DOI: 10.51153/kjcis.v7i1.194

Power Optimized Task Scheduling using Genetic Algorithm (POTS-GA) in Cloud Environment

Sana Saleem[a], Minhaj Ahmad Khan[a*]
[a]Bahauddin Zakariya University, Multan, Pakistan
ssaleem3399@gmail.com, mik@bzu.edu.pk

*Corresponding Author: Minhaj Ahmad Khan mik@bzu.edu.pk

Abstract

In a cloud environment, the allocation of tasks has become pivotalon account ofrapid growth of user requests. The processing of user requests leads to a significant execution time, and a huge amount of power is also consumed. Consequently, task scheduling for optimizing makespan and power usage has become critical, particularly in a heterogeneous environment. This research work proposes Power-Optimized Task Scheduling using Genetic Algorithm (POTS-GA) that aims to minimize execution time and power consumption. The proposed strategy employs genetic algorithm to take scheduling decision while taking into consideration the execution time and overall power consumption of resources. The fitness computation considering both objectives and the customized genetic operators ensure to search for a better scheduling solution. The experiments performed on a large number of tasks and virtual machines show that the proposed POTS-GA approach outperforms other task scheduling strategies including Efficient Task Allocation using Genetic Algorithm (ETA-GA), Round Robin algorithm (RRA), First Come First Serve (FCFS) and Greedy algorithm in terms of makespan and power consumption.

Keywords: Cloud Computing, Scheduling, GA, Makespan, Power Optimization

1. Introduction

As a cutting-edge technology, cloud computing has revolutionized the digital realm by providing economical solutions for its users. It has changed the computing mechanism by its on-demand delivery of IT resources to its users over internet with pay-per-use model [1]. The ubiquity of cloud computing makes services available over internet from anywhere. Hence, many industries and research organizations have adopted cloud computing technologies for economic benefits [2]. Nowadays, the internet users may access computing services all over the world, without requiring them to think about the hosting infrastructure. Moreover, the capability of internet regarding provision of services through access to resources such as storage, network

and processors, is fully harnessed by cloud computing. This type of hosting architecture is made up of powerful machines that are deployed by service providers for use by their consumers. By offering services to cloud service consumers, the cloud service providers generate revenue, which makes cloud computing a widely used paradigm.

The cloud computing environment generally provides services to its users through a variety of models corresponding to the Infrastructure, Software, and, Platform [3]. Such environment can offer high performance because it distributes workloads over all resources in a fair and efficient manner, resulting in reduced execution times, waiting times, maximum throughput, and efficient resource utilization. Since the demand for cloud services is rapidly growing, the rate of growth for large-scale computing datacenters along with huge amount of high performance resources is also increased. A datacenter in a cloud environment serves as a storage infrastructure for physical resources in order to offer cloud services to clients [4] [5].

The virtual resources that are used to handle the client requests are mapped to physical machines in the cloud computing environment. The virtual machine (VM) placement algorithms in this context are used to identify the appropriate physical machines for mapping to the VMs [6] [7] [8] [9]. The user requests arranged as tasks are allocated to VMs with different computational capacities. An effective task scheduling strategy maps the tasks to appropriate VMs for minimizing overall execution time (or makespan) [10] [11] [12]. During execution of tasks, the excessive resource utilization results in increased power consumption. Furthermore, the deployment of multiple servers by cloud providers in the data centers results in the consumption of large amount of power and raises the level of CO_2 emission in the environment which is not suitable for a green cloud computing environment [13] [14] [15]. Therefore, many efforts are made to reduce power usage and emission of CO_2 through effective management of resources.

The scheduling techniques can generally be classified into static and dynamic categories based on time the user requests are generated [16]. In static scheduling, the complete information regarding the tasks and resources is known prior to execution [17]. Consequently, the decision may be performed during compilation process. The dynamic scheduling, in contrast, maps the tasks at runtime. Some scheduling techniques use heuristic methods that aim to reduce the makespan, or schedule length, which affects execution and processing times in general. Since the heuristic methods draw conclusion without large exploration, the solutions of scheduling algorithms using this approach may not attain optimality. Other scheduling techniques implement meta-heuristic algorithms that explore a large search-space in order to find an

optimal solution. Meta heuristic methods in contrast to the heuristic methods explore locally as well as globally to obtain sufficiently optimal solutions for any optimization problem. There are many meta-heuristic approaches such as Genetic Algorithm (GA) [18], Ant Colony Optimization [19], Particle Swarm Optimization (PSO)[20] and Artificial Bee Colony (ABC) [21] optimization method to optimize scheduling in the cloud environment. These scheduling strategies also aim to reduce makespan for client requests, similar to most of the heuristic strategies. Moreover, the solutions in [22] [23] aim to minimize power in cloud environments. Similarly, hybrid techniques [24] [25] incorporate heuristic and meta-heuristic approaches combined together, while attempting to improve the exploration of search space in order to determine an optimal solution. However, most of these approaches do not either aim to concurrently reduce both makespan as well as power consumption, or are unable to produce effective schedules minimizing both objectives.

In this paper, we propose Power Optimized Task Scheduling using Genetic Algorithm (POTS-GA) to schedule the tasks to VMs in the cloud environment. The proposed algorithm reduces the overall execution time of tasks and power by using fitness function that considers both objectives to search for optimal mapping of tasks to VMs. We use genetic algorithm for task scheduling while searching a large solution space. Our experiments performed with a large number of tasks and VMs demonstrate that the proposed POTS-GA algorithm outperforms other scheduling techniques in terms of makespan and power consumption.

The remaining paper is organized as follows. Section 2 describes a brief overview of the work related to task scheduling. The proposed POTS-GA algorithm is presented in Section 3. The experimental setup with system configuration and the results of evaluation are discussed in Section 4. The paper is concluded with findings and future directions in Section 5.

2. Related Work

Aimed at optimizing diverse objectives, several task scheduling strategies have recently been proposed by the researchers. Table 1 and Table 2 show a comparative analysis of the prominent research work conducted for heuristic and meta-heuristic based scheduling strategies, respectively. The comparison is performed in terms of major contribution, features and weaknesses. The scheduling strategies deploying heuristic, meta-heuristic or hybrid approaches for scheduling are summarized below.

Table 1. Comparison of prominent heuristic-based scheduling methodologies

Reference	Algorithm	Contribution	Main Features	Weakness
[26]	HEFT	It works on reduction of execution time (Makespan)	Use combination of two algorithms which are HEFT and critical path on processor (CPOP)	There is no evaluation of power consumption.
[27]	SHEFT	Improves Makespan	A directed cyclic graph(DAG) is used for weights calculation	Although there has been improvement in resource utilizationbut power consumption is not considered.
[28]	Min-Min Max Min	It improves makespan as well as resource utilization for small tasks	It implements combination of two scheduling approaches according to size of tasks	It is evaluated on small data in grid environment. without any consideration of power consumed.
[29]	TSACS	Focus on improving schedule length and load balancing.	Travelling salesman approach (TSP) is used for the selection of tasks.	There is no consideration of power consumption for data centers.
[30]	FCFS, RRA	Minimize makespan	The approach maps the tasks to VMs according to their arrival time in a queue	It shows better results in start and its performance is ceased with the passage of time. Not suitable for multiple objectives.

Table2. Comparison of prominent meta heuristic-based scheduling methodologies

Reference	Algorithm	Contribution	Main Features	Weakness
[22]	TS-GA	Reduce makespan and cost of tasks	Tournament Selection is used in Genetic Algorithm (GA) for the	The performance of the algorithm is evaluated on small dataset with no

			selection of tasks	consideration of power consumed.
[31]	MGGS	Improves execution time and resource utilization	It uses Greedy algorithm for Updation of vectorsand roulette wheel is used for the selection method.	It combines GA with greedy strategy, while the power consumption is not optimized.
[18]	ETA-GA	It reduces overall completion time of tasks and chances of failure	It has multi-objective optimization.	It converges early and consequently, the performance of algorithm drops as the number of cloudlets increases.
[32]	Multi-objective PSO	Optimize task execution time and cost	A set of dominated and non-dominated particles is kept.	For optimization, the approach does not include power consumption.

In heuristic-based scheduling algorithms, the Heterogeneous-Earliest-Finish-Time (HEFT) algorithm proposed by Topcuoglu et al. [26] is a widely used algorithm for scheduling tasks on heterogeneous systems. The tasks at each stage are selected rank-wise and assigned to the processors, while employing an insertion-based strategy to minimize the task's earliest finish time. Lin et al. [27] proposed the scalable HEFT algorithm to allocate the workflow dynamically on cloud platforms by choosing a resource that has the shortest completion time from the current free resources. The resources that remain idle from a given threshold are released elastically at run time. Etminani et al. [28] present an algorithm to improve resource utilization and makespan for grid environment. The proposed method combines the Min-Min and Max-Min scheduling algorithms in order to gain the advantages of both algorithms. The Max-Min approach is executed for large sized tasks in the queue, whereas, Min-Min algorithm works well for a large number of small sized tasks. Similarly, Gupta et al. [33] propose several variants of HEFT based on consideration of communication cost. The proposed approach is shown to improve performance of HEFT, however, the variants work with higher complexity than HEFT and are applicable to workflows having dependencies among tasks.

A Trust Aware Distributed and Collaborative Scheduler (TADCS) is proposed by Rouzaud et al. [34] that allocates the tasks on heterogeneous VMs based on the user objectives, trust and protection to optimize the power as well as to increase the performance. Elzeki et al. [35] use max-min scheduling approach in which execution time is used for the selection of tasks instead of completion time to reduce both execution and waiting time. The task with the highest execution time is mapped to resource producing the lowest completion time to improve overall response time.

An integer programming based optimization technique is presented by Liu et al. [36] with the aim to reduce energy consumption and overall processing time of the tasks. Their method sorts all servers based on energy consumption of resources, and then assigns jobs in a greedy manner to the most energy-efficient server to optimize power. Buyya et al. [37] focus on developing dynamic resource provisioning and provide methods for efficiently managing workloads between datacenters to optimize energy conservation and enhance the performance of data centers. Their algorithm explains the VM allocation policies that consider both QoS and power consumption.

An energy-efficient task scheduling algorithm (ETSA) was presented by Panda et al. [38] for scheduling heterogeneous workloads in cloud environment. The ETSA algorithm makes a scheduling decision after considering the task's completion time and overall resource utilization. The task with minimum normalization value is mapped to virtual machine to reduce makespan as well as power consumption. Similarly, a task scheduler based on travelling salesman problem (TSP) is proposed by Nasr et al. [29] for task scheduling to improve makespan. In the proposed algorithm, tasks are grouped as clusters, then execution time of each cluster is calculated to create a matrix similar to the TSP matrix, and at the end by using the nearest neighbor algorithm clusters are mapped to virtual machines.

The Cloud Acknowledgement Scheme (CACKS), a new method for acknowledging packets by using a single hop cloud in order to reduce energy consumption and improve network performance is proposed by Kaja et al. [39]. The proposed approach uses variable time and variable frequency based algorithms to minimize energy and increase performance. The shortest-round-vibrant-queue (SRVQ) algorithm proposed by Jeevitha et al. [40] combines the shortest-job-first algorithm and the round-robin method, while optimizing the energy consumption. All tasks are sorted in ascending order by burst time, which is subsequently used to optimize makespan. The method employs the voltage & frequency scaling approach, which aims to minimize process waiting time and enhance the efficiency of energy consumption.

In [30], the author describes First-Come-First-Serve (FCFS), Round-Robin, Shortest-Job-First , Min-Min, and Max-Min scheduling algorithms in detail. The FCFS approach maps the cloudlets to virtual machines according to their arrival time in a queue, and the allocation policy of algorithm is simply sequential. The Round-Robin algorithm iteratively allocates virtual machines to tasks with each iteration starting from first virtual machine and allocation of task is decided by time slice. In Min-Min task scheduling algorithm, small tasks are mapped first, while in Max-Min scheduling, the large tasks are executed first.

Li et al. [41] propose a greedy approach to decrease the overall completion time. In their scheduling approach, the tasks are initially arranged in descending order w.r.t their length while virtual machines are arranged in descending order according to processing power. After sorting, the tasks are iteratively mapped to virtual machines like RRA.

In terms of the meta-heuristic-based scheduling strategies, Rekha et al. [18] proposed the ETA-GA scheduling approach to perform mapping of tasks to resources based on resource capability. The task completion time and failure probability are used to compute the fitness of each chromosome. The ETA-GA approach chooses two chromosomes from the population for crossover and mutation based on fitness value. The algorithm suffers from pre-mature convergence due to selection of best chromosome in each generation. Moreover, the power consumption is not optimized, which results in VMs consuming more energy. Similarly, Zhao et al. [42] suggest a GA-based optimization method for allocating independent tasks to dynamic resources in cloud computing environments. The algorithm encodes chromosomes as digital strings, while fitness function takes into account the deadline criteria of task completion for optimizing resource usage and execution time. Soulegan et al.[43] propose another genetic algorithm based approach for task scheduling. Their approach uses fitness function as a sum of cost and makespan for optimizing the schedules. Similarly, a power-aware approach using non-dominated sorting genetic algorithm for scheduling on cloud platforms is given by Khan [11]. The approach initially optimizes the objective functions and then arranges VM indexes so that the VMs with higher weights are considered for frequent assignment to tasks.

Kaur et al. [23] combine two heuristics corresponding to assigning of the longest cloudlet and the shortest cloudlet to the fastest processor for task scheduling in cloud environment while taking into account the parameters of computational complexity and the capacity of resources. The characteristics of heuristics and randomization are also incorporated in their strategy to increase population diversity and find a better solution with a short makespan. Similarly, the Tournament Selection based Genetic Algorithm (TS-GA) is proposed by

Hamad et al. [22]. Their algorithm uses binary encoding scheme for population initialization. Its objective function considers overall finish time of tasks on available resources. When compared to round-robin and simple genetic algorithms, TS-GA performs better for small number of VMs.

Ying et al.[12] introduce an energy-aware task scheduling algorithm to improve the makespan and energy consumption using GA. The Dynamic Voltage Scaling (DVS) is employed to allow CPUs to react dynamically on various voltage supply levels to increase energy efficiency. The entire search space is explored by implementing a genetic algorithm. A modified genetic algorithm with enhanced max-min algorithm is introduced by Singh et al. [44] for scheduling independent tasks. The enhanced max-min algorithm is used to generate initial population to get optimal results for makespan. The largest task based on execution time is assigned to the VM having small amount of processing power. Similarly, in [45] [46] the authors also use genetic algorithm to reduce response time and energy consumption. In hybrid scheduling approaches, Alsaidy et al. [25] suggest heuristics for initialization of solutions for PSO. The heuristics in the hybrid approach allocate VMs to tasks using minimum execution time of a task computed among all virtual machines and a round-robin assignment mechanism. Another hybrid technique [24] combining heuristic and metaheuristic strategies also attempts to improve the exploration of search space in order to determine an optimal solution by incorporating the Shortest-Job-to-Fastest-Processor (SJFP) method along with PSO. The authors introduce SJFP heuristic in initialization phase for a better selection of overall population. Similarly, Manasrah et al. [47] propose a hybrid algorithm that divides iterations in PSO and GA algorithms. The population is initially updated using the iterations of GA algorithm. Subsequently, the population generated by the GA algorithm is used by PSO for optimizing schedules of workflows executing on cloud platforms.

In contrast to the methodologies stated above, this paper proposes power-optimized task scheduling using genetic algorithm (POTS-GA) that aims to efficiently execute user requests to generate small schedule length, while optimizing makespan as well as power consumption. The finish time of each VM and overall power consumption of tasks on virtual machines are used for computing fitness and subsequent assignment of VMs to tasks. Through consideration of both objectives, the POTS-GA strategy is able to significantly perform better than other approaches in terms of makespan and power consumption.

3. Power Optimized Task Scheduling using Genetic Algorithm (POTS-GA)

The POTS-GA scheduling approach uses genetic algorithm to search for solution space in order to get optimal scheduling. It aims at minimizing power consumption as well as reducing response time without affecting task performance.

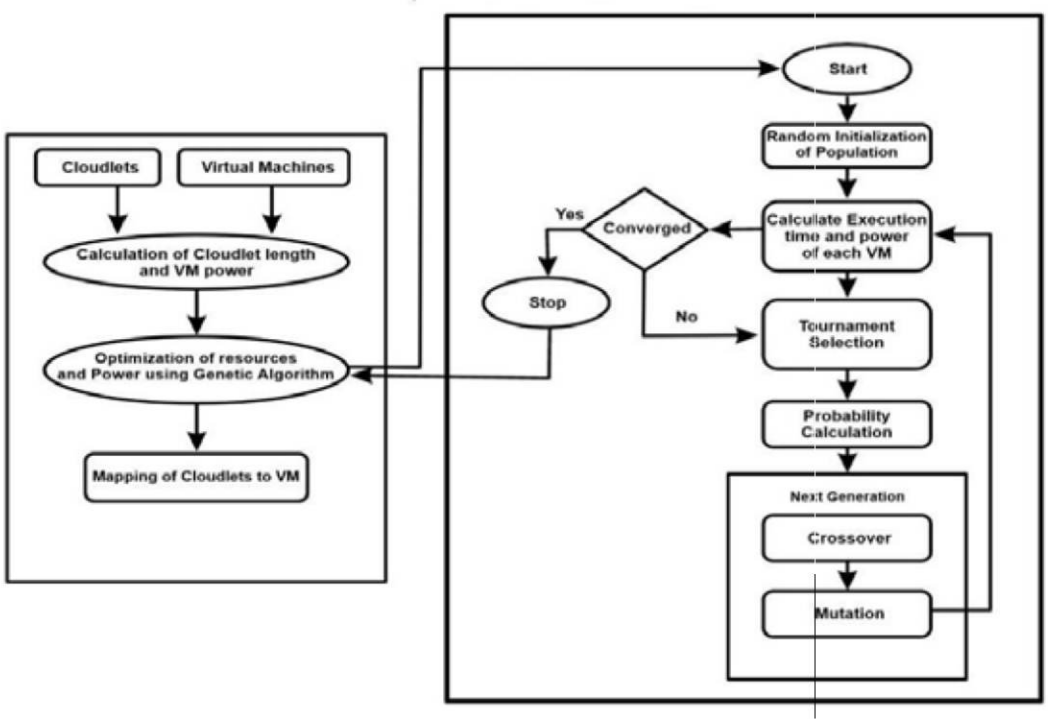

Figure 1. POTS-GA Scheduler

Figure 1 depicts the main phases of the proposed POTS-GA algorithm. The tasks of various sizes and virtual machines with diverse processing capability and power consumption are used as input by the algorithm to get best mapping of tasks to VMs. The GA starts with a random population of individual chromosomes, where each chromosome represents a feasible mapping. The fitness of each individual chromosome is assessed by using the overall execution time and the power of virtual machines. The genetic operators i.e. crossover and mutation are used to generate new viable solutions. The chromosomes (mappings) with low makespan and power are likely to be retained among next generations. The tasks are then mapped to virtual machines using the most appropriate mapping found through the chromosome having the best fitness value.

The major steps of the POTS-GA scheduling approach are detailed below:

3.1. Problem Encoding and Population Initialization

The POTS-GA algorithm uses a discrete encoding scheme for representing a chromosome as a collection of **1×n**values, where **n** represents the number of tasks. The tasks are to be mapped to **m** virtual machines. The entire population of chromosomes is initialized with chromosomes having random gene values, where each gene value represents index of the VM to be considered for allocation to a task.

Assume the tasksT_1, T_2, $T3$, ..., $T_{10}$to be mapped to virtual machines V_1,V_2, V_3, V_4, V_5, where more than one task can be mapped to a VM. A chromosome with values {3,5,3,2,5,1,3,2,3,2} would imply the taskT_1being mapped to V_3. Similarly, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10} are mapped to V_5, V_3, V_2, V_5, V_1, V_3, V_2, V_3, V_2, respectively. The POTS-GA algorithm searches for the mapping with the objective of minimizing makespan and power consumption.

3.2. Fitness Function

The fitness of a chromosome in the POTS-GA algorithm depends upon both the makespan and the power consumption. The computation of makespan requires execution time and finish time of the virtual machines, whereas, the power consumption is based on the power requirement of the machine and the execution time of tasks. For our scenario, we formulate the problem as $\boldsymbol{Min(\Theta(\psi, W))}$, as discussed below.

The execution time $\boldsymbol{E_{(T_i,V_k)}}$of a task $\boldsymbol{T_i}$ that is assigned to the VM $\boldsymbol{V_k}$is computed as:

$$\boldsymbol{E_{(T_i,V_k)}} = \frac{\boldsymbol{T_i.length}}{\boldsymbol{V_k.mips}} \qquad (1)$$

where $\boldsymbol{T_i}$,$\forall i$=0, 1, 2….n-1, represents tasks, the variable ***length*** represents the individual length or number of instructions of each task, and ***mips*** is the processing capability of each virtual machine in millions of instructions per second.

The finish time of a virtual machine $\boldsymbol{V_k}$ is the sum of execution times of tasks $T_s \subseteq T$, assigned to the virtual machine, as given below:

$$\boldsymbol{F_{V_k}} = \sum_{i=0}^{|T_s|-1} \boldsymbol{E_{(T_{si},V_k)}}, \qquad (2)$$

where $\boldsymbol{F_{V_k}}$,k=0,1,2,…m-1 represents the finish time of tasks $\boldsymbol{T_s}$ assigned to each virtual machine $\boldsymbol{V_k}$. Since the virtual machines are running concurrently, the makespan $\boldsymbol{\psi}$ is the maximum finish time among all VMs.

$$\boldsymbol{\psi} = \max\left(F_{V_k}\right), \forall k=0,1,2,\ldots..,m\text{-}1 \qquad (3)$$

The power consumption $\boldsymbol{W}$for overall execution of tasks is computed as:

$$\boldsymbol{W} = \quad \boldsymbol{\psi} * \left(\frac{pwr}{3600*1000}\right) \qquad (4)$$

where ***pwr*** represents the power consumption (in kWh) of the host on which the virtual machines are running.

The fitness of a chromosome is computed as the sum of scaled values of makespan and power consumption. Let $\boldsymbol{\psi}^{max}$&$\boldsymbol{\psi}^{min}$represent the maximum and minimum makespan values, and $\boldsymbol{W}^{max}$&$\boldsymbol{W}^{min}$ represent the maximum and minimum power consumption among all mappings (chromosomes,) as computed below:

$$\boldsymbol{\psi}^{max} = \text{n} * \left(\frac{(\boldsymbol{max}(\forall_{i=0}^{n-1} C_i.length)}{(\boldsymbol{min}(\forall_{i=0}^{m-1} V_i.mips)}\right) \qquad (5)$$

$$\boldsymbol{\psi}^{min} = \quad \left(\frac{(\boldsymbol{min}(\forall_{i=0}^{n-1} C_i.length)}{(\boldsymbol{max}(\forall_{i=0}^{m-1} V_i.mips)}\right) \qquad (6)$$

$$\boldsymbol{W}^{max} = \boldsymbol{\psi}^{max} * \left(\frac{pwr}{3600*1000}\right) \qquad (7)$$

$$\boldsymbol{W}^{min} = \boldsymbol{\psi}^{min} * \left(\frac{pwr}{3600*1000}\right) \qquad (8)$$

For a chromosome, having power **W** and makespan $\boldsymbol{\psi}$, the fitness $\boldsymbol{\Theta}$ is calculated as:

$$\boldsymbol{\Theta} = \frac{\boldsymbol{\psi}-\boldsymbol{\psi}^{min}}{\boldsymbol{\psi}^{max}-\boldsymbol{\psi}^{min}} + \frac{\boldsymbol{W}-\boldsymbol{W}^{min}}{\boldsymbol{W}^{max}-\boldsymbol{W}^{min}} \qquad (9)$$

The two objectives of minimizing the power consumption and makespan are integrated into the fitness value$\boldsymbol{\Theta}$ that is used by the POTS-GA algorithm to search for optimal mapping.

3.3. Selection Operator

The selection operator used in the proposed algorithm is tournament selection which selects k-individuals from the population. The fitness of each chromosome based on execution time and power is calculated and the selected 'k' chromosomes are sorted in order of fitness values. The best chromosomes with high fitness value are then selected for generating new chromosomes through crossover and mutation.

3.4. Crossover

The POTS-GA algorithm uses single-point &two-point crossover operators to produce offspring for a new population. For instance, two chromosomes (having indices of mapped virtual machines) with a randomly selected crossover point C=5 for one-point crossover, and the corresponding output chromosomes obtained after crossover are shown in Figures 2 and 3, respectively.

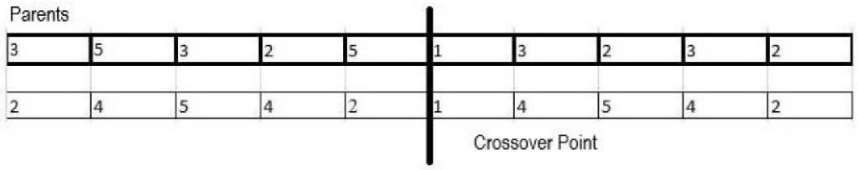

Figure 2. Chromosomes before Single Point Crossover

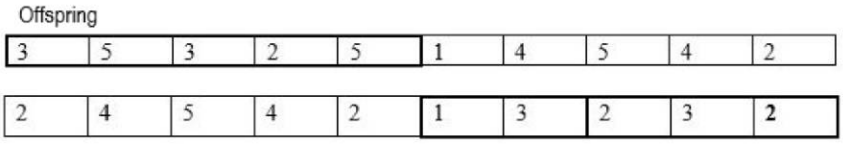

Figure 3. Chromosomes after Single-Point Crossover

Similarly, for the two-point crossover with two randomly chosen crossover points C_1=4 and C_2=8, the input chromosomes and the offspring are shown in Figures 4 and 5, respectively.

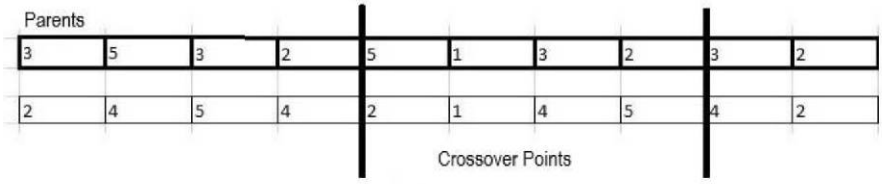

Figure 4. Chromosomes before the Two-Point Crossover

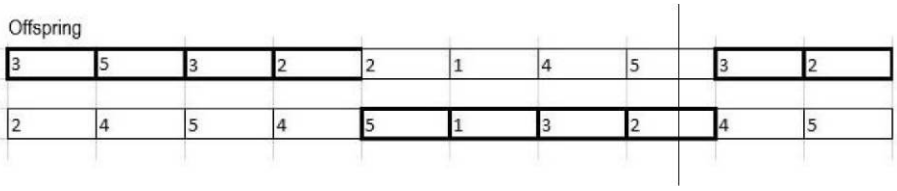

Figure 5. Chromosomes after Two-Point Crossover

3.5. Mutation

The POTS-GA algorithm performs mutation (Figure 6) to generate random VM indices to be used for mapping. The mutation rate with a probability defines the frequency of replacing VM indices with new indices.

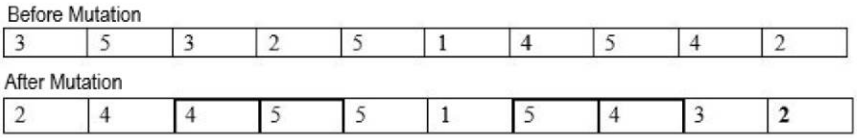

Figure 6. Mutation Operation

For mutation, a random integer ***r,*** $0 \leq r \leq 1$ is generated and the VM indices in a chromosome are replaced only if the value of **r** exists within the range of the mutation probability.

3.6. Find Best Mapping

The POTS-GA algorithm uses the fitness value to determine the best mapping. The chromosome producing the lowest fitness value is computed and compared with the previous best chromosome at each iteration for update.

3.7. Allocation of Virtual Machines

After execution of all iterations, the chromosome having the best fitness value is used for assigning tasks to VMs. The individual genes representing VMs are allocated to the corresponding tasks.

Algorithm 1: Power optimized task scheduling using genetic algorithm (POTS-GA)

1: /* Let V_j, **∀j** =0,1,2,...,**m-1**represent the set of virtual machines, each having different parameters and let T_i,**∀i=0,1,2,...,n-1**be the set of tasks, each being characterized with different features. The variable η is used to represent the tournament size. Let P_i, **∀i**=0,1,2,..., |***P***|-1 represent the population of chromosomes */

2: **Begin**

3: x= 1, ***Max***= 1000

4: // Main loop to search for best mapping

5: **while** (**x**< =***Max***) **do**

6: //Initialize the population P chromosomes with random //VM indices $\mathbf{R}_j$

$\boldsymbol{P_i}=\{\boldsymbol{R_j}, \forall j=0,1,2,...,n-1\}, \ \forall i=0,1,2,..., |\boldsymbol{P}|-1$

7: **if (x == 1) then**

8: **B**=0 //Initialize index for the global best //chromosome

9: **end if**

10: **Compute** fitness $\boldsymbol{\Theta_i}$ of each chromosome $\boldsymbol{P_i}$, $\forall i=0,1,2,...,|\boldsymbol{P}|-1$ by using Equations (1-9) of **S**ection 3.2

11: $\boldsymbol{Q}=\{\}$, $\boldsymbol{R}=\{\}$,**S**$=\{\}$ // Initialize Q and R *and* S as //temporary population variables

12: //Select $\boldsymbol{\eta}$ chromosomes using tournament selection given in **S**ection 3.3.

13: $\boldsymbol{R}$= **tournamentSelection** ($\boldsymbol{P}$)

14: //Let S be the remaining individuals to be passed to //next generation as elites

15: **if** $\boldsymbol{\eta}$%2!=0 **then**

16: **Add** chromosome $\boldsymbol{R}_{\eta-1}$ to the population $\boldsymbol{Q}$

17: **end if**

18: //Apply crossover as mentioned in **S**ection 3.4.

19: **for** j= 0 to $\boldsymbol{\eta}$ -1 **step** 2 **do**

20: (u,v) = **crossover** ($\boldsymbol{P}$, j, j+1) // return pair of //chromosomes u and v

21: **Add** chromosomes u and v to population Q

22: **end for**

23: //Apply mutation operator as described in **S**ection 3.5

24: $\boldsymbol{P}$ = **mutation** (Q)

25: **Add** $\boldsymbol{S}$ to population $\boldsymbol{P}$

26: $\boldsymbol{x}=\boldsymbol{x}$+1

27: //Find index of the chromosome having the best fitness //values as given in Section 3.6

28: $\boldsymbol{k}$ = ***findBestMapping()***

29: **if** $\boldsymbol{\Theta_B}<\boldsymbol{\Theta_k}$ **then**

30: **B** = **k** //Update the global best chromosome
31: **endif**
32: **end while**
33: //Allocate the tasks to VMs
34: **for**j=0,1,2,...,n-1 **do**
35: Map task $\mathbf{T_j}$ to the VM $P_B[j]$
36: **end for**
37: **End**

The POTS-GA algorithm (Algorithm 1) performs initialization of the variables for iteration count and maximum iteration at step 3. At step 5, the loop is iterated for the given iteration count to search for optimal mapping. The step 6 initializes the population with random virtual machine and index of the global best chromosome **B** is initialized at steps 7-9. The fitness of each individual chromosome based on makespan and power consumption is computed at step 10. The step 11 initializes temporary population variables. The η chromosomes are selected through tournament selection at step 13. The steps 15-22 add chromosomes to the population **Q** in the form of pairs. The remaining chromosome is added using steps 15-17, while the steps 19-22 perform crossover to add new chromosomes as pairs to the population **Q**. The mutation operator is performed on the population **Q** to produce the new population **P** at step 24. The remaining chromosomes **S** are added to the population **P**, and the loop count is incremented subsequently. The steps 28-31 find the best mapping and update the global best chromosome **B**. The allocation of virtual machines to task is performed through final steps by using the elements of the best chromosome $\mathbf{P_B}$. The complexity of the POTS-GA algorithm is $O(|P|*Max*n)$, where $|P|$, Max, and n represent respectively the population size, the number of iterations and the number of tasks. Since POTS-GA uses a meta-heuristic approach, its cost for large-scale environments may be reduced by limiting the population size and the number of iterations.

4. Experimentation and Results

4.1 Simulation Setup

For experimentation, the CloudSim framework [48] has been used. It includes PowerDatacenter, PowerHost and PowerVm for power-based configurations for Datacenters, Hosts and VMs, respectively. Each task contains a user request that is comprised of a number of instructions. The algorithms have been evaluated with number of tasks set to vary from 50

to 500 and the number of VMs set to vary from 4 to 16. We have performed simulation using configuration parameters given in Table 3.

Table 3. Platform and configuration parameters

Simulation Platform	CloudSim 4.0	**Number of Host Machines**	4
Virtual Machine Monitor (VMM)	Xen	**Task Length (Millions of instructions)**	1000-2000
Task Scheduler	Space Shared	**VM processing elements**	1
System Architecture	X86	**Number of Tasks**	50,100,…,500
Processing elements of Tasks	1	**Number of VMs**	4,8,16
Number of Datacenters	2	**VM RAM**	512MB
VM Bandwidth	10 Megabits/s	**Power (Watts)**	200-300

Table 4. Parameters used for the POTS-GA algorithm

Encoding	Discrete	**Crossover probability**	0.8
Population Size	100	**Mutation operator**	Random resetting
Max iterations	1000	**Mutation probability**	0.05
Crossover operator	30% Single point and 70% two- point	**Tournament size**	80% of population

The parameters used for the POTS-GA algorithm are given in Table 4. For performance evaluation, the POTS-GA algorithm is compared with other algorithms including the ETA-GA , Robin algorithm (RRA), FCFS, and Greedy algorithm. The algorithms are evaluated in

terms of makespan and power consumption. The makespan or schedule length is the total amount of time required for execution of the tasks mapped to virtual machines. Similarly, the power consumption (kWh) represents the power consumed by the virtual machines for executing all the scheduled tasks.

4.2 Evaluation Outcomes

This section provides an overview of the performance results of power and makespan for the 04, 08, and 16 virtual machines, respectively.

4.2.1 Makespan

Figure 7 depicts the makespan of various methods utilizing four VMs with diverse numbers of tasks, ranging from 50 to 500.The performance of the POTS-GA algorithm and the ETA-GA algorithm is nearly identical when the number of tasks is small. It is evident from the figure that with a large number of tasks, the performance of ETA-GA degrades beyond the batch size of 300. Overall, POTS-GA surpasses all other strategies in terms of makespan. The ETA-GA algorithm performs next to the POTS-GA algorithm by producing small makespan values.

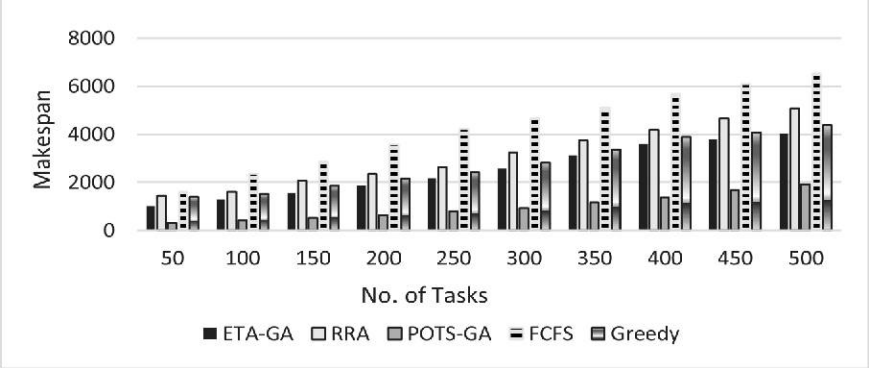

Figure 7. Makespan for different number of tasks using 04 virtual machines

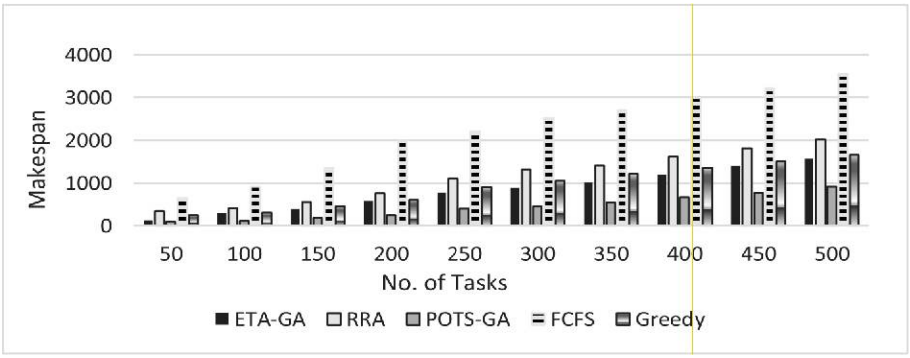

Figure 8. Makespan for different number of tasks using 08 VMs

With eight VMs, the makespan for the evaluated algorithms with various number of tasks is shown in Figure 8. The proposed technique POTS-GA has drastically produced small makespan when compared to all other benchmark scheduling techniques. When there are a few tasks, the RRA algorithm performs better than the Greedy algorithm. As the number of tasks is increased, the RRA performs the worst by producing large makespan values. Followed by the performance of the POTS-GA algorithm, the ETA-GA algorithm performs significantly better than RRA, Greedy and FCFS for all batch sizes.

The schedule length for various scheduling algorithms, with different number of tasks being executed on 16 VMs is depicted in Figure 9. Similar to previous scenarios, when virtual machines are small in number, the POTS-GA algorithm continues to outperform other task scheduling techniques in this case when the number of VMs is large.

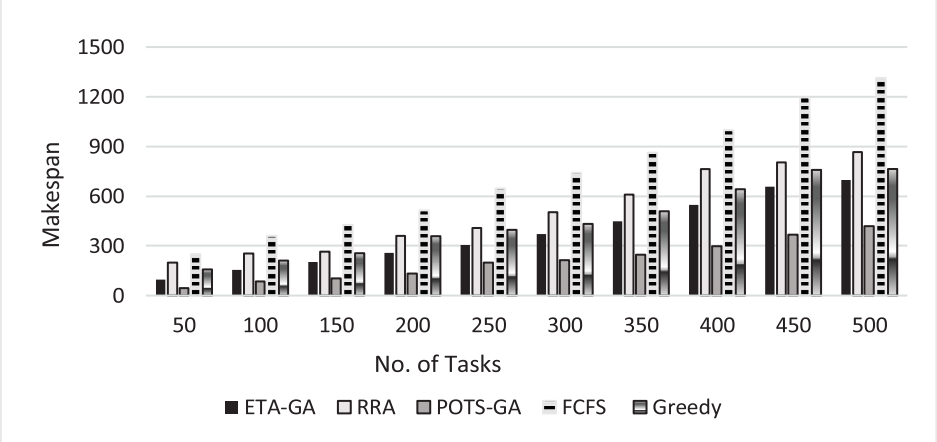

Figure 9. Makespan for different number of tasks using 16 virtual machines

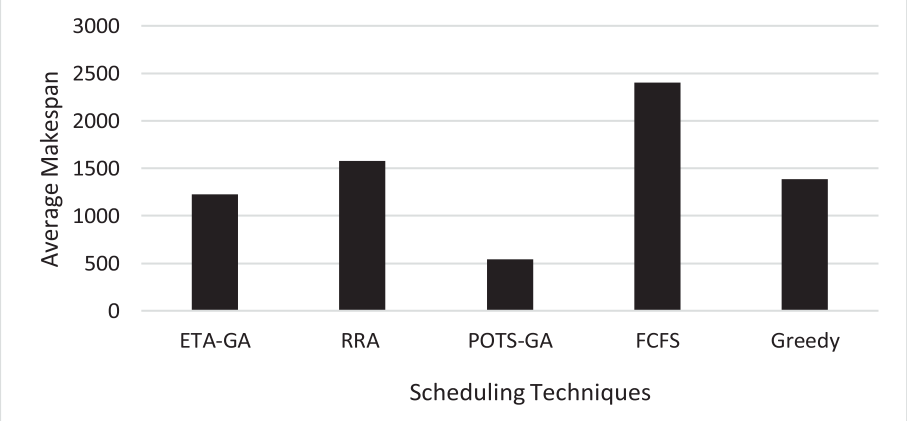

Figure 10. Average Makespan obtained for all configurations of VMs

The overall average makespan of various scheduling techniques is shown in Figure 10. The results depict that the proposed POTS-GA algorithm provides better execution time in almost all batch sizes of tasks. Overall, the POTS-GA algorithm achieves65%, 75%, 80% and 73% average improvement over the ETA-GA, RRA, FCFS and Greedy algorithms, respectively, in terms of schedule length.

4.2.2 Power Consumption

The power consumed for execution of tasks using the schedule generated by the various scheduling strategies, with different number of tasks and VMs is given in Table 5.

Table 5. Power Consumption for different number of tasks using 04 virtual machines

	Tasks									
	50	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
ETA-GA	42.82	67.46	143.60	182.60	210.30	253.3	284.20	332.20	367.10	407.50
RRA	52.39	85.52	133.18	177.94	204.06	220.37	269.66	323.02	352.33	379.19
POTS-GA	2.97	8.42	13.92	30.51	39.94	46.52	70.03	87.12	119.86	142.26
FCFS	91.23	157.8	215.45	278.18	327.64	367.52	398.78	426.34	473.36	512.93
Greedy	62.9	92.8	110.45	158.3	181.6	210.3	267.5	305.5	335.2	375.2

The power consumption incurred for schedules produced by ETA-GA and the proposed strategy POTS-GA is similar to some extent when number of tasks is small. But the power consumption for execution of tasks scheduled through ETA-GA is exponentially increased for large number of tasks. Table 5clearly depicts that the power consumption incurred for execution of tasks scheduled through POTS-GA is significantly less than RRA, FCFS, Greedy and ETA-GA.

Table 6. Power consumption for different number of tasks using 08 virtual machines

	Tasks									
	50	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
ETA-GA	21.86	39.98	60.45	86.29	114.46	148.54	191.89	251.97	298.25	367.72
RRA	49.88	95.32	172.50	201.86	256.00	286.44	310.73	347.12	415.25	456.68
POTS-GA	7.62	11.92	28.07	35.94	41.52	66.03	85.12	121.86	146.26	167.98

FCFS	100.82	165.24	215.2	295.72	385.87	476.21	543.06	597.94	655.14	715.47
Greedy	76.24	92.85	167.28	197.36	256.65	298.57	343.42	368.24	386.92	395.56

The power consumption of different algorithms having various number of tasks with eight VMs is shown in Table 6. Overall, the POTS-GA outperforms all other scheduling strategies including the ETA-GA algorithm. For large number of tasks, the Greedy algorithm outperforms RRA in terms of power consumption. The FCFS algorithm results in the highest power consumption for all configurations with varying number of tasks.

Table 7: Power Consumption incurred for execution of tasks on 16 VMs

	Tasks									
	50	**100**	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
ETA-GA	27.63	85.82	165.45	196.56	196.46	232.54	254.89	294.97	323.92	362.72
RRA	42.87	94.54	195.92	257.86	277.00	358.74	408.07	449.12	487.76	510.87
POTS-GA	7.41	20.92	35.07	54.94	71.52	88.03	125.12	136.86	162.26	185.87
FCFS	98.68	163.54	272.32	345.67	412.79	495.44	578.07	654.28	714.83	847.65
Greedy	52.85	103.78	207.26	244.65	256.57	297.94	315.24	388.43	405.56	432.48

Table 7 illustrates the power consumption of POTS-GA and other methods while using 16 virtual machines with varying number of tasks. According to Table 7, POTS-GA consumes less power than FCFS, RRA, Greedy, and ETA-GA. The FCFS algorithm results in the highest values of power consumption for all scenarios. The ETA-GA algorithm consumes less energy for 50 to 250 tasks. However, as the number of tasks reaches up to 300, the performance of the ETA-GA algorithm starts to deteriorate. Similar to the results for 08 VMs, the RRA algorithm generates schedules that incur less power consumption in comparison with the FCFS algorithm.

Figure 11 shows the overall power usage of several scheduling strategies with various virtual machines. As shown in the diagram that the POTS-GA consumes less power than other approaches for all batch sizes of tasks using different number of VMs. The average improvement in power consumption achieved by POTS-GA over the ETA-GA, RRA, FCFS and Greedy algorithms is 66%, 76%, 82%, and 74%, respectively.

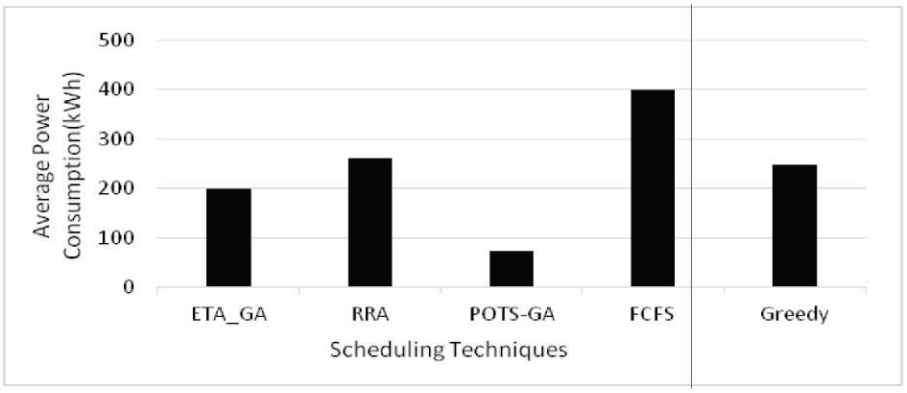

Figure 11. Average power consumption for all configurations of virtual machines

5. Conclusion

For cloud computing environments, task scheduling is considered to be a major issue due to its complexity. On high performance cloud platforms, an effective task scheduling can improve overall execution time as well as power consumption. In this paper, we have proposed the POTS-GA algorithm that aims to optimize execution time and power consumption by generating small length schedules while simultaneously considering the power of each virtual machine. The genetic algorithm uses power of virtual machine and makespan for computing fitness of each chromosome. After a number of iterations, the chromosome with best mapping is found and the tasks are then assigned to VMs. We have conducted experiments using various number of tasks and VMs to evaluate the performance of POTS-GA and other algorithms. The performance results show that the POTS-GA algorithm performs 65%, 75%, 80% and 73%better for makespan and 66%, 76%, 82%, and 74%better for power consumption than the ETA-GA, RRA, FCFS, and Greedy algorithms, respectively.

For our current implementation, the execution cost & resource utilization are not considered in our proposed approach, so we aim to extend our work by considering these optimization metrics in future. Moreover, a heuristic for initializing the population may be incorporated to enhance performance of the proposed algorithm.

Acknowledgment

This research work was accomplished as part of the research work conducted during MS Thesis at BZU Multan, Pakistan. The authors are thankful to the Department of Computer Science, BZU, Multan, for provision of equipment required to carry out this research work.

References

1 Zhang, S., Zhang, S., Chen, X., & Huo, X. (2010). Cloud computing research and development trend. *2nd International Conference on Future Networks, ICFN 2010*, 93–97. https://doi.org/10.1109/ICFN.2010.58

2 Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. *Communications of the ACM*, *53*(4), 50–58. https://doi.org/10.1145/1721654.1721672

3 Odun-Ayo, I., Ananya, M., Agono, F., & Goddy-Worlu, R. (2018). Cloud Computing Architecture: A Critical Analysis. *Proceedings of the 2018 18th International Conference on Computational Science and Its Applications, ICCSA 2018*, 1–7. https://doi.org/10.1109/ICCSA.2018.8439638

4 Birke, R., Chen, L. Y., & Smirni, E. (2012). Data centers in the cloud: A large scale performance study. *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, 336–343. https://doi.org/10.1109/CLOUD.2012.87

5 Yuan, H., Kuo, C. C. J., & Ahmad, I. (2010). Energy efficiency in data centers and cloud-based multimedia services: an overview and future directions. *2010 International Conference on Green Computing, Green Comp 2010*, 375–382. https://doi.org/10.1109/GREENCOMP.2010.5598292

6 Abdessamia, F., Tai, Y., Zhang, W. Z., & Shafiq, M. (2017). An improved particle swarm optimization for energy-efficiency virtual machine placement. *Proceedings - 5th International Conference on Cloud Computing Research and Innovation, ICCCRI 2017*, 7–13. https://doi.org/10.1109/ICCCRI.2017.9

7 Bobroff, N., Kochut, A., & Beaty, K. (2007). Dynamic placement of virtual machines for managing SLA violations. *10th IFIP/IEEE International Symposium on Integrated Network Management 2007, IM '07*, *5*, 119–128. https://doi.org/10.1109/INM.2007.374776

8 Malekloo, M. H., Kara, N., & El Barachi, M. (2018). An energy efficient and SLA compliant approach for resource allocation and consolidation in cloud computing environments. *Sustainable Computing: Informatics and Systems*, *17*, 9–24.

https://doi.org/10.1016/j.suscom.2018.02.001

9 Pietri, I., & Sakellariou, R. (2016). Mapping virtual machines onto physical machines in cloud computing: A survey. *ACM Computing Surveys*, *49*(3). https://doi.org/10.1145/2983575

10 Al-Maytami, B. A., Fan, P., Hussain, A., Baker, T., & Liatsist, P. (2019). A Task Scheduling Algorithm with Improved Makespan Based on Prediction of Tasks Computation Time algorithm for Cloud Computing. *IEEE Access*, *7*, 160916–160926. https://doi.org/10.1109/ACCESS.2019.2948704

11 Khan, M. A. (2022). A cost-effective power-aware approach for scheduling cloudlets in cloud computing environments. *Journal of Supercomputing*, *78*(1), 471–496. https://doi.org/10.1007/s11227-021-03894-2

12 Ying, C. T., & Yu, J. (2012). Energy-aware genetic algorithms for task scheduling in cloud computing. *Proceedings - 7th ChinaGrid Annual Conference, ChinaGrid 2012*, 43–48. https://doi.org/10.1109/ChinaGrid.2012.15

13 Ahmad, R. W., Gani, A., Hamid, S. H. A., Shiraz, M., Yousafzai, A., & Xia, F. (2015). A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, *52*, 11–25. https://doi.org/10.1016/j.jnca.2015.02.002

14 Duy, T. V. T., Sato, Y., & Inoguchi, Y. (2010). Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. *Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum, IPDPSW 2010*, 1–8. https://doi.org/10.1109/IPDPSW.2010.5470908

15 Jafarnejad Ghomi, E., Masoud Rahmani, A., & Nasih Qader, N. (2017). Load-balancing algorithms in cloud computing: A survey. *Journal of Network and Computer Applications*, *88*(December 2016), 50–71. https://doi.org/10.1016/j.jnca.2017.04.007

16 Deepa, T., & Cheelu, D. (2018). A comparative study of static and dynamic load balancing algorithms in cloud computing. *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing, ICECDS 2017*, 3375–3378. https://doi.org/10.1109/ICECDS.2017.8390086

17 Manglani, V., Jain, A., & Prasad, V. (2018). Task Scheduling in Cloud Computing

Environment. *International Journal of Computer Sciences and Engineering*, *6*(5), 513–515. https://doi.org/10.26438/ijcse/v6i5.513515

18 Rekha, P. M., & Dakshayini, M. (2019). Efficient task allocation approach using genetic algorithm for cloud environment. *Cluster Computing*, *22*(4), 1241–1251. https://doi.org/10.1007/s10586-019-02909-1

19 Wang, Y., Zuo, X., Wu, Z., Wang, H., & Zhao, X. (2022). Variable neighborhood search based multiobjective ACO-list scheduling for cloud workflows. *Journal of Supercomputing*, *78*(17), 18856–18886. https://doi.org/10.1007/s11227-022-04616-y

20 Nabi, S., Ahmad, M., Ibrahim, M., & Hamam, H. (2022). AdPSO: Adaptive PSO-Based Task Scheduling Approach for Cloud Computing. *Sensors*, *22*(3), 1–22. https://doi.org/10.3390/s22030920

21 Navimipour, N. J. (2015). Task scheduling in the Cloud Environments based on an Artificial Bee Colony Algorithm. *Proceedings of 2015 International Conference on Image Processing, Production and Computer Science (ICIPCS'2015)*, 38–44.

22 Hamad, S. A., & Omara, F. A. (2016). Genetic-Based Task Scheduling Algorithm in Cloud Computing Environment. *Computer Science and Application*, *06*(06), 317–322. https://doi.org/10.12677/csa.2016.66038

23 Kaur, S., & Verma, A. (2012). An Efficient Approach to Genetic Algorithm for Task Scheduling in Cloud Computing Environment. *International Journal of Information Technology and Computer Science*, *4*(10), 74–79. https://doi.org/10.5815/ijitcs.2012.10.09

24 Abdi, S., Motamedi, S. A., & Sharifian, S. (2014). Task Scheduling using Modified PSO Algorithm in Cloud Computing Environment. *Proceedings - International Conference on Machine Learning, Electrical and Mechanical Engineering, ICMLEME 2014*, 37–41. https://dx.doi.org/10.15242/IIE.E0114078

25 Al-Saidy, S. A., Abbood, A. D., & Sahib, M. A. (2022). Heuristic initialization of PSO task scheduling algorithm in cloud computing. *Journal of King Saud University - Computer and Information Sciences*, 34(6), 2370–2382. https://doi.org/10.1016/j.jksuci.2020.11.002

26 Topcuoglu, H., Hariri, S., & Society, I. C. (2002). *Performance-Effective and Low-*

Complexity. *13*(3), 260–274.

27 Lin, C., & Lu, S. (2011). Scheduling scientific workflows elastically for cloud computing. *Proceedings - 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011*, 746–747. https://doi.org/10.1109/CLOUD.2011.110

28 Etminani, K., & Naghibzadeh, M. (2007). A min-min max-min selective algorihtm for grid task scheduling. *2007 3rd IEEE/IFIP International Conference in Central Asia on Internet, ICI 2007*. https://doi.org/10.1109/canet.2007.4401694

29 Nasr, A. A., El-Bahnasawy, N. A., Attiya, G., & El-Sayed, A. (2019). Using the TSP Solution Strategy for Cloudlet Scheduling in Cloud Computing. *Journal of Network and Systems Management*, *27*(2), 366–387. https://doi.org/10.1007/s10922-018-9469-9

30 Salot, P. (2016). A survey of scheduling algorithm in cloud computing environment. *International Journal of Control Theory and Applications*, *9*(36), 137–145.

31 Zhou, Z., Li, F., Zhu, H., Xie, H., Abawajay, J.H., & Chowdhury, M.U.(2020). An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments. *Neural Computing& Application***32**, 1531–1541. https://doi.org/10.1007/s00521-019-04119-7

32 Ramezani, F., Lu, J., Hussain, F. (2013). Task Scheduling Optimization in Cloud Computing Applying Multi-Objective Particle Swarm Optimization. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds) *Service-Oriented Computing. ICSOC 2013. Lecture Notes in Computer Science*, vol 8274. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-45005-1_17-

33 Gupta, S., Iyer, S., Agarwal, G., Manoharan, P., Algarni, A. D., Aldehim, G., & Raahemifar, K. (2022). Efficient Prioritization and Processor Selection Schemes for HEFT Algorithm: A Makespan Optimizer for Task Scheduling in Cloud Environment. *Electronics (Switzerland)*, *11*(16). https://doi.org/10.3390/electronics11162557

34 0020Rouzaud-cornabas, J. (2011). *A Trust Aware Distributed and Collaborative Scheduler for Virtual Machines in Cloud*.

35 Elzeki, M.O., Z. Reshad, M., & A. Elsoud, M. (2012). Improved Max-Min Algorithm in Cloud Computing. *International Journal of Computer Applications*, *50*(12), 22–27. https://doi.org/10.5120/7823-1009

36 Liu, N., Dong, Z., & Rojas-Cessa, R. (2012). Task and server assignment for reduction of energy consumption in datacenters. *Proceedings - IEEE 11th International Symposium on Network Computing and Applications, NCA 2012*, 171–174. https://doi.org/10.1109/NCA.2012.42

37 Buyya, R., Beloglazov, A., & Abawajy, J. (2010). *Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges*. *May 2016*. http://arxiv.org/abs/1006.0308

38 Panda, S. K., & Jana, P. K. (2019). An energy-efficient task scheduling algorithm for heterogeneous cloud computing systems. *Cluster Computing*, *22*(2), 509–527. https://doi.org/10.1007/s10586-018-2858-8

39 Kaja, S., Shakshuki, E. M., Guntuka, S., Ul, A., Yasar, H., & Malik, H. (2020). Acknowledgment scheme using cloud for node networks with energy - aware hybrid scheduling strategy. *Journal of Ambient Intelligence and Humanized Computing*, *11*(10), 3947–3962. https://doi.org/10.1007/s12652-019-01629-z

40 Jeevitha, J. K., & Athisha, G. (2021). A novel scheduling approach to improve the energy efficiency in cloud computing data centers. *Journal of Ambient Intelligence and Humanized Computing*, *12*(6), 6639–6649. https://doi.org/10.1007/s12652-020-02283-6

41 Li, J., Feng, L., & Fang, S. (2014). An Greedy-Based Job Scheduling Algorithm in Cloud Computing. *Journal of Software*, *9*(4), 921–925. https://doi.org/10.4304/jsw.9.4.921-925

42 Zhao, C., Zhang, S., Liu, Q., Xie, J., & Hu, J. (2009). Independent task scheduling based on genetic algorithm in cloud computing. *5th International Conference on Wireless Communications, Networking and Mobile Computing, Beijing, China, 2009, pp. 1-4,* doi: 10.1109/WICOM.2009.5301850.

43 Soulegan, N. S., Barekatain, B., & Neysiani, B. S. (2021). MTC: Minimizing Time and Cost of Cloud Task Scheduling based on Customers and Providers Needs using Genetic Algorithm. *International Journal of Intelligent Systems and Applications*, *13*(2), 38–51. https://doi.org/10.5815/ijisa.2021.02.03

44 Singh, S., & Kalra, M. (2014). Scheduling of independent tasks in cloud computing

using modified genetic algorithm. *Proceedings - 2014 6th International Conference on Computational Intelligence and Communication Networks, CICN 2014*, 565–569. https://doi.org/10.1109/CICN.2014.128

45 Gabaldon, E., Lerida, J. L., Guirado, F., & Planes, J. (2017). Blacklist muti-objective genetic algorithm for energy saving in heterogeneous environments. *Journal of Supercomputing*, *73*(1), 354–369. https://doi.org/10.1007/s11227-016-1866-9

46 Kar, I., Parida, R. N. R., & Das, H. (2016). Energy aware scheduling using genetic algorithm in cloud data centers. *International Conference on Electrical, Electronics, and Optimization Techniques, ICEEOT 2016*, 3545–3550. https://doi.org/10.1109/ICEEOT.2016.7755364

47 Manasrah, A. M. & Ali, H. B. (2018). Workflow Scheduling Using Hybrid GA-PSO Algorithm in Cloud Computing. *Wireless Communications and Mobile Computing*, *2018*, 1530-8669. https://doi.org/10.1155/2018/1934784

48 Calheiros, R., Ranjan, R., Beloglazov, A., De Rose, C. A. F., & Buyya, R. (2009). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software - Practice and Experience*, *39*(7), 701–736. https://doi.org/10.1002/spe