5] Parsa, “RASA: A New Grid Task Scheduling Algorithm,” *JDCTA*, 2009, doi: 10.4156/jdcta.vol3.issue4.10.

[6] R. Kaur and S. Kinger, “Analysis of Job Scheduling Algorithms in Cloud Computing,” *IJCTT*, vol. 9, no. 7, pp. 379–386, 2014, doi: 10.14445/22312803/IJCTT-V9P169.

[7] Mohialdeen, “COMPARATIVE STUDY OF SCHEDULING AL-GROTIHMS IN CLOUD COMPUTING,” *Journal of Computer Science*, vol. 9, no. 2, pp. 252–263, 2013, doi: 10.3844/jcssp.2013.252.263.

[8] O. M.Elzeki, M. Z. Reshad, and M. A. Elsoud, “Improved Max-Min Algorithm in Cloud Computing,” *IJCA*, vol. 50, no. 12, pp. 22–27, 2012, doi: 10.5120/7823-1009.

[9] S. Sindhu and S. Mukherjee, “Efficient Task Scheduling Algorithms for Cloud Computing Environment,” in *Communications in Computer and Information Science, High Performance Architecture and Grid Computing*, A. Mantri, S. Nandi, G. Kumar, and S. Kumar, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 79–83.

[10] A. Jain and A. Upadhyay, “Cloud Scheduling using Meta Heuristic Algorithms,” *ijcse*, vol. 5, no. 10, pp. 132–139, 2017, doi: 10.26438/ijcse/v5i10.132139.

[11] M. Kalra and S. Singh, “A review of metaheuristic scheduling techniques in cloud computing,” *Egyptian Informatics Journal*, vol. 16, no. 3, pp. 275–295, 2015, doi: 10.1016/j.eij.2015.07.001.

[12] R. Pratap and T. Zaidi, “Comparative Study of Task Scheduling Algorithms through Cloudsim,” in *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, Noida, India, Aug. 2018 - Aug. 2018, pp. 397–400.