QoS-Aware Energy-Efficient Task Scheduling on HPC Cloud Infrastructures Using Swarm-Intelligence Meta-Heuristics

Amit Chhabra[1, *], Gurvinder Singh[2] and Karanjeet Singh Kahlon[2]

Abstract: Cloud computing infrastructure has been evolving as a cost-effective platform for providing computational resources in the form of high-performance computing as a service (HPCaaS) to users for executing HPC applications. However, the broader use of the Cloud services, the rapid increase in the size, and the capacity of Cloud data centers bring a remarkable rise in energy consumption leading to a significant rise in the system provider expenses and carbon emissions in the environment. Besides this, users have become more demanding in terms of Quality-of-service (QoS) expectations in terms of execution time, budget cost, utilization, and makespan. This situation calls for the design of task scheduling policy, which ensures efficient task sequencing and allocation of computing resources to tasks to meet the trade-off between QoS promises and service provider requirements. Moreover, the task scheduling in the Cloud is a prevalent NP-Hard problem. Motivated by these concerns, this paper introduces and implements a QoS-aware Energy-Efficient Scheduling policy called as CSPSO, for scheduling tasks in Cloud systems to reduce the energy consumption of cloud resources and minimize the makespan of workload. The proposed multi-objective CSPSO policy hybridizes the search qualities of two robust metaheuristics viz. cuckoo search (CS) and particle swarm optimization (PSO) to overcome the slow convergence and lack of diversity of standard CS algorithm. A fitness-aware resource allocation (FARA) heuristic was developed and used by the proposed policy to allocate resources to tasks efficiently. A velocity update mechanism for cuckoo individuals is designed and incorporated in the proposed CSPSO policy. Further, the proposed scheduling policy has been implemented in the CloudSim simulator and tested with real supercomputing workload traces. The comparative analysis validated that the proposed scheduling policy can produce efficient schedules with better performance over other well-known heuristics and meta-heuristics scheduling policies.

Keywords: HPC-as-a-Service, task scheduling, quality-of-service, meta-heuristics and energy-efficiency.

[1] Department of Computer Engineering & Technology, Guru Nanak Dev University, Amritsar, 143005, India.

[2] Department of Computer Science, Guru Nanak Dev University, Amritsar, 143005, India.

[*] Corresponding Author: Amit Chhabra. Email: amit.cse@gndu.ac.in.

Received: 08 April 2020; Accepted: 27 April 2020.

1 Introduction and motivation

Cloud computing is emerging as a cost-effective, pay-as-you-go and anytime-anywhere service paradigm that offers remote computational resources in the form of utility services. Cloud computing systems allow the provisioning of a large number of resources to applications by sharing and aggregation of computing resources (virtual machines (VMs)) from single and multiple data centers [Moghaddam, Buyya and Kotagiri (2019)]. Due to low capital investment, availability of a large number of resources and pay-as-you-go model characteristics, the users have started the migration of their HPC applications from traditional Cluster and Grid computing systems to HPC clouds [Netto, Calheiros, Rodrigues et al. (2018)]. The Cloud computing service providers (CSP) offer different types of computational resources to users in the form of HPCaaS to execute HPC applications [Amazon EC2 instances (2020)].

Day by day, the QoS demands of cloud users regarding execution time, makespan, execution budget, utilization, and availability are increasing. These issues bring new challenges for the job schedules to deal with job scheduling and resource allocation issues for satisfying QoS demands. Moreover, with the continuous increase in capacity, size, and use of cloud computing systems, the associated energy consumption has also increased tremendously [Ayanoglu (2019)]. The energy consumption associated with data centers is mainly due to two reasons; the energy required for the operation of hardware components and energy used by cooling facilities to regulate heat dissipation in data centers. The energy demand by global data centers is continuously increasing and currently, it is reported to be between 1-3% of total demand for electricity in the world [Ilager, Ramamohanarao and Buyya (2019]. The increased energy consumption induces escalation in the electricity bills of system providers, reduces hardware life and releases more carbon (CO_2) emissions in the environment. This situation calls for the design of efficient task schedulers in large-scale computing infrastructures that could provide a trade-off between QoS parameters and conflicting energy consumption objectives simultaneously.

The primary goal of task scheduling policies in the Cloud is to provide the mapping of jobs to appropriate cloud resources to obtain maximal performance. However cloud task scheduling is a known NP-Hard problem and complexity of the problem rises further due to large problem size, dynamicity and heterogeneity of computing resources. The scheduling policy in the cloud can be static or dynamic. In static scheduling, all the details regarding task characteristics and virtual machines configurations are known to the scheduler before the start of scheduling, whereas in the case of dynamic scheduling, cloudlets arrive at unknown times and scheduling decisions are taken at run-time [Hemasian-Etefagh and Safi-Esfahani (2019)]. In this paper, our focus is on the static scheduling of cloudlets.

Motivated by the highlighted issues in this section, we have proposed the CSPSO policy to deal with the task scheduling problem in cloud computing systems. The many-fold contributions of the current research paper are summarized as follows:

1) An exhaustive literature survey related to the undertaken task scheduling problem in the large-scale Cloud computing infrastructure is carried out. An objective function for the minimization of makespan and energy consumption is designed.

2) A fitness-aware resource allocation heuristic is developed to allocate virtual machines

(and their cores) to tasks by considering both makespan and energy consumption. This resource allocation heuristic is used by the proposed CSPSO algorithm to convert the real-value population into the discrete population and to allocate resources to tasks efficiently.

3) A scheduling algorithm (CSPSO) by combining the search mechanisms of CS and PSO algorithms is suggested to provide an optimal solution to task scheduling problem in the Cloud. The suggested CSPSO policy possesses diversity in scheduling solutions over successive generations, avoidance of trapping in local optima and information exchange mechanism between CS and PSO solutions. Further, a cuckoo velocity update mechanism is designed to incorporate velocity features in the cuckoo search phase in CSPSO algorithm. The suggested CSPSO policy has the potential to optimize both makespan and energy consumption objectives.

4) Lastly, the experimental results of the proposed policies are compared with classical heuristics and meta-heuristics by using workloads obtained from real-supercomputing sites.

The remaining part of the paper is structured as follows. Section 2 presents extensive literature works in context with the underlying scheduling problem. Section 3 describes the system model, problem definition and proposed fitness function. Solution encoding and proposed hybrid meta-heuristics algorithm are explained in Section 4. Section 5 presents the simulation results and analysis of results. Finally, the paper is summed up with the observed conclusions in Section 6.

2 Related works

Nature-inspired meta-heuristic approaches such as evolutionary and swarm intelligence algorithms have been successfully applied over NP-hard task scheduling problem in distributed computing systems such as Grid and Cloud environments to obtain optimal or approximate solutions [Madni, Latiff, Coulibaly et al. (2017); Yu, Li, Yang et al. (2018); Mansouri, Mohammad Hasani Zade and Javidi (2019); Motlagh, Movaghar and Rahmani (2020)].

Genetic Algorithm (GA) and its variants have been extensively applied to propose novel cloud task scheduling techniques to improve various QoS parameters [Wang, Liu, Chen et al. (2014); Fang, Xia and Ge (2019)]. Shojafar et al. [Shojafar, Javanmardi, Abolfazli et al. (2015)] have combined Fuzzy logic with a Genetic algorithm to propose FUGE method for scheduling tasks over the Cloud for reducing makespan, execution budget and degree of imbalance. Vila et al. [Vila, Guirado, Lerida et al. (2019)] have suggested a BLEMO (blacklist evolutive multi-objective) approach based on GA method, to schedule batch of parallel tasks composed of independent tasks over IaaS Cloud resources to minimize the makespan and energy consumption. Shojafar et al. [Shojafar, Kardgar, Hosseinabadi et al. (2015)] have proposed a two-phase GA-based task scheduler to reduce energy and makespan on cloud computing systems.

The artificial bee colony (ABC) algorithm has been applied in Rastkhadiv et al. [Rastkhadiv and Zamanifar (2016)], with the objectives to minimize makespan and the degree of imbalance. Jena [Jena (2017)] introduced multi-objective ABC approaches for the Cloud task scheduling to optimize energy consumption, task completion time, execution cost, and utilization. A few Ant colony optimization (ACO) based algorithms for task scheduling are also found in the literature [Guo (2017); Li and Wu (2019)].

PSO scheduling algorithm has been applied in various research studies to efficiently allocate the cloud resources to tasks to improve QoS parameters [Zhang and Zuo (2013); Zuo, Zhang and Tan (2014); Zhao (2014)]. An APSO-VI based task scheduling algorithm was introduced in the cloud environment by considering deadline as a constraint [Kumar and Sharma (2018)].

In Navimipour et al. [Navimipour and Milani (2015)], the authors suggested cuckoo search based policy to efficiently schedule tasks and improve QoS parameters. Madni et al. [Madni, Latiff, Ali et al. (2019)], have introduced a multi-objective algorithm CS policy to provide an efficient task-resource mapping to minimize makespan and execution cost in IaaS cloud systems. Further, a hybrid gradient descent cuckoo search algorithm was suggested by combining the features of gradient descent approach and cuckoo search algorithm for scheduling meta-task in IaaS cloud [Madni, Latiff, Abdulhamid et al. (2019)]. Another CS based hybrid task scheduling method was introduced by combining the position update mechanisms of both CS and PSO algorithms to optimize makespan and budget cost considering deadline constraint [Jacob and Pradeep (2019)].

A multi-objective independent task scheduling scheme has been suggested by employing a mean Grey wolf optimization (GWO) method to minimize both makespan and energy [Natesan and Chokkalingam (2019)]. Another task scheduling method based on multi-objective Whale optimization algorithm (WOA) was introduced to deal with various QoS metrics [Reddy and Kumar (2017)]. In another study, the WOA method was applied to schedule independent tasks in the Cloud for reducing both makespan and energy consumption [Sharma and Garg (2017)]. A multi-objective WOA scheme, called as W-Scheduler for independent task scheduling in clouds was suggested for minimizing the makespan and execution cost [Sreenu and Sreelatha (2019)].

Most of the research works discussed in this section deals with the scheduling of bag-of-tasks applications where each task in the application requires a single core or CPU for execution and mostly the artificial data-sets were used for benchmarking. Further, analysis of the existing literature reveals that most of cloud scheduling algorithms either do not produce optimal results for different scheduling objectives or usually do not simultaneously consider both QoS and energy saving as objectives. In this paper, we deal with the scheduling of bag-of-tasks, where each task is a parallel HPC task consisting of a number of tasks and requiring a fixed number of cores for execution. Moreover, we have used the real workloads, which are extracted from workload traces of supercomputing facilities. We have proposed a multi-objective task scheduling algorithm by considering both makespan and energy efficiency aspects. Our proposed scheduling approach is capable of optimizing both task-order and allocation of VMs to task problems as opposed to other scheduling policies presented in this section, which optimize only resource allocation problem.

3 System model and problem definition

In this section, we describe the HPC task model, cloud datacenter model, makespan model, energy model, and fitness function underneath our approach.

3.1 Cloud data-center model and HPC tasks

The cloud computing system in this paper is an HPC Cloud that consists of a single data-center and K different classes of VMs such that VMC={VMC_1, VMC_2,…, VMC_K}. Each VM class consists of a distinct number of virtual machines (*VM*s) whereas each *VM* is represented by two tuples; a number of CPU cores and computational capacity of each core (in MIPS format). It is assumed in this paper that VMs are pre-configured from physical machines (PMs) and available to the data-center broker for allocation to waiting queue tasks.

Each arriving task in the cloud system is an embarrassingly parallel task composed of a fixed number of independent tasks. Each parallel task (also known as cloudlet) consists of two main parameters; task length (in MI) and task size (which are the number of CPU cores required to execute the task). Since our focus in this paper is on static scheduling, therefore we assume that several arriving tasks are collected in a batch and submitted to a data-center broker for assignment of pre-configured virtual machines. Therefore input to the proposed scheduling policy in the cloud data-center broker is a batch of cloudlets obtained from real-workloads and set of pre-configured virtual machines. The task scheduler will provide the mapping of VM cores to the cloudlets on the basis of the number of cores requested by each parallel task subject to the fulfillment of pre-defined objectives.

3.2 Task execution time model

The actual execution time (in seconds) of each task to be executed on the number of CPU cores is calculated using the Eq. (1). The task length parameter is obtained from the real-workload trace and the computation capacity of VM cores is acquired from the VM configuration, as shown in Tab. 1.

$$\text{Execution Time} = \frac{\text{Task Length (in MI)}}{\text{VM.NumCores x VM.Core.Capacity (in MIPS)}} \tag{1}$$

3.3 Problem definition and fitness function

3.3.1 Makespan model

The Makespan (MS) is the finish or completion time of all tasks of workload and it is calculated by Eq. (2). Finish time of a job is calculated as the sum of job start time and job execution time.

$$MS = max_{j\in J}\left(FinishTime_j\right) - min_{i\in J}\left(arrivalTime_i\right) \tag{2}$$

3.3.2 Energy model

The energy consumption (EC) by a single VM core can be expressed by Eq. (3).

$$EC\ (C_k) = \int_0^{MS} EC_{comp}\ (C_k\,, t) + EC_{idle}\ (C_k\,, t)\, dt \tag{3}$$

where $\mathrm{EC_{comp}}$ and $\mathrm{EC_{idle}}$ measure the energy consumption (in watts) during the computation and the idle period, respectively, by the CPU core $\mathrm{C_k}$ in the time-period *t*. The total energy consumption of all the CPU cores of all VMs provisioned from all physical machines of HPC cloud can be calculated by Eq. (4) as follows:

$$\mathrm{EC} = \textstyle\sum_{\mathrm{C_k} \in \mathrm{PM}} \mathrm{EC(C_k)} \quad \forall \mathrm{PM} \in \text{PMs in the datacenter} \tag{4}$$

3.3.2 Fitness function

In this paper, the static task scheduling problem on HPC Cloud is expressed as a bi-objective optimization problem for the minimization of makespan and total energy consumption of workload composed of a set of parallel tasks.

The weighted-sum-method is used to define the fitness function for the bi-objective scheduling problem, as shown in Eq. (5).

Fitness function, *ObjFn=min* (w1×MS+w2×EC) (5)

where w1 and w2 are the weights of Makespan (MS) and Energy consumption (EC) objectives, respectively. The sum of the weights of both objectives is equal to 1. Minimizing MS and EC in the fitness function, *ObjFn*, are conflicting objectives.

4. Scheduling methodology

4.1 Standard Cuckoo search

The CS algorithm is based on the unique reproduction behavior of a cuckoo bird species combined with the Lévy flight mechanism of some birds and fruit flies [Yang and Deb (2009)]. This algorithm has proved its effectiveness over many other meta-heuristics for solving a variety of single and multiple-objective problems.

First of all, the parameters of the CS algorithm are initialized and an initial random population X of N host nest (and cuckoos) is randomly generated. A random cuckoo individual X_i^d (with dimension *d*) is selected from the current generation (*G*) population and the cuckoo position is updated using the Lévy flight mechanism using Eq. (6).

$$X_{i,G+1}^d = X_{i,G}^d + \alpha \oplus \text{Lévy}(\beta) \tag{6}$$

where α is a step size whose value can be decided on the basis of the underlying optimization problem. The step size can be obtained from the Eq. (7).

$$\alpha = \alpha_0 (X_{i,G}^d - gBest(G)) \oplus \text{Lévy}(\beta) \tag{7}$$

where α_0 is a scaling factor (generally $\alpha_0 = 0.01$) and *gBest* is the best solution obtained so far. The product operator ⊕ denotes entry-wise multiplications. Lévy(*β*) is a random number, which is drawn from a Lévy distribution for large steps, as shown in Eq. (8).

$$\text{Lévy}(\beta) \sim \mathrm{u} = t^{-1-\beta}, (0 < \beta \leq 2) \tag{8}$$

In the implementation, Lévy(*β*) can be calculated as follows using Eq. (9):

$$\text{Lévy}(\beta) \sim \frac{\phi \times u}{|v|^{\frac{1}{\beta}}},\ \phi = \left(\frac{\Gamma(1+\beta) \times \sin\left(\pi \times \frac{\beta}{2}\right)}{\Gamma((1+\beta)/2 \times \beta \times 2^{(}\beta-1)/2)} \right)^{\frac{1}{\beta}} \tag{9}$$

where *β* is a constant and set to 1.5 in the standard software implementation of CS [Yang and Deb (2010)], and u and v are random numbers drawn from a normal distribution with a mean of 0 and a standard deviation of 1, and Γ is a gamma function.

During each generation, a fixed fraction p_a of solutions from the population is abandoned and replaced by random solutions to avoid local optima. Solutions are ranked using fitness function and the current global best solution is obtained. The whole procedure is repeated until the termination condition is reached.

4.2 Standard particle swarm optimization (PSO) algorithm

Each individual ($X_{i,G}^{d}$) in PSO is known as Particle, which represents the candidate solution for the undertaken problem, where *i* is the index of individual and *d* is the dimension of the individual. Each particle consists of two tuples; velocity and position. Particles are randomly initialized in the beginning to build an initial population. The fitness of the particles is calculated from the defined objective function.

For every particle in the population, velocity ($V_{i,G}^{d}$) and position ($X_{i,G}^{d}$) of the particle are updated based on its previous best position and global best position of the whole population, as shown in Eqs. (10)-(12).

$$V_{i,G+1}^{d} = W_G V_{i,G}^{d} + c_1 r_1 (pBest_{i,G}^{d} - X_{i,G}^{d}) + c_2 r_2 (gBest(g) - X_{i,G}^{d}) \tag{10}$$

$$W_{G+1} = W_G * \alpha \tag{11}$$

where c1 and c2 are self-recognition and social constant respectively and r1, r2 two random numbers generated uniformly between 0 and 1. W_G is inertia weight and α is a decrementing factor randomly generated between 0 and 1. Personal best position and global best particle position are found using Eqs. (13) and (14).

$$X_{i,G+1}^{d} = X_{i,G}^{d} + V_{i,G+1}^{d} \tag{12}$$

$$\boldsymbol{pBest_{i,G}^{d}} = \begin{cases} \boldsymbol{X_{i,G}^{d}}; & \boldsymbol{F(X_{i,G+1}^{d}) < F(X_{i,G}^{d})} \\ \boldsymbol{X_{i,G+1}^{d}}; & \boldsymbol{F(X_{i,G+1}^{d}) > F(X_{i,G}^{d})} \end{cases} \tag{13}$$

$$gBest_G = minarg_{\forall i}(pBest_{i,G}^{d}) \tag{14}$$

This procedure of velocity and position update is repeated until the maximum generations condition is achieved.

4.3 The proposed CSPSO scheduling policy

The CS algorithm requires very few control parameters and possesses powerful global-search ability. In CS, the solution updating mechanism using Lévy flights provides high randomness, which allows the solution search process to quickly bounce from one area to another in the solution space leading to robust global-search ability. However, this high randomness sometimes also starts an obtuse search resulting in a slowdown of the convergence rate and reduction of searching ability when the search reaches near to optimal solution due to lack of solution diversity [Chi, Su, Qu et al. (2019)]. On the other hand, PSO has been known for its faster convergence and global searching ability due to its ability to memorize previous solutions and unique information exchange mechanisms between PSO particles after the end iteration [Mohanapriya and Kalaavathi (2019)]. In order to improve the convergence speed and increase the solution diversity of CS, we have hybridized the PSO algorithm with CS algorithm for task scheduling in cloud computing infrastructures to optimize both makespan and energy consumption. In every generation in the CSPSO algorithm, the fitted individuals from CS and PSO phases are mixed in a unique way leading to sustained solution diversity and avoidance of premature convergence. These unique abilities of the proposed CSPSO scheduling policy resulted in efficient global and local search, faster convergence, and better diversity of solutions throughout the CSPSO algorithm.

4.3.1 Solution encoding

The most important step in the implementation of the proposed multi-objective hybrid policy CSPSO for performing task scheduling is the solution representation. Task scheduling problem generally consists of two sub-problems; 1) Task ordering problem, to decide the order of execution of queued tasks, and 2) Allocation problem, to decide the mapping of computational resources to tasks. Therefore an individual X, which represents a task scheduling solution, is characterized by two decision variables; task-order component (O) and allocation component (A). X^{O+A} represents an individual X_i solution for both task-ordering and allocation sub-problems.

Both CS and PSO algorithms are originally designed to deal with optimization problems with real-value continuous solutions. However, task scheduling sub-problems such as task-ordering and allocation problem involves discrete-value task numbers and virtual machines. Therefore we need a mechanism to convert real-value solutions produced by meta-heuristics to discrete values required by the task scheduling problem. Initially, we represent the job vector part of the solution by continuous solution as X^O, which is converted to discrete-valued task execution order solution X^{*O} using the Smallest Position Value (SPV) method [Tasgetiren, Liang, Sevkli et al. (2006)].

Allocation part (X^A) of the task scheduling solution is initially represented by the VM availability matrix (AM) scheme, which is adapted from the proven blacklist allocation mechanism by Gabaldon et al. [Gabaldon, Lerida, Guirado et al. (2017)] and mapping method by Srichandan et al. [Srichandan, Kumar and Bibhudatta (2018)]. Each cell entry in the VM availability matrix is a real value between 0 and 1, providing complete unavailability of all VMs to complete availability of all VMs of a particular VM class, reserving a few VMs of each PM for queued tasks. Discrete-value allocation part (X^{*A}) of solution is obtained by applying the fitness-aware resource allocation heuristic on availability matrix and discrete task-order vector X^{*O}.

We adapted and modified the allocation algorithm used in Vila et al. [Vila, Guirado, Sergi et al. (2019)] to improve the discrete resource allocation to tasks for the current research paper. The significant differences between our resource allocation heuristic and the original allocation algorithm are the level at which availability of VMs is decided and the way of selecting VMs for allocation to tasks. The improved resource allocation heuristic hereafter called Fitness-Aware Resource Allocation (FARA) heuristic is explained using pseudo-code in Algorithm 1. The FARA heuristic starts by updating the VMs availability in availability matrix (AM) for each task by multiplying the corresponding AM (*task, vmClass.vms*) pair value with the total VMs of the corresponding *vmClass* (Line 4). In the next step, for each task, it prepares a list of all those *virtual machines* which have CPU cores greater than or equal to the number requested by the current task (Lines 5-10). Further, a temporary allocation is done for (task, shortlisted *vm) pair* and the fitness function (*ObjFn*) value using Eq. (5) for all shortlisted *vm'*s is calculated (Lines 11-14).

Algorithm 1: Fitness-aware Resource Allocation (FARA) Heuristic

Input: VMC: Set of different classes of virtual machines (vmClass), VM: Total virtual machines in the system, vmClass.vms: Virtual machines (vm) in each class, TK: Set of queued tasks provided by discrete cloudlet-order vector X_i^O

Require: VMs availability represented by the availability matrix (AM)

Output: Allocation list (A): a pair of ($task \in$ TK, $vm \in$ VM)

```
1. for each $task \in$ TK do
    2. Declare sets: vmClass.vms.available, vm.shortlisted, vm.final, TempA: a temporary pair of ($task \in$ TK, $vm \in$ VM)
3.     for each $vmClass \in$ VMC do
4.         vmClass.vms.available=vmClass.vms*AM (task, vmClass)
5.             for each $vm \in vmClass.vms.available$
6.                 if (vm.cores.available$\geq$task.coresReqd)
7.                 vm.shortlisted$\leftarrow$vm.shortlisted $\cup$ VM
8.             end if
9.     end for
10. end for
11.    for each $vm \in vm.shortlisted$ do
12.        TempA$\leftarrow$TempA U (task, vm)
13.        ObjFn.vm$\leftarrow$ObjFn.vm (TempA)
14.    end for
15.    vm.shortlisted=argmin$_{\forall vm \in vm.shortlisted}$(ObjFn.vm)
16.    vm.final=best-fit (vm.shortrlisted, vm.cores.available)
17.    A$\leftarrow$A $\cup$ (task, vm.final)
18. end for
```

In the next step, the best-fit method is applied to select a single *vm* out of multiple short-listed *vms* (Line 15), which leaves the least free cores after current task allocation to reduce external fragmentation. In the last step, the selected VM cores are finally allocated and the allocation list (A) is updated. This procedure continues until all tasks are mapped to virtual machines. If we run out of CPU cores during the VM allocation process, the resource allocation heuristic predicts the first task to finish and then releases the allocated VM cores from the predicted task for the queued tasks.

The proposed, multi-objective cuckoo search particle swarm optimization algorithm (CSPSO) is explained using the pseudo-code given in Algorithm 2. The major stages in the pseudo-code of CSPSO based scheduling policy are discussed as follows:

4.3.2 Initial stage- solution encoding and population Initialization

In this step, parameters of the underlying task scheduling problem and meta-heuristics such as the number of parallel-tasks, configuration of VMs are initialized. An objective function is defined using Eq. (5). Scheduling solution is represented using the encoding scheme discussed in Section 4.3.1. The initial population of N real-value individuals is generated randomly and the corresponding discrete-form population is generated using the SPV method and FARA heuristic discussed in the Section 4.3.1. The fitness of each solution in the population is evaluated and the overall best solution is recorded.

Algorithm 2: Cuckoo Search Particle Swarm Optimization (CSPSO) Algorithm

Input: Tasks, Virtual machines, scheduling solution encoding, objective function: *ObjFn* (X),

Variables: X: Real-value population, X^*: Equivalent integer-value population, *G*: Index of current generation; *NP*: Population size, *MaxGeneration*: maximum number of generations of CSPSO

Output: Best solution (schedule) representing task-order and allocation of VMs to tasks

```
1: Initialize parameters of CS algorithm and PSO algorithm
2: $G$=0                //initial generation
3: $X_{i,G} \leftarrow X_{i,G}^{O+A}$       //real-value solution (for both task-order (O) and allocation (A) components)
4: $X_{i,G}^* \leftarrow X_{i,G}^{*(O+A)}$ //integer-value solution (for both task-order (O) and allocation (A) components)
5: Generate initial real-value population X and equivalent integer-form population $X^*$ using SPV
   rule and FARA heuristic
6: Fitness ($X^*$)←ObjFn ($X^*$)          // Evaluate the fitness of population using objective function
7: $X^{*B}$=argmin (Fitness ($X_{i,G}^*$))        // global best solution among whole population
8: while ($G$<MaxGeneration)
9:    Split the real-value population X with size NP into two equal random halves; sub-population1
      with range [1...NP/2] and sub-population2 with range [NP/2+1...NP]
10:       for each individual $X_{i,G}$ in sub-population1 do                //CS phase starts
11:               for both task-order and allocation dimensions of individual $X_{i,G}$ do
12:                       Apply CS search on individual $X_{i,G}$ using Eqs. (6)-(9) to obtain $X_{i,G+1}$
13:                       Update position if $X_{i,G+1}$ is better than $X_{i,G}$ and mark it as pBest. $X_i$
14:                       Update velocity of individual $X_{i,G+1}$ using Eq. (15) to produce $V_{i,G+1}$
15:               end for
16:        end for
17:  Update sub-population1 by replacing a fraction of worst individuals by OBL random walks
18:                                                                    //CS phase ends
19:       for each individual $X_{i,G}$ in sub-population2  do               //PSO phase starts
20:               for both task-order and allocation dimensions of individual $X_{i,G}$ do
21:                       Update velocity of individual $X_{i,G}$ using Eqs. (10)-(11) to produce $V_{i,G+1}$
22:                        Apply PSO search on individual $X_{i,G}$ using Eq. (12) to obtain $X_{i,G+1}$
```

```
23:                 Update pBest.$X_i$ of individual $X_{i,G+1}$ using Eq. (13)
24:         end for
25:     end for
26:                                                     //PSO phase ends
27: //Hybridization phase starts
28: Combine the updated sub-population1 and sub-population2 into new population of size NP
29: //Hybridization phase ends
30: Obtain integer-value population $X^*$ using SPV and FARA heuristic
31: Evaluate the fitness of new population using ObjFn
32: Update the integer and real-value global best solution of the current population of size NP
33: end while
34: Record the best solution $X^{*B}$ (schedule) found
```

4.3.3 Population partitioning and updating stage

The real-value population is divided randomly into two equal sub-populations. CS updates the position and velocity of individuals of sub-population1 using a levy flight mechanism, and PSO updates the velocity and position of particles of sub-population2. In order to increase the local-search efficiency in cuckoo search, the worst solutions are replaced by solutions obtained from the opposition based learning (OBL) technique instead of standard random walks. OBL is simple yet powerful technique to generate opposite solutions [Tizhoosh (2005)]. During the random equal partitioning of the whole population after every alternate generation, some of the solutions processed by the CS phase in the previous generation may become part of the PSO sub-population in the next generation.

Since CS does not have velocity mechanism, but the PSO method requires velocity to update the position of solutions. In the current situation, the PSO phase may not be able to improve the position of solution due to the unavailability of velocities of those solutions, which were earlier part of the cuckoo sub-population. Therefore we introduced a velocity update mechanism using Eq. (15) in the CS phase to add velocity feature to solutions in cuckoo search.

$$V_{i,G+1} = \begin{cases} X_{i,G+1} - gBest_G, & F(X_{i,G+1}) > gBest_G \\ gBest_G - X_{i,G+1}, & F(X_{i,G+1}) < gBest_G \end{cases} \tag{15}$$

4.3.4 Hybridization stage

The new sub-populations obtained from both CS and PSO phases are combined to build a new population. If the maximum number of iterations condition reached is false, then Step 2 is called again. This procedure of population division, individual position updating by CS and PSO phases and population hybridization continues till maximum iteration condition is satisfied. Finally, at the end of the proposed CSPSO algorithm, a global best solution is found, which represents the optimal schedule, which provides optimized task ordering and resource allocation to tasks.

5 Experimentation and results

In this section, we have conducted an extensive number of experiments to analyze the efficiency of the proposed CSPSO scheduling algorithm over other existing scheduling techniques. FCFS, MOCS, MOPSO, and CSPSO scheduling algorithms are implemented in the CloudSim 3.0.3 simulator with the help of Java and JMetal 5.4 meta-heuristic framework for multi-objective optimization [JMetal (2019)]. The computer system used for experimentation is i7-8550U @ 1.80-2.0 GHz (8 Cores), 16 GB RAM running over Windows 10. The benchmarking results of Min-Min, Hill-Climbing, and BLEMO algorithms have been obtained from the corresponding author of selected base paper [Vila, Guirado, Lerida et al. (2019)].

5.1 Real-workloads and cloud configuration

The input workloads for the benchmarking of scheduling algorithms are acquired from workload traces of two real-supercomputing sites; CEA-Curie and HPC2N. Workload logs of both CEA-Curie and HPC2N traces are maintained by Feitelson [Feitelson (2005)]. In our experimentation, we have used 20 workloads (10 workloads are extracted from each of the CEA-Curie and HPC2N workload traces). Each workload contains 200 tasks. In this paper, the Cloud computing system modeled for experimentation in CloudSim is consisting of one data center with five classes of pre-configured VMs. The same VM configuration is taken for the experimentation as used in our baseline research paper [Vila, Guirado, Lerida et al. (2019)] and it is shown in Tab. 1.

Table 1: VM configuration used for experiments in CloudSim

VM Class	VM instance id	# VMs	# vCPU/ Cores per VM	MIPS	CPU model	Energy during idle time (W/h)	Energy during comp. time (W/h)
Class 1	T2.nano	20	1	3400	Xeon E5-2637 V4	23.625	33.75
Class 2	T2.xlarge	10	4	2600	Xeon E5-2623 V4	59.5	85
Class 3	T2.2xlarge	8	8	2100	Xeon E5-2620 V4	59.5	85
Class 4	M5.4xlarge	6	16	2500	Xeon Plat. 8180 M	82	117.14
Class 5	M4.10xlarge	4	40	2400	Xeon E5-2686 V4	225.55	322.22

5.2 Baseline scheduling algorithms

The performance of the proposed CSPSO scheduling policy has been compared with popular existing single-objective heuristics (FCFS, Hill-Climbing, and Min-Min) and multi-objective meta-heuristic algorithms (MOPSO, MOCS and BLEMO). *FCFS* schedules the tasks in the waiting queue using First Come First Serve approach. *Hill-Climbing* continuously scans the solution space towards increasing elevation to seek the best solution to the problem and terminates at a point where no other neighbor has a higher value. *Min-Min* is a variant of traditional Min-Min algorithm which selects the task with maximum completion time and assigns it to appropriate computing resource with the objectives to reduce overall makespan and efficient resource utilization [Patel, Mehta and Bhoi (2015)]. *MOPSO* is a multi-objective version of the PSO algorithm used by many task scheduling research studies [Alkayal, Jennings and Abulkhair (2016)]. *MOCS* is a multi-objective version of the cuckoo search (CS) algorithm implemented in many research studies focused on task scheduling in cloud computing systems [Madni, Latiff, Ali et al. (2019)]. *BLEMO* is a multi-objective scheduling policy that uses a blacklist allocation scheme and GWASF-GA meta-heuristics algorithm to decide the mapping of VMs to cloudlets [Vila, Guirado, Lerida et al. (2019)]. FCFS and Min-Min are deterministic and non-iterative, whereas the rest of the algorithms are stochastic and iterative.

5.3 Comparative results

5.3.1 Convergence analysis of meta-heuristic approaches

Before conducting simulation experiments, we have conducted a number of independent experiments by varying population size and the number of generations with 200 tasks to determine optimal parameters for the scheduling policies. The final meta-heuristics parameters for experimentation for both HPC2N and CEA-Curie workloads as follows:

-MOPSO policy

● Population size: 60, Generations: 45, Inertia weight (ω): 0.9, random numbers (r1 and r2): 0.65 and 0.35, and Coefficients (c1 and c2): 2.0 and 2.0

-MOCS policy

● Population size: 60, Generations: 50

● CS parameters: step size, α_0: 0.01, fraction of worst individuals (p_a): 0.25

-CSPSO policy

● Population size: 50, Generations: 50

● CS parameters: step size, α_0: 0.01, fraction of worst individuals (p_a): 0.25

● PSO parameters: Inertia weight (ω): 0.9, random numbers (r1 and r2): 0.65 and 0.35, and Coefficients (c1 and c2): 2.0 and 2.0

-BLEMO policy

● Population size: 60, Generations: 40, Crossover rate: 0.33 and Mutation rate: 0.10

5.3.2 Makespan and energy consumption results of different workloads

This section presents the makespan and energy consumption results of all the scheduling policies executed on both CEA-Curie and HPC2N workloads. Each scheduling experiment was repeated 30 times for the same input conditions, and the average of 30 readings was recorded to remove randomness from collected experimental results.

Fig. 1 shows that the proposed CSPSO policy generated optimal makespan value among all tested scheduling policies for W0-W6 workloads. For W7-W9 workloads, proposed CSPSO is the second-best policy after Min-Min policy. FCFS policy produced the worst results, followed by the Hill-Climbing policy. BLEMO policy ended up with a second and third spot in most of the tested workload instances for the makespan objective.

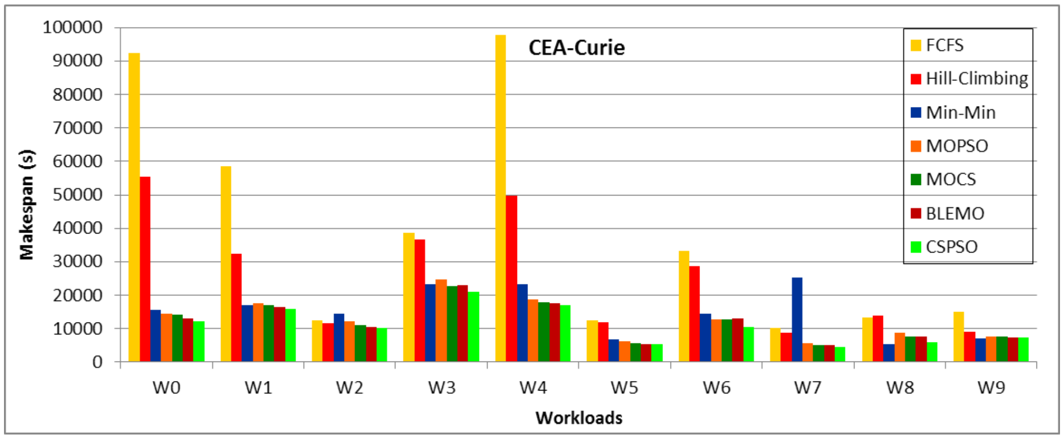

Figure 1: Makespan of scheduling policies for CEA-Curie workloads

Fig. 2 indicates that the CSPSO policy yielded optimal energy consumption values for 80% workloads among all scheduling policies for CEA-Curie trace followed by BLEMO, MOCS, and Min-Min policy.

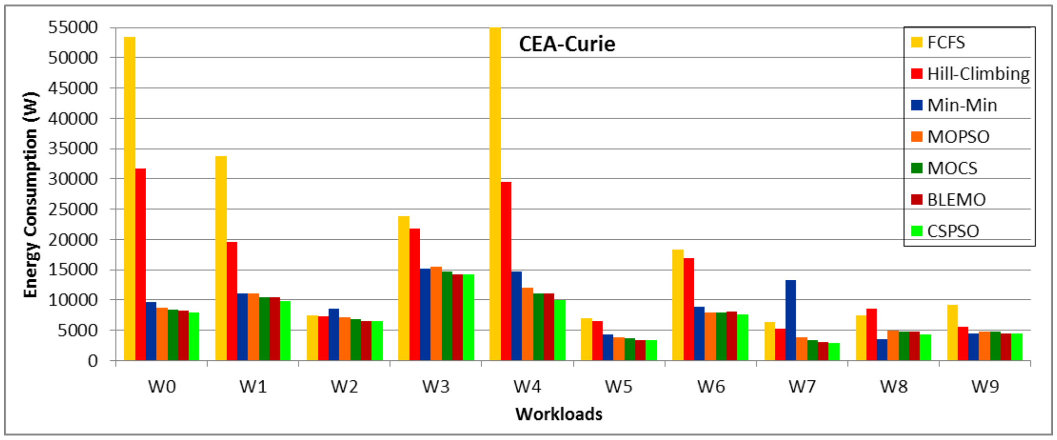

Figure 2: Energy consumption of scheduling policies for CEA-Curie workloads

Figs. 3 and 4 present the makespan and energy consumption results for all tested scheduling policies for HPC2N workloads. The proposed CSPSO policy produced lowest makespan and energy consumption as compared to other policies for most of the HPC2N workloads followed by BLEMO, MOCS, MOPSO and Min-Min policy. FCFS and Hill-Climbing policies yielded the worst results since these policies failed to exploit the heterogeneity of virtual machines. FCFS also could not produce good results since it did not employ any logic for task ordering and resource allocation to tasks.

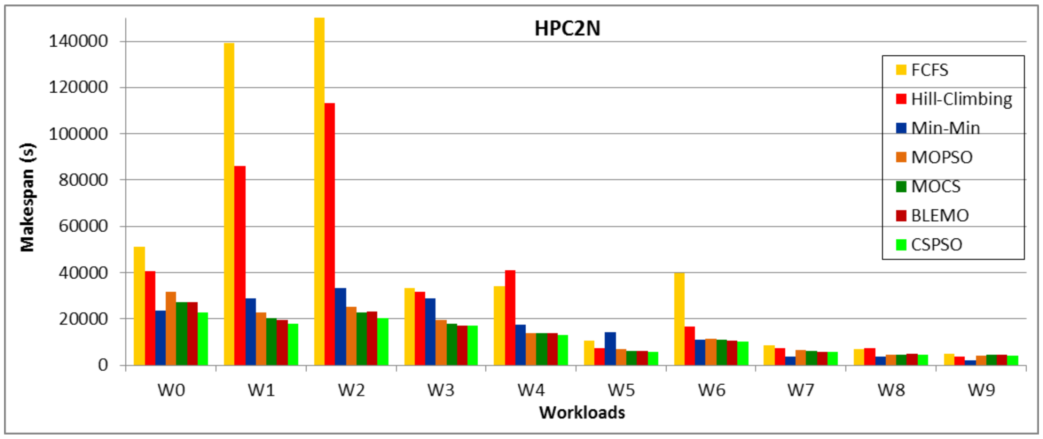

Figure 3: Makespan of scheduling policies for HPC2N workloads

5.4 Overall experimentation results and analysis

Finally, the box plots are used to explain the overall experimentation results of all scheduling policies, which were collected from both CEA-Curie and HPC2N workloads. Figs. 5 and 6 show the box plot for overall makespan and energy consumption results of all scheduling policies by executing 10 workloads of CEA-Curie trace.

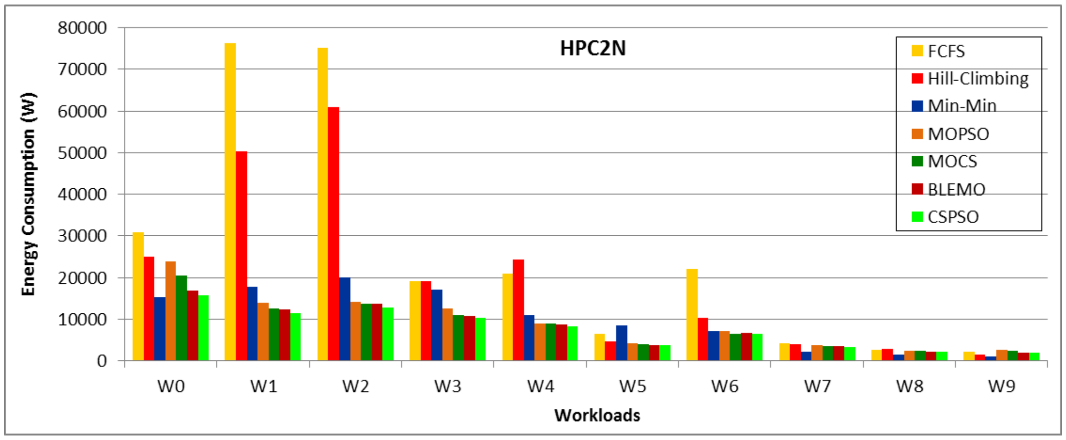

Figure 4: Energy consumption of scheduling policies for HPC2N workloads

It is evident from box plot that among all scheduling policies, the proposed CSPSO policy produced both lowest median (as shown by the dark horizontal black line in the interquartile range (IQR)) makespan and energy consumption results as well as lowest mean (represented by '+' sign in IQR) makespan and energy consumption values. The proposed CSPSO policy also produced lower minimum makespan and energy consumption values, as indicated by the lowest whisker in the box plot. FCFS and Hill-climbing policies produced worst and second-worst makespan and energy consumption values (represented by worst mean and median values) respectively. The CSPSO policy also obtained better overall minimum, maximum, mean, and median values of makespan and energy consumption objectives among all scheduling policies for CEA-Curie workloads as shown in Figs. 7 and 8.

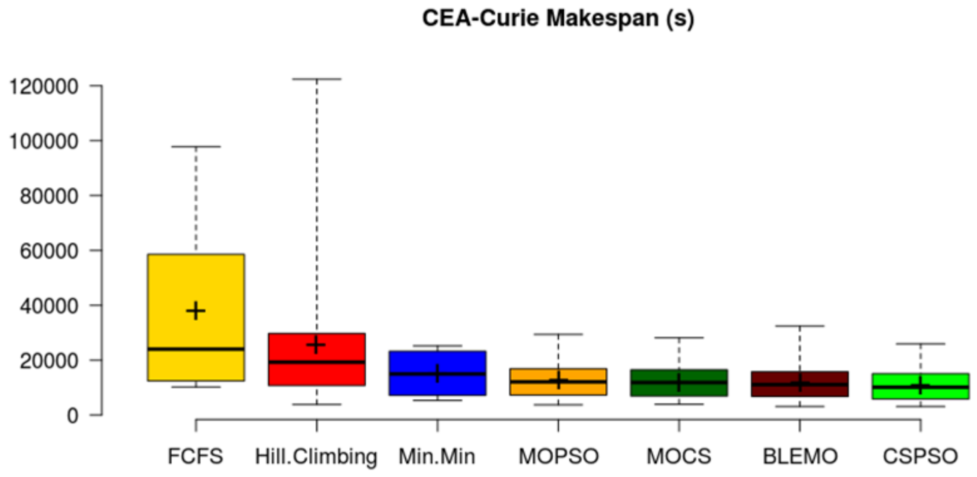

Figure 5: Makespan box plot of all experiments with CEA-Curie workloads

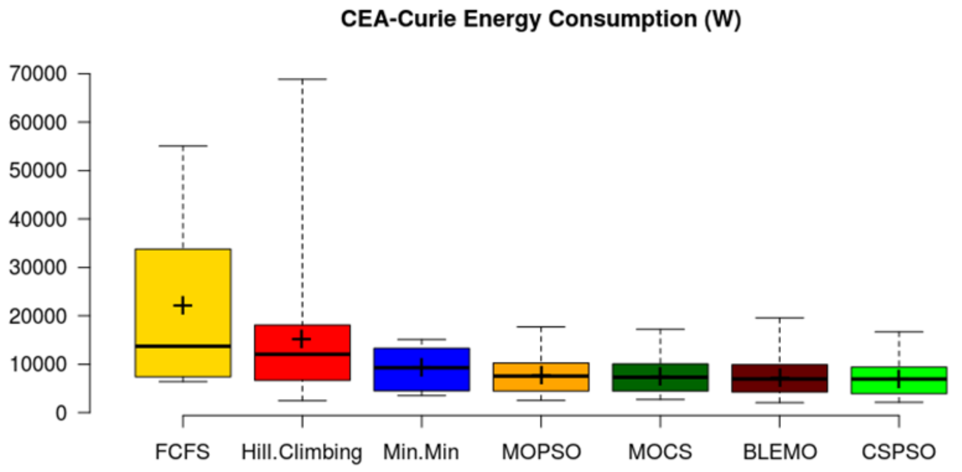

Figure 6: Energy consumption box plot with CEA-Curie workloads

Finally, for quantitative analysis, the mean values of overall experimental results (of both makespan and energy consumption) for all scheduling policies and performance improvement rate percentage (PIR% calculated using Eq. (17)) of proposed CSPSO scheduling policy over other baseline scheduling policies are shown in Tab. 2.

$$PIR(\%) = \frac{Algorithm\ - CSPSO\ Algorithm}{CSPSO\ Algorithm} \times 100\% \tag{17}$$

It is evident from the Tab. 2 that among all scheduling policies, the CSPSO scheduling policy has achieved minimum mean values for both makespan and energy consumption objectives for both CEA-Curie and HPC2N workloads.

Table 2: Overall mean values and PIR% of the CSPSO over other scheduling policies

Policy	**FCFS**	**Hill-Climbing**	**Min-Min**	**MOPSO**	**MOCS**	**BLEMO**	**CSPSO**
CEA-Curie							
Makespan (s)	38395.91	25811.11	15285.12	12838.08	12189.95	11905.82	**11000.33**
PIR% of CSPSO over	+249.04%	+134.64%	+38.95%	+16.71%	+10.81%	+8.23%	---
Energy Consumption (W)	22188.44	15318.96	9371.26	7972.46	7602.63	7431.35	**7122.73**
PIR% of CSPSO over	+211.52%	+115.07%	+31.57%	+11.93%	+6.74%	+4.33%	---
Simulation Time (s)	**0.39**	0.73	0.45	10.29	12.35	15.75	10.73
HPC2N							
Makespan (s)	48199.21	36972.56	16643	14670.64	13404.35	13285.63	**12154.53**
PIR% of CSPSO over	+296.55%	+204.19%	+36.93%	+20.70%	+10.28%	+9.31%	---
Energy Consumption (W)	26024.09	21123.09	10188.52	9407.02	8592.42	8088.15	**7664.85**
PIR% of CSPSO over	+239.53%	+175.58%	+32.93%	+22.73%	+12.10%	+5.52%	---
Simulation Time (s)	**0.41**	0.93	0.44	14.24	17.58	26.32	15.07

As far as PIR% is concerned, the CSPSO records the makespan improvement in the range 8.23-249.04% and 4.33-211.52% improvement in energy consumption over rest of the scheduling policies for CEA-Curie workloads. +sign against PIR% indicates that all scheduling algorithms other than CSPSO policy produced increased makespan and energy consumption mean results than CSPSO policy.

Our proposed CSPSO scheduling policy obatined makespan reduction in the range of 9.31-296.55% and improved energy consumption in the range of 5.52-239.53% over other scheduling policies for HPC2N workloads. The proposed CSPSO policy produced quality scheduling solutions with acceptable convergence speed represented by simulation time for both workloads, as shown in Tab. 2. The time taken by the proposed CSPSO policy is better than the MOCS and BLEMO meta-heuristic based scheduling policies. The higher simulation time of the CSPSO as compared to FCFS and Min-Min heuristics is also acceptable with respect to significant makespan and energy-consumption improvement of the CSPSO over these heuristics.

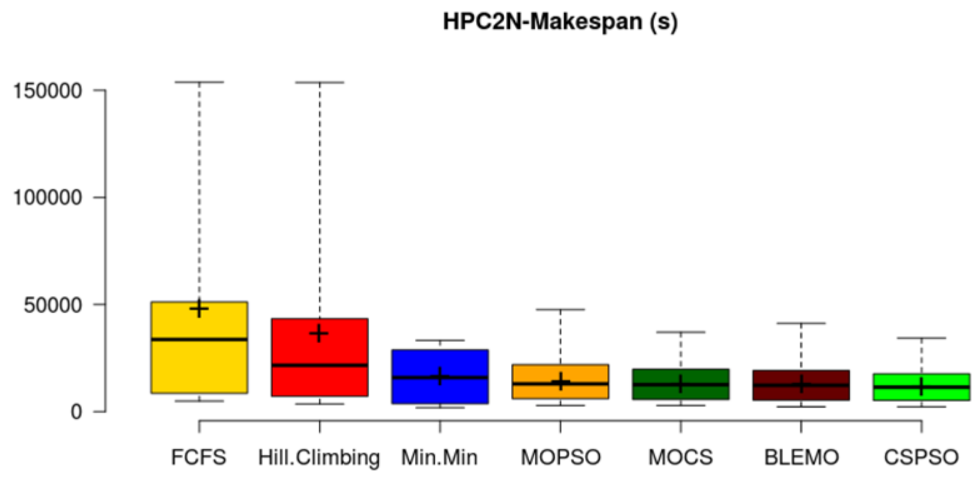

Figure 7: Makespan box plot of all experiments with HPC2N workloads

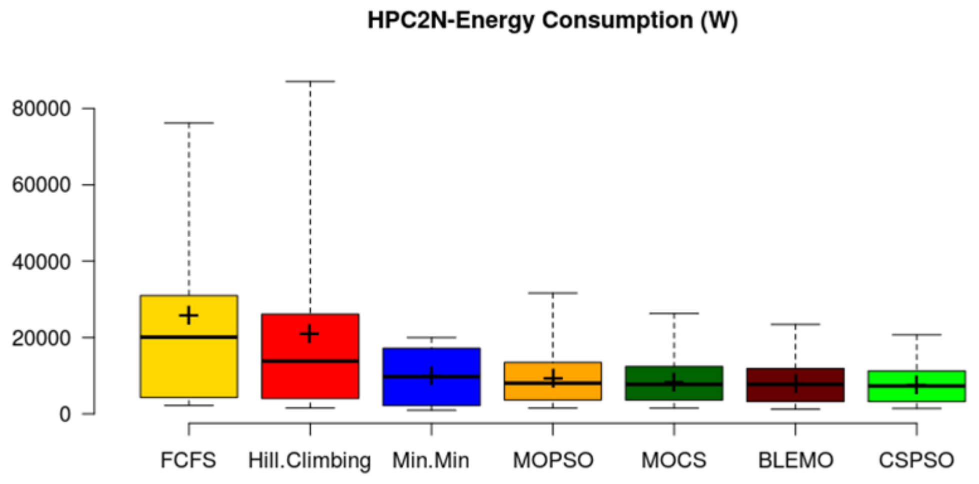

Figure 8: Energy consumption box plot of all experiments with HPC2N workloads

6 Conclusions

The user base, the size and scale of cloud computing environments is increasing due to cost-effectiveness, anytime-anywhere service, and pay-as-you-go model characteristics. Due to these characteristics, HPC users nowadays are migrating from traditional clusters and grids to cloud computing systems to execute HPC applications. However, with these trends, associated energy consumption is also rising to a greater extent leading to higher carbon emissions and system provider expenses. This situation calls for the development of efficient schedulers to allocate the resources to HPC applications for meeting both QoS-specific and energy-saving expectations. In this paper, a hybrid multi-objective policy, CSPSO, is proposed for scheduling a set of parallel tasks in IaaS cloud systems to meet QoS and energy-efficiency expectations. The proposed CSPSO scheduling policy effectively combines the searching qualities of both CS and PSO algorithms resulting in the increase in the searching efficiency, better convergence rate, and solution diversity over successive generations, which finally helped to produce quality task schedules and resource allocation solutions. The CSPSO policy uses a fitness-aware resource allocation heuristic to efficiently allocate resources to tasks in the task vector obtained from the best real real-value solution of the CSPSO policy.

The proposed CSPSO policy was evaluated against many well-known heuristics and meta-heuristic scheduling techniques using twenty different workloads extracted from two real workload traces. The obtained experimental results have proved the efficacy of our CSPSO policy over other tested scheduling policies by obtaining promising performance improvement in terms of QoS objective makespan and energy consumption. Another reason for the performance improvement obtained by our proposed CSPSO policy is the ability to optimize both task-ordering and allocation issues of the task scheduling problem as opposed to only allocation optimization done by other tested scheduling policies. In the future, we plan to evaluate our proposed policy against other robust meta-heuristic approaches with different supercomputing workloads using different objective functions.

Availability of Data and Materials: All the input and output data file links are available in the manuscript.

Acknowledgement: Authors acknowledge the help of Sergi Villa Almenara, Department of Computer and Industrial Engineerings, Universitat de Lleida, Lleida, Spain, for providing few comparative benchmarking results.

Funding Statement: The author(s) received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Alkayal, E. S.; Jennings, N. R.; Abulkhair, M. F. (2016): Efficient task scheduling multi-objective particle swarm optimization in cloud computing. *Proceedings of the IEEE 41st Conference on LCN Workshops, Dubai,* pp. 17-24.

Ayanoglu, E. (2019): Energy-efficiency in data centers, *IEEE ComSoc Technical Committees Newsletter*, pp. 1-8.

Babu, D.; Krishna, P. (2013): Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing*, vol. 13, pp. 2292-2303.

Chi, R.; Su, Y.; Qu, Z.; Chi, X. (2019): A hybridization of cuckoo search and differential evolution for the logistics distribution center location problem. *Mathematical Problems in Engineering*, pp. 1-16.

EC2-AWS Instances (2020): Amazon EC2 instance types-Amazon Web services. *Amazon Web Services, Inc.* https://aws.amazon.com/ec2/instance-types/.

Elaziz, M. A.; Xiong, S.; Jayasena, K. P. N.; Li, L. (2019): Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution. *Knowledge-Based Systems*, vol. 169, pp. 39-52.

Fang, Y.; Xia, X.; Ge, J. (2019): Cloud computing task scheduling algorithm based on improved genetic algorithm. *IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference*, pp. 852-856.

Feitelson, D. (2005): *Parallel Workloads Archive Available online:*

https://www.cse.huji.ac.il/labs/parallel/workload/.

Gabaldon, E.; Lerida, J. L.; Guirado, F.; Planes, J. (2017): Blacklist multi-objective genetic algorithm for energy saving in heterogeneous environments. *Journal of Supercomputing*, vol. 73, pp. 354-369.

Guo, Q. (2017): Task scheduling based on ant colony optimization in cloud environment, *AIP Conference Proceedings, Busan, South Korea*.

Hemasian-Etefagh, F.; Safi-Esfahani, F. (2019): Dynamic scheduling applying new population grouping of whales meta-heuristic in cloud computing. *Journal of Supercomputing*, vol. 75, pp. 6386-6450.

Ilager, S.; Ramamohanarao, K.; Buyya, R. (2019): ETAS: energy energy and thermal-aware dynamic virtual machine consolidation in cloud data center with proactive hotspot mitigation. *Concurrency and Computation: Practice and Experience*, vol. 31, no. 17, pp. 1-15.

Jacob, T. P.; Pradeep, K. (2019): A multi-objective optimal task scheduling in cloud environment using Cuckoo particle swarm optimization. *Wireless Personal Communications*, vol. 109, pp. 315-331.

Jena, R. K. (2017): Task scheduling in cloud environment: a multi-objective ABC framework. *Journal of Information & Optimization Sciences*, vol. 38, pp. 1-19.

JMetal 5 (2019): jMetal 5 web site available online: http://jmetal.github.io/jMetal/.

Kumar, M.; Sharma, S. C. (2018): PSO-COGENT: cost cost and energy efficient scheduling in cloud environment with deadline constraint. *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 147-164.

Li, G.; Wu, Z. (2019): Ant colony optimization task scheduling algorithm for SWIM based on load balancing. *Future Internet*, vol. 11, no. 4, pp. 1-18.

Madni, S. H. H.; Abd Latiff, M. S.; Abdulhamid, S. M.; Ali, J. (2019): Hybrid gradient descent cuckoo search (HGDCS) algorithm for resource scheduling in IaaS cloud computing environment. *Cluster Computing*, vol. 22, pp. 301-334.

Madni, S. H. H.; Abd Latiff, M. S.; Abdullahi, M.; Abdulhamid, S. M.; Usman, M. J. (2017): Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment. *PLoS One*, vol. 12, no. 5, pp. 1-26.

Madni, S. H. H.; Latiff, M. S. A.; Ali, J.; Abdulhamid, S. M. (2019): Multi-objective-oriented Cuckoo search optimization-based resource scheduling algorithm for clouds. *Arabian Journal for Science and Engineering*, vol. 44, pp. 3585-3602.

Mansouri, N.; Mohammad Hasani Zade, B.; Javidi, M. M. (2019): Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory. *Computers & Industrial Engineering*, vol. 130, pp. 597-633.

Moghaddam, S. K.; Buyya, R.; Kotagiri, R. (2019): Performance-aware management of cloud resources: a taxonomy and future directions. *ACM Computing Surveys*, vol. 54, no. 2, pp. 1-37.

Mohanapriya, N.; Kalaavathi, B. (2019): Adaptive image enhancement using hybrid particle swarm optimization and watershed segmentation. *Intelligent Automation and Soft Computing*, vol. 25, no. 4, pp. 663-672.

Motlagh, A. A.; Movaghar, A.; Rahmani, A. M. (2020): Task scheduling mechanisms in cloud computing: a systematic review. *International Journal of Communication Systems*, vol. 33, pp. 1-23.

Natesan, G.; Chokkalingam, A. (2019): Task scheduling in heterogeneous cloud environment using mean grey wolf optimization algorithm. *ICT Express*, vol. 5, pp. 110-114.

Navimipour, J. N.; Milani, S. F. (2015): Task scheduling in the cloud computing based on the Cuckoo search algorithm. *International Journal of Modeling and Optimization*, vol. 5, no. 1, pp. 44-47.

Netto, M. A. S.; Calheiros, R. N.; Rodrigues, E. R.; Cunha, R. L.; Buyya, R. (2018): HPC cloud for scientific and business applications: taxonomy, vision, and research challenges. *ACM Computing Surveys*, vol. 51, no. 1, pp. 1-29.

Patel, G.; Mehta, R.; Bhoi, U. (2015): Enhanced load balanced min-min algorithm for static meta task scheduling in cloud computing. *Procedia Computer Science*, vol. 57, pp. 545-553.

Qin, X., Yang, Z., Li, W., Yang, Y. (2013): Optimized task scheduling and resource allocation in cloud computing using PSO based fitness function. *Information Technology Journal*, vol. 12, pp. 7090-7095.

Rastkhadiv, F.; Zamanifar, K. (2016): Task scheduling based on load balancing using artificial bee colony in cloud computing environment. *International Journal of Advance Biotechnology Research*, vol. 7, pp. 1058-1069.

Reddy, G. N.; Kumar, S. P. (2017): Multi objective task scheduling algorithm for cloud computing using whale optimization technique. *International Conference on Next Generation Computing Technologies*, pp. 286-297.

Shojafar, M.; Javanmardi, S.; Abolfazli, S.; Cordeschi, N. (2015): FUGE: a joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method. *Cluster Computing*, vol. 18, pp. 829-844.

Shojafar, M.; Kardgar, M.; Hosseinabadi, A. A. R.; Shamshirband, S.; Abraham, A. (2015): TETS: a Genetic-based scheduler in cloud computing to decrease energy and makespan. *Proceedings of the 15th International Conference HIS on Hybrid Intelligent Systems*, pp. 103-115.

Sreenu, K.; Sreelatha, M. (2019): W-Scheduler: whale optimization for task scheduling in cloud computing. *Cluster Computing*, vol. 22, pp. 1087-1098.

Srichandan, S.; Kumar, T. A.; Bibhudatta, S. (2018): Task scheduling for cloud computing using multi-objective hybrid bacteria foraging algorithm. *Future Computing and Informatics Journal*, vol. 3, pp. 210-230.

Tasgetiren, F. M.; Liang, Y. C.; Sevkli, M.; Gencyilmaz, G. (2006): Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem. *International Journal of Production Research*, vol. 44, pp. 4737-4754.

Tizhoosh, H. R. (2005): Opposition-based learning: a new scheme for machine intelligence. *Proceedings of CIMCA-IAWTIC*, vol. 1, pp. 695-701.

Vila, S.; Guirado, F.; Lerida, J. L.; Cores, F. (2019): Energy-saving scheduling on IaaS HPC cloud environments based on a multi-objective genetic algorithm. *Journal of Supercomputing*, vol. 75, pp. 1483-1495.

Wang, T.; Liu, Z.; Chen, Y.; Xu, Y.; Dai, X. (2014): Load balancing task scheduling based on genetic algorithm in cloud computing. *Proceedings of the IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pp. 146-152.

Yang, X. S.; Deb, S. (2009): Cuckoo search via Lévy flights. *IEEE World Congress on Nature and Biologically Inspired Computing*, pp. 210-214.

Yu, W.; Li, X.; Yang, H.; Huang, B. (2018): A multi-objective metaheuristics study on solving constrained relay node deployment problem in WSNS. Intelligent *Automation and Soft Computing*, vol. 24, no. 2, pp. 367-376.

Zhang, G.; Zuo, X. (2013): Deadline constrained task scheduling based on standard-PSO in a hybrid cloud. *Advances in Swarm Intelligence ICSI, Lecture Notes in Computer Science*, vol. 7928.

Zhao, G. (2014): Cost-aware scheduling algorithm based on PSO in cloud computing environment. *International Journal of Grid & Distributed Computing*, vol. 7, pp. 33-42.

Zuo, X.; Zhang, G.; Tan, W. (2014): Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud. *IEEE Transactions on Automation Science and Engineering*, vol. 11, pp. 564-573.