International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-8 Issue-4, November 2019

Hybrid Metaheuristic Based Offline Parallel Job Scheduling in Heterogeneous Computing Systems

Amit Chhabra, Gurvinder Singh, Karanjeet Singh Kahlon

Abstract*:Over two decades, Heterogeneous Computing Systems (HCS) are offering large amount of federated computing resources, spanning across different administrative domains, to compute-intensive user applications. Efficient job schedulers are required to allocate HCS resources to user applications to satisfy system provider and user requirements. Offline scheduling is most popular kind of job scheduling in heterogeneous system, in which jobs are collected in batch and scheduled together. Job scheduling in HCS has become NP-hard problem due to system scale, federated structure and high resource as well as job heterogeneity. Simple queuing and deterministic heuristics have failed to provide optimal solution to NP-hard job scheduling problem. Due to NP-hard nature of job scheduling problem, there is always a scope to propose new scheduling solutions using meta-heuristics. Offline scheduling in HCS has been focused more on scheduling independent sequential tasks viz. Bag-of-tasks or Many-tasks. Offline scheduling of parallel jobs (composed of collaborating tasks with no precedence) in HCS has not gained much attention. In this paper, a novel hybrid multi-objective meta-heuristic known as HCSPSO, which combines the qualities of Cuckoo search (CS) and Particle Swarm Optimization (PSO), has been proposed to schedule batch of parallel jobs in multi-cluster HCS platform. Proposed HCSPSO policy is extensively compared with different heuristics and metaheuristics using different resource configurations and real supercomputing workload logs. Comparative results have showed the dominance of the proposed hybrid scheduling algorithm over other algorithms.

***Keywords*:Heterogeneous Computing systems, Metaheuristics Offline Scheduling, Parallel Job..**

I. INTRODUCTION

Heterogeneous Computing Systems (HCS) provides a large scale computing platform encompassing multiple geographically distributed computing resources, from different administrative domains- ranging from campus-wide resources to worldwide federated resources, to execute complex applications [1]-[3]. This aggregation and sharing of multiple resources from distributed computing sites for executing large user applications is known as co-allocation or cross-site allocation[4][5]. Co-allocation improves the overall utilization of resources, removes internal fragmentation at computing sites and reduce the wait time of applications resulting into improved execution time[5][6].

The most common HCS platforms for High-performance Computing applications are Multi-cluster system (MCS) and, Computational Grids (CG) and Clouds. Multi-cluster system and Computational Grids both consists of multiple clusters connected with each other using communication link. Communication link in MCS is high-speed dedicated link as opposed to dynamic and non-dedicated one (usually internet) in CG. Number and heterogeneity of resources as well as the federated scale can be higher in Grid as compared to MCS [1][6]. In some of the computational grids, for example volunteering grids, resources may leave and arrive back, presenting a more dynamic infrastructure as compared to controlled Multi-cluster environment[5]. Cloud platform is established on the concept of utility computing and virtualization for offering different types of services such as Infrastructure as a service (IaaS), Platform as a service (PaaS) and Software as a service (SaaS) to users [7].

One of the main challenges in HCS platforms from last two decades is to develop efficient job scheduling algorithms in order to meet the expectations of the service provider and user [3]. Job scheduling problem (JSP) in HCS platforms primarily deals with two scheduling issues; first to decide the order of execution of waiting jobs (also known as job ordering) and second is allocation or mapping of jobs (also refereed as tasks or applications) to HCS resources [8]. In scheduling literature, job ordering policies and resource mapping or allocation techniques are sometimes collectively or individually called as Job scheduling policies.

A. Motivation

Job schedulers are the core components of any computing platform to efficiently exploit the availability and strengths of computing resources; therefore JSP issue has been dealt by using diverse approaches belonging to queuing algorithms, deterministic heuristic algorithms and meta-heuristic algorithms. However most of the research works in popular offline scheduling in HCS platforms has been focused on scheduling batch of independent sequential jobs also known as bag-of-tasks (BoTs) or Many-tasks [8]. In offline scheduling, also called as static scheduling, jobs are collected in a batch and scheduled together according to specified job ordering policy [9]. It is assumed that most of information regarding job characteristics such as job expected execution time, job arrival time, required resources is known before execution as opposed to the dynamic scheduling where most of job information is available at the run time.

Manuscript published on November 30, 2019.
* Correspondence Author
Amit Chhabra*, Department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar, India.
Gurvinder Singh, Department of Computer Science, Guru Nanak Dev University, Amritsar, India.
Karanjeet Singh Kahlon, Department of Computer Science, Guru Nanak Dev University, Amritsar, India.

Static job scheduling is preferred over dynamic scheduling due to their ease of implementation and small overhead involved. Moreover static batch job scheduling is the default approach used in most of job schedulers and resource management systems of clusters and grid computing systems. As discussed above, since most of the research on offline scheduling in HCS platforms is focused on batch of independent sequential tasks, therefore we have been motivated to work on less explored area of scheduling batch of parallel jobs onto heterogeneous computing resources. In this paper, a parallel job refers to a job consisting of fixed number of tasks where each task requires single processor for execution and tasks of a job may collaborate with each other during communication phase. Examples of such parallel jobs are high performance computing (HPC) application such as electromagnetic computing [10], plasma simulations [11], and combinatorial and mathematical studies [12, 13].

Secondly the JSP issue in HCS platforms is a popular NP-hard problem due to federated resources, problem size, high heterogeneity of computational resources as well as applications and dynamic nature of resources [14]. Therefore there is always an opportunity to develop new optimal solutions for scheduling jobs using metaheuristics.

Thirdly hybridization of metaheuristics [14][26][30] is often used by researchers to solve diverse problem areas including offline sequential task scheduling in HCS platforms. Hybrid meta-heuristic combine the best qualities of more than one metaheuristic to provide best optimal solution of the undertaken problem and found to produce better results than stand-alone metaheuristics. This inspires us to propose hybrid meta-heuristic based scheduling solution to execute batch of parallel jobs. With all these three motivations listed above, we tend to work for the development of hybrid metaheuristic for scheduling batch of parallel jobs in heterogeneous computing systems.

B. Contributions

The contributions of the present research paper are listed as following

1) Extensive literature survey regarding job scheduling problem solutions in HCS platforms has been presented.

2) A hybrid multi-objective HCSPSO algorithm that combines the advantages of CS and PSO metaheuristics has been proposed for scheduling batch of collaborative parallel jobs.

3) Objective function using weighted sum method approach is designed to optimize three conflicting objectives viz. Makespan, Energy Consumption and Average Flowtime simultaneously.

4) Benchmarking of proposed hybrid algorithm is done with two real supercomputing workload logs as opposed to synthetic workloads found in most scheduling research.

5) Proposed algorithm has been extensively compared with popular heuristics and metaheuristics with different workload and machine configurations.

The rest of the research paper is structured as follows. The second section presents the relevant existing related work on solving the problem of job scheduling in HCS platforms using meta-heuristic policies. The problem definition and system model used to evaluate the performance of the proposed methods has been described in third section. The proposed hybrid metaheuristic HCSPSO is outlined in fourth section. The overview on standard CS and PSO algorithms is also available in fourth section. Experimentation, workloads, results of algorithms and analysis of results are available in fifth section. Conclusions and future directions are summarized in the last section.

II. RELATED WORKS

In the past, extensive research has produced many simple queuing and heuristic algorithms such as FCFS, SJF, min-min etc. to deal with job scheduling problem in HCS platforms. Such heuristics produced good results for small problem and system sizes. However, to effectively deal with the NP-hard JSP problem in complex HCS platforms, several meta-heuristic based offline solutions are developed over the years [14-17][30]. The major focus of the research was on scheduling independent jobs i.e. either independent sequential job such as Bag-of-tasks or parallel job with independent tasks without inter-cluster communications.

A. Scheduling on Parallel jobs

For offline scheduling of parallel jobs with co-operating tasks on federated clusters systems, Authors in [15] has proposed a PSO based scheduling approach for minimization of energy consumption. The proposed PSO approach was used to deal with job ordering and mapping of resources to jobs. The PSO approach was found to be least computationally expensive approach than other metaheuristic technique. Authors in [16] proposed MOPSO-FGA approach for scheduling batch of parallel jobs of collaborative tasks in heterogeneous multi-cluster systems. During PSO iteration, a fuzzy genetic crossover operator was applied over PSO particles with the objective to minimize makespan and energy consumption simultaneously. MOPSO-FGA failed to produce best makespan and energy consumption when compared with MOGA [18] but it produced shorter algorithm run time than MOGA.

Authors in [2015-Switlaski] introduced the Generalized External Optimization (GEO) based scheduling policy for batch of parallel jobs consisting of independent tasks in computational grids to minimize the makespan. The proposed policy was compared with GA with the help of synthetic workload. GEO based policy produced better results over GA policy in all the test cases of three sets of jobs.

Authors in [18] proposed Multi-objective Genetic Algorithm (MOGA) based solution using NSGA-II for scheduling of batch of collaborative parallel jobs in federated cluster computing structures to minimize makespan and energy consumption simultaneously. MOGA was used to provide solution for job ordering and processor allocation to jobs. The performance of MOGA was found to be much better than the other explored scheduling approaches in the work.

Authors in [19] concluded that offline scheduling by treating a group of parallel jobs together for scheduling produced better results than allocating each job one by one from waiting queue.

Authors in [20] used Mixed-Integer programming (MIP) model, to propose a job execution ordering technique based to decide order of waiting jobs using sets of job packages. Proposed solution took longer time and not feasible for scheduling bigger problems in large scale systems.

Author developed the GEO based policy for batch of independent parallel jobs without any communication requirements for multiprocessor systems [21] and concluded that GEO-based schedulers can compete with GA-based schedulers any computing environment.

Some research works focused on studying the co-allocation benefits and reducing the unnecessary co-allocations for scheduling parallel applications. Authors in [5] explored the benefits of processor co-allocation in multi-cluster Grid systems by conducting experiments in a real and simulated multi-cluster grid environment. The performance of co-allocation was evaluated under various system load settings for various parallel applications ranging from computation-intensive to communication-intensive. Similar other co-allocation research studies [4][6] have also concluded that though co-allocation results in improved performance by reducing waiting times of queued jobs but unlimited co-allocation may decrease the obtained performance due to increase in the overall execution due to communication penalty. However in all of these research works, one parallel job is considered at a time for allocation of resources without considering the allocation impact on other queued or future arrival jobs.

B. Scheduling on Independent Sequential jobs

Here we have present few research works, out of hundreds of research papers,for scheduling independent sequential jobs in HCS platforms.

Author in [22] proposed GA based approach known as Pro-GA for offline scheduling of BoTs in multi-core heterogeneous computing systems for minimization of makespan, resource utilization and speedup ratio. Authors in [23] proposed multi-objective hybrid algorithm by combining CS and GA in pipelined manner for independent sequential task scheduling. The performance of CSGA was found to be better when compared with GA, CS and ACO with ETC model benchmarks.

Authors in [14] proposed two strongly coupled hybrid metaheuristics ACO-VNS and GA-VNS for offline sequential task scheduling to minimize makespan. The performance of hybrid algorithms was compared with many heuristics and metaheuristics. Authors in [24] proposed a novel genetic algorithm (Im-GA) based independent task scheduling technique to optimize four conflicting objectives, makespan, load balancing, resource utilization, and speed up ratio using a novel mutation technique.

Authors in [25] also proposed a GA based scheduling algorithm for scheduling Bag-of-tasks in computational grids using modified mutation strategy to improve the makespan. The proposed GA obtained best makespan over many sequential task scheduling methods, Authors in [26] proposed six multiobjective genetic algorithms using single and multiple population schemes for solving the scheduling problem of batch of independent tasks in Computational Grid (CG). Proposed methods minimized makespan, energy consumption and flow-time by considering different security constraints. Author in [27] has proposed a GA based scheduling solution for independent sequential jobs in the grid computing to minimize makespan time and job wait times.

For offline scheduling of independent sequential tasks on computational grids, Authors in [28] proposed a hybrid meta-heuristic algorithm known as GA-GELS to decrease overall runtime and missed task deadlines. On the pattern of GA-GELS, Authors in [29] proposed a hybrid metaheuristic known as PSO-GELS, to schedule batch of independent tasks over grid resources. Simulated experimental results demonstrated the effectiveness of PSO-GELS over others.

Authors in [30] also suggested the Bag-of-tasks scheduling in Computational grids (CG) as a multi-objective problem to minimize makespan and energy consumption.The Dynamic Voltage Scaling (DVS) methodology was used for frequency scaling to reduce cumulative system power energy. A fuzzy Particle Swarm Optimization (PSO) meta-heuristic algorithm was proposed for bag-of-tasks scheduling on computational grid by authors in [31] to minimize the makespan time. Associated performance was benchmarked by extensive simulations using different resource-job pair test cases. Their PSO method was found to produce faster and feasible solutions as compared to GA and SA approaches.

The authors in [32] has suggested a heuristic based on ACO algorithm to schedule jobs andload balancing among grid resources by governing the completion time of tasks. In [33], the authors introduced a fuzzy based PSO algorithm to optimize the finishing time of jobs and resource utilization. The authors in [34] has proposed a scheduling approach for Bag-of-tasks based on PSO approach to find solution for multi-objective grid JSP issue to reduce the overall completion time of jobs.

For solving the multi-objective offline sequential job scheduling problem, GA-based scheduler has been proposed in [35] in Grid computational systems. Moreover, various genetic operators have been implemented to improve the performance of GA based scheduler.

In one of the earliest and highly cited research work [36], eleven static job scheduling heuristics for independent jobs, has been proposed for heterogeneous distributed computing systems. The goal of the heuristics was to minimize the makespan i.e. total execution time. Simulated results demonstrated the dominance of proposed GA over the other ten algorithms explored in the work.

Most of the research works listed in this section focused on scheduling either independent jobs or parallel jobs consisting of independent tasks in HCS platforms without considering bandwidth penalties. We found very few research works [15-16], [18-20] which provide offline solution for scheduling collaborative parallel jobs by considering computation heterogeneity and inter-cluster communications. Such parallel job's tasks have computation and communication phases and tasks can collaborative with each other during communication phase. Therefore we have been motivated to work in finding scheduling schemes for executing collaborating parallel jobs and we will use best papers out of these research works as our baseline papers.

III. SYSTEM MODEL

A. Heterogeneous Computing System and Parallel Batch Jobs

The Heterogeneous Computing System model and offline scheduling problem are defined as follows (Blanco 2011). Heterogeneous computing system consists of multiple federated clusters connected by a dedicated communication link. Each unique cluster resource (*CR*) consists of number of homogenous processing elements (*PEs*).

However Cluster resources are different from each other in terms of relative computing power (calculated using relative million instructions per second (*MIPS*) rating) and energy consumption of associated processing elements. In other words, cluster resources are heterogeneous in terms of PEs computing power and energy consumption.

HCS is formally defined as:

$$HCS= \{CR_1(PE_i,PE_{i+1},...,PE_n), CR_2(PE_j,PE_{j+1},...,PE_n),...,CR_m(PE_k,PE_{k+1},...,PE_n)\}$$

Where $m$is the number of cluster resources and $n$is the number of processing elements (PEs) in each unique cluster resource known as *cluster size*.

In a multi-cluster HCS system, a set J of independent parallel jobs $J_1, J_2, ..., J_j$ waits to be scheduled over cluster resource. A job j is described by a quintuple $<T_{aj}, size_j, T_{bj}, sigma_j, PTBW_j>$. We consider non-preemptive offline scheduling; therefore, the arrival time, T_{aj} of each job is equal to zero. The $size_j$ known as degree of parallelism (*DOP*) refers to the fixed number of tasks a job contains or the fixed number of processing elements (*PEs*) required by job j for execution. In order to satisfy the job size, the PEs can be either obtained from singleton cluster resource or from co-allocated cluster resources across two or more clusters (in order to reduce wait time of job when no single cluster resource is able to satisfy job size j) so that $size_j \leq$ n must be satisfied, where n is total number of *PEs* in all cluster resources. Each parallel job is a kind of data-parallel job consisting of multiple computation and communication phases. All tasks of the job work independently during computation phase and may work in collaboration during communication phase by passing processed data among each other following BSP model[37]. T_{bj} is defined as the estimated execution time of job j when executed with slowest cluster resource in the system.

We assume space-sharing approach i.e. all processors assigned to job tasks could not be shared with any other job till the completion of current job and assigned processors will be released only after the completion of job (i.e. time when job complete its execution). Completion time of job in multi-cluster HCS platforms is bounded by the finish time of the job's task executed on slowest cluster resource. Moreover, a job cannot be reallocated to a different cluster resource.

B. Job Execution Time Model

We follow the execution time model of [38] which is also adapted by many related research works [16][18-19], to model the execution time slowdown due to heterogeneity of computational resources and communication contention present on shared communication link in multi-cluster HCS platform. For a parallel job j, Cumulative slowdown SD_j in the presence of processing slowdown (SP_j) and communication slowdown (SC_j), is calculated using (1). σ_j represents the relevance of processing time with respect to communication time. For more details on calculating SP and SC, readers can refer to [16][19][38].

$$SD_j = \sigma_j \times SP_j + (1 - \sigma_j) \times SC_j \quad , \forall j \in J \qquad (1)$$

The actual job execution time T_{ej} with processing and communication slowdown is calculated in (2).

$$T_{ej} = T_{bj} \times SD_j \qquad (2)$$

where T_{bj} is the base execution time of job j in dedicated HCS environment with SP and SC equals to one.

C. Fitness Function

The performance evaluation criteria of all the algorithms in this research paper are most commonly used metrics i.e. makespan, flowtime and energy consumption.

Makespan: The Makespan (MS) represents the total workload completion time, is the time duration measured from the arrival time of the first job submitted to the system, until the job that finishes last in the system. Makespan (MS) is expressed by (3).

$$MS = \max_{j \in J} (FinishTime_j) - \min_{i \in J} (arrivalTime_i) \qquad (3)$$

Average Flowtime: The flow time of a job is the time difference between the finishing time and the arrival time of the job. The finish time of a job comprises the wait time and execution time. The flowtime (FT) is defined in (4) as

$$FT = (FinishTime_j) - (arrivalTime_j) \ \forall j \in J \qquad (4)$$

Average Flowtime (AFT) is defined as the sum of finishing times of all jobs averaged over total number of jobs (N) in the batch J as shown in the (5). Average flowtime is an indicator of how fast an individual job completes in the system.

$$AFT = \frac{\sum_{j \in J} FT_j}{N} \qquad (5)$$

Energy consumption: The energy consumption by single processing element can be expressed by (6)

$$EC(PE_k) = \int_0^{MS} EC_{comp}(PE_k, t) + EC_{idle}(PE_k, t)\, dt \qquad (6)$$

where EC_{comp} and EC_{idle} are the energy consumption (in watt-hour units) during the computation and the idle period, respectively, by the processing element PE_k in the time period t .The total energy consumption (*TEC*) of all the PEs of all clusters of HCS can be calculated by (7) as follows:

$$TEC = \sum_{PE_k \in C} EC(PE_k) \quad \forall C \in Clusters \ in \ HCS \qquad (7)$$

In this paper, offline parallel job scheduling problem on federated HCS is formulated as a tri-objective optimization problem for minimization of makespan, total energy consumption and average flowtime simultaneously. The *fitness function* for the tri-objective scheduling problem is defined using weighted-sum-method by (8):

$$\textit{Fitness function, } F = w_1 \text{ x } MS + w_2 \text{ x } TEC + w_3 \text{ x } AFT \qquad (8)$$

where w1, w2 and w3 are the weights of Makespan (MS), Energy consumption (TEC) and Average Flowtime (AFT) objectives respectively. Sum of weights of all objectives is equal to 1.

Minimizing MS, TEC and AFT in *fitness function*F are conflicting objectives. An attempt to decrease any one objective, may lead to increase in the other objective.

IV. OFFLINE PARALLEL JOB SCHEDULING USING METAHEURSITC

In this section, first of all the standard CS and PSO algorithms which are to be used in the proposed hybrid

HCSPSO metaheuristic are discussed followed by explanation of detailed methodology used in HCSPSO method.

A. Standard Cuckoo Search Algorithm

Cuckoo search (CS) is a swarm-based multi-population algorithm by authors in [39], which has shown remarkably good performance on many optimization problems [40]. This algorithm is motivated by the exceptionalbehavior of a bird species called cuckoo. Cuckoo birds are known for not building their own nests and laying their eggs in the other host bird nests. In general, CS has strong capabilities for local as well as global search and required less number of parameters to optimize.

Algorithm1 Standard Cuckoo Search (CS) algorithm
1:**Input** the scheduling problem, fitness function F(S)
2:Initialize the parameters of scheduling problem and CS algorithm
3: Initial population generation of *n* host nestsS_i^d (i = 1, 2,..., n) ; (dimension d = 1, 2,…,m)
4: **while** (k < MaxGeneration)
5: Apply Lévy flightsmechanism on random cuckoo using (9) and evaluate its fitness F(S_i^d) using objective function
6: Choose a nest among n (say, j) randomly
7: **if** ($F(S_i^d) < F(S_j^d)$)
8: ExchangeS_j^d by the new S_i^dsolution;
9: **end if**
10: Abandon a fraction (p_a) of worse nests and build new nests
11: Keep the best nest solutions;
12: Calculate fitness of the solutions and find the current best solution
13: **end while**
14: **Output** the best solution found

Each individual S_i^d in cuckoo search is also known as cuckoo or nest. Initial population S of n individuals is generated specific to the problem undertaken. A cuckoo is randomly selected from population and moved to another position using Levy flight mechanism using equation.

$$S_i^d(k+1) = S_i^d(k) + \alpha_0(S_i^d(k) - gBest(k)) \oplus Levy(\lambda) \quad (9)$$

where α_0> 0 is the step size,which can be carefully selected according to the problem to be solved. The product ⊕indicates entry-wise multiplications. The Lévy flight provides a random walk, in which the step length is determined by Lévy distribution

$$Levy(k,\lambda) \sim k^{-\lambda}\ ,(1 < \lambda \leq 3)$$

Fitness of this new cuckoo S_i^d is compared with any randomly selected cuckoo S_j^d from population S. Solution S_i^d is replaced with S_i^d if fitness of S_i^d is better than S_j^d individual. A fixed fraction p_a of solutions from population is abandoned and replaced by random solutions to prevent falling into local optima during each generation (Yang and Deb 2009). The intention is to use the new and better solutions (cuckoos) to replace worse solutions present in the nests. The fraction p_a parameter helps to maintain balance between exploration and exploitation capabilities of CS. Solutions are ranked using fitness function and global best solution is obtained. The whole procedure is repeated till maximum generations or termination condition is reached.

B. Standard Particle Swarm optimization Algorithm

Particle swarm optimization (PSO) method was introduced by authors in [41] inspired from the bird flocking behavior. PSO is a well-known high level algorithm for finding the best global optimum solution. The concept of standard PSO is explained using pseudo-code as follows.

Algorithm 2 Standard PSO algorithm
1:**Input** the scheduling problem, objective function F
2:Initialize the parameters of scheduling problem and PSO
3:Initial population generation Sof *n*particles S_i^d (i = 1, 2,..., n) ; (dimension d = 1, 2,…,m)with random positions and velocities
4:Compute the fitness of each individual S_i^d in the population using objective function
5:Mark the particle with best fitness value (*gBest)*in population
6:**while** maximum iterations or termination criterion is not attained **do**
7: **for** each particleS_i^d**do**
8:Use (10) to update the particle's velocity
9:Correct the velocity boundaries if required
10: Update the particle's position using (12)
12:Compute the fitness of the particle
13: **if**F($S_i^d(k+1)$) <F($S_i^d(k)$)
14: Update the *pBest*.$S_i^d(k)$= $S_i^d(k+1)$
15: **end if**
16: **if**F($pBest.S_i^d(k)$) < $F(gBest)$
17: Update the *gBest*
18: **end if**
19: **end for**
20:**end while**
21:**Output** the best particle as the final solution.

Each individual S_i^d i.e. candidate solution for the undertaken problem in PSO is known as Particle which in turn consists of velocity and position. Particles are randomly initialized in the first step to build population (line 3). Fitness of the particles i.e. personal best positions (*pBest*) is calculated from given objective function and a particle with best overall fitness known as *gBest* is marked (line 4-5). For every particle in the population, particle's velocity and position are updated using based on its previous best position and global best position of the whole population as shown in equations.

$$V_i^d(k+1) = wV_i^d(k) + c_1r_1(pBest_i^d(k) - S_i^d(k)) + c_2r_2(gBest(k) - S_i^d(k)) \quad (10)$$

$$\omega(k+1) = \omega(k) * \alpha \quad (11)$$

where, c1 and c2 are self-recognition and social constant respectively and r1, r2 two random numbers generated uniformly between 0 and 1. ω is inertia weight and α is a decrementing factor randomly generated between 0 and 1.

$$S_i^d(k+1) = S_i^d(k) + V_i^d(k+1) \quad (12)$$

This whole process gives a chance to the particles to "fly" toward the globally optimal region in the search space. The whole procedure is repeated till the maximum generations or termination condition is achieved. Some of the advantages of PSO method are the ability to memorize the past good solutions, from where information passes on to future generations,

inherent cooperation between particles of the swarm due to their ability to work together to create solutions and faster convergence to achieve global solution.

Proposed HCSPSO metaheuristic:

In this section, we propose a hybrid algorithm HCSPSO for the parallel batch job scheduling problem by combining the qualities of both CS and PSO algorithms. During each generation, HCSPSO selects either CS or PSO based on random selection which will help to avoid falling into local minima by providing diversity over successive populations. Moreover there is an information exchange in terms of passing previous particle position, particle velocity and global best solution between CS and PSO algorithms which helped to achieve global optima. Merits of individual CS algorithm viz. strong local as well as global search abilities and PSO algorithm viz. faster convergence and global search ability are also retained in the HCSPSO.

The detailed descriptions of the hybrid algorithm in terms of solution representation and proposed hybrid algorithm pseudo code are discussed in the following section.

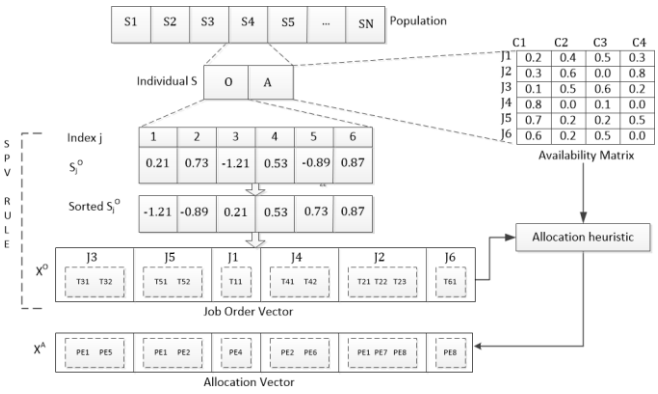

Fig.1. Solution Representation

C. Solution Representation

The first step for application of metaheuristic in scheduling problem is the solution representation. The most of the standard metaheuristics such as ACO, ABC, PSO and CS algorithms were originally developed to take care of continuous optimization problems with real number solutions, while the offline parallel job scheduling is a discrete problem involving discrete valued job numbers and processors. Thus, the standard metaheuristics e.g. CS and PSO cannot be directly used to solve the job scheduling problem. In this paper, the SPV rule [43], due to its simplicity, is applied to convert continuous values in job vector to discrete valued unique job order known as job permutation.

Since job scheduling problem consists of two sub-problems; Job execution order and resource allocation, therefore each individual S, also known as a candidate solution to the scheduling problem, is composed of job order (O) vector and resource availability (A) matrix components [18]. Let each index of a job order continuous solution S_i^O represent a unique job from set of *n* jobs in the multi-cluster system. Then, n indexes denote n different jobs. Assume that $S_i^O = \{x_1, x_2, ..., x_n\}$ is a real-valued solution of n jobs. By sorting the position values of S_i^O in non-decreasingorder, a job order X_i^Oknown as job permutation is obtained. Figure 1 presents an example of the SPV rule for generating discrete-valued job ordered vector from continuous job vector.

Second component (S_i^A)known as resource allocation component of individual solution *S*, is initially represented using Availability Matrix (AM) which records the fixed percentage of cluster PEs available to each job from each cluster. This resource availability representation scheme is adapted from availability mechanism [42] and blacklist matrix allocation scheme [18]proposed by authors of one of our base papers. Availability matrix (AM) contains (job, cluster) pair elements as real numbersin the range [0, 1]. This value ensures that only the fixed percentage of cluster PEs are available for the allocation to the corresponding job and reserves rest of the cluster PEs for other jobs in the queue.

The actual resource allocation for mapping of PEs of clusters to the waiting jobs in discrete-valued job execution vector X_i^O, is done by applying resource allocation heuristic shown in Algorithm 3. This resource allocation heuristic will convert the resource allocation from availability matrix (real values) to the allocation vector integer-valued processing elements.

The detailed pseudo code for the heuristic for resource allocation is shown in Algorithm 3 below.

Algorithm 3: Resource Allocation Heuristic

INPUT: Set of Clusters (C), set of jobs (J) obtained from job execution order vector X_i^O

Require: AM: Cluster resource availability represented by the AM matrix

Output: Allocation list (A), pair of (*job*∈J, *cluster* ∈C)

1. **for** *job* ∈J**do**
2. Declare sets: *avail_clusters, avail_clusterPEs, co-allocatedJobs and non-coallocatedJobs*
3. **for** *cluster*∈C**do**
4. *avail_clusterPEs= clusterTotalPEs* ∗ AM(*job, cluster*)
5. **if** *avail_clusterPEs*≥*jobSize***then**
6. *avail_clusters_list1*← *avail_cluster_list1*∪ *cluster*
7. *mark the job as "non-coallocated" and update non-coallocatedJobs*
8. **else**
9. *avail_clusters_lsit2* ← *avail_clusters_list1*∪*avail_clusters_list2*∪ *cluster*
10. *mark the job as "co-allocated" and update co-allocatedJobs*
11. **end if**
12 **end for**
13. *avail_clusters_list1* = *ascendingSort*(*avail_clusters_list1, MIPS*)
14. *avail_clusters_list2* = *ascendingSort*(*avail_clusters_list2, MIPS*)
15. **if** (*job*∈*non-coallocatedJobs*)
16. A← A∪(*job, allocate* (*firstClusterPEs, avail_clusters_list1*))
17. **else**
18. A← A∪(*job, allocate* (*Co-allocatedClusterPEs, avail_clusters_list2*))
19. **end if**
20. **end for**

First of all, the resource allocation heuristic updates the availability of PEs of a given cluster for each job by multiplying the corresponding AM (job, cluster) pair value with total PEs of the cluster in AM pair (Line 4).

In the next step, allocation heuristic categorizes each job into non-coallocated or co-allocated based on the whether the updated available PEs of a single cluster has enough PEs to satisfy the job size requirements or PEs from across the clusters are required to co-allocated to execute the job (Line 5-12). The above categorization forms two cluster lists according to the availability of PEs; first list,*avail_cluster_list1*, contains only those clusters which have enough PEs to single handedly execute the job and *avail_cluster_list2*contains all the HCS clusters. Both cluster lists are arranged in increasing order of their computation power using MIPS rating (Line 13-14). If a job is non-coallocated one, then PEs from first slowest cluster in the sorted available cluster list will be allocated to the job otherwise PEs across the multiple clusters starting from slowest cluster can be allocated to the job (Line 15-19).

In nutshell, Resource allocation heuristic tends to avoid unnecessary co-allocations to reduce communication slowdown to reduced actual execution time, by allocating single cluster PEs and tries to allocate low computation power resources to jobs to further reduce energy consumption.

Whenever the real-valued job order vector and availability matrix are modified due to metaheuristic operations, SPV rule and allocation heuristic are used to provide the discrete valued job execution order vector and resource allocation to the jobs.

D. Proposed HCSPSO algorithm

Detailed working of proposed HCSPSO approach is explained with the help of pseudo code in Algorithm 4.

Step 1: The first step in HCSPSO algorithm is the individual solution representation as shown in figure. Population S of n real-valued individuals (for both job order and allocation dimension) is generated. Discrete value population X is obtained from continuous valued population S by applying SPV rule and allocation heuristics. Parameters of scheduling problem, CS and PSO algorithms are also initialized (Line 1-3).

Step 2: Fitness of Population X (i.e. all the individuals) using objective function is calculated using (8). Personal best (pBest) and overall gBest of population X is calculated (Line 4-7).

Step 3: Then within the while loop (at Line 8-9), the random selection between CS and PSO method is made with the help of random numbers, and accordingly either CS or PSO method is selected and applied over real-valued population S. This random selection helps to avoid CS and PSO algorithm in HCSPSO to trap into local minima by providing diversity in population of solutions over the generations. Diversified population is possible due to passing the previously generated population by one metaheuristic equally likely to another metaheuristic in next generation. In this way, population of solutions produced by one metaheuristic will act as seed for other metaheuristic over the generations.

Step 4: Suppose if CS is applied first using random selection at generation k+1, then all the individuals (for both real-valued job vector and availability matrix dimensions) in population S are updated using (9). CS algorithm in (9) utilizes the previous *gBest* information found using our approach *gBest* selection method shown in Algorithm 5 to lead towards global solution. *gBest* selection method ensures that best previous recorded *gBest* solution should be made available to metaheuristic algorithms of HCSPSO over successive generations.

Originally there is no velocity updating scheme available in cuckoo search algorithm. We developed method for updating velocity as shown in (13) for the CS phase so that updated velocities of particles be made available to the PSO phase in the next generations for updating positions of particles. This updated velocity will help the individuals in PSO phase to fly towards global best solution. Worst performing individuals according to fixed fraction are discarded and new individuals are added to population (Line 11-19).

$$V_i^d(k+1) = \begin{cases} S_i^d(k+1) - gBest(k), & F(S_i^d(k+1)) < gBest(k) \\ gBest(k) - S_i^d(k+1), & F(S_i^d(k+1)) > gBest(k) \end{cases} \quad (13)$$

Step 5: Suppose next phase is PSO phase according to random selection method, then previous gBest, positions and velocities of particles are supplied by CS phase to PSO phase to update positions and velocities at current generation using equations. Overall *gBest*, which is minimum *pBest* among all particles, is calculated. In this way positions and velocities of all individuals and overall *gBest* of one generation is memorized and passed to the next generation till maximum generations are reached. Scheduling solution improves over generations due to exchange of information between successive generations and increase in diversity of solutions over generations.

Table-I: HCS Resource Configuration

Algorithm4: Hybrid Cuckoo Search Particle Swarm Algorithm (HCSPSO)

1:**Input** the scheduling problem, objective function F(X)
2:Initialize the parameters of scheduling problem, CS algorithm and PSO algorithm
3: Generate discrete-valued initial population X of n individuals i.e. X_i^d (i = 1, 2,..., n) ;
(dimension d = 2, i.e. for job order (O) and allocation (A) decision variables) from real-valued random population S
4: **for** each discrete valued individual $X_i^{O+A} \in$ X**do**
5: Evaluate the fitness of X_i^{O+A} using objective function defined in (8)
6: Calculate *pBest* of all individuals and obtain overall population gBest
7: **end for**
8: **while** (k < MaxGeneration)
9: **if** ($r_1 < r_2$) // Random numbers r_1, $r_2 \in$ U (0, 1)
10: //Apply Cuckoo Search algorithm on population using lines 11-19
11: **for**each real-valued individual $S_i^{O+A} \in$ S**do**
12: Select previous *gBest* individual using*gBest* selection methodin Algorithm 5
13: Update cuckoo individual using levy flights using (9)
14: Update Velocity of cuckoo individuals using (13)
15: **end for**
16: Obtain integer-valued population X from population S using SPV rule and allocation heuristic
17: Discard a fraction of worst individuals from population X
18: Reproduce new random individuals in S , Update population S and X
19: Rank the individual candidate solutions (*pBest*) and find the current global best solution (*gBest*)
20:**else**
21: //Apply PSO algorithm on population using lines 22-30
22: **for** each particle $S_i^{O+A} \in$ S**do**
23:Select previous *gBest* individual using *gBest* selection method
24:Update the particle's velocity and position using(10) and (12)
25:Check and correct the velocity boundaries for each component of velocity-vector
26:Update the particle's position using (10)
27: **end for**
28: Obtain population X from population S using SPV rule and allocation heuristic
29:Update*pBest* and *gBest* solutions
30: **end if**
31: **end while**
32: **Output** the best solution found

Algorithm 5:*gBest* Selection Method

INPUT: Previous gBest of CS and PSO phase, Current Generation = k+1
OUTPUT:*gBest(k)* to be used for updating CS and PSO individuals at next k+1 generation
1. *gBestPSOtemp* = *gBest* of previous PSO phase
2. *gBestCStemp* = *gBest* of previous CS phase
3. **if** (PSO phase comes after PSO phase || CS phase comes after CS phase)
4. *gBest(k) = gBest(k-1)*
5.**else if** (PSO phase comes after CS phase || CS phase comes after PSO phase)
6. **if***fitness*(*gBestPSOtemp*) <*fitness*(*gBestCStemp*)
7. *gBest(k) = gBestPSOtemp*
8. **else**
9. *gBest(k) = gBestCStemp*
10. **end if**
11.**end if**
12.**end if**

V. EXPERIMENTATION AND RESULTS

In this section, we have conducted number of experiments to analyze the effectiveness of the proposed hybrid HCSPSO scheduling technique. The experimentation using GridSim simulator was based on two different workloads extracted from traces of real supercomputing sites described by Feitelson in [44]. The selected workloads are: METACENTRUM (originally consisting of 5,731,100 parallel jobs collected during Jan 2013 to Apr 2015 at MetaCentrum national grid of the Czech Republic) and LLNL-THUNDER (originally consisting of 128,662 parallel jobs collected at Thunder cluster of the Lawrence Livermore National Lab (LLNL) during Feb 2007 to Jun 2007). In our experimentation, we have used the fraction of these workloads by using sets of 200, 400, 600, 800 and 1000 jobs of each workload to reduce the compute times of the simulations. Moreover, workload cleaning is also done to remove some of the unwanted or invalid entries.

	Resource configuration			Speed Heterogeneity & Energy Consumption								
	RC1	RC2	RC3	SH1(Low) & EC1			SH2(Medium) & EC2			SH3(High) & EC3		
Cluster	#PEs	#PEs	#PEs	MIPS	EC(comp) W/h	EC(idle) W/h	MIPS	EC(comp) W/h	EC(Idle) W/h	MIPS	EC(comp) W/h	EC(Idle) W/h
CR1	64	48	32	1000	300	130	1000	300	130	1000	300	130
CR2	64	48	32	1200	320	150	1500	415	195	2000	520	250

CR3	64	48	32	1300	400	250	2000	520	325	3000	650	400
CR4	64	48	32	1800	400	280	3000	520	365	4000	650	450

A. Baseline policies for comparison

The performance of the proposed HCSPSO algorithm is compared with baselineheuristics First-fit [4], Energy-aware variant of min-min (EAMM) [45], Energy-aware Minimum Execution Time Loss (METLE) [19] and Best-Fit [4] and baseline metaheuristics namely MOPSO-FGA [16] and MOGA [18] in federated cluster system. First-Fit uses FCFS as job ordering policy and look for the first cluster with enough PEs to fit the job. METLE firstly select the job which results into minimum execution time loss by taking into account the resource heterogeneity and system communication characteristics and in next phase allocate the job to the processor(s) that result in minimum energy consumption. Best-Fit always selects the cluster for allocation which results into least left-out free processors after allocation of job. Explanation about MOGA and MOPSO-FGA policies is already available in related work section.

B. Comparative Results and Analysis

HCS platform resource configuration consisting of four clusters is given in Table-I. The proposed HCSPSO and other baseline policies are extensively compared with each other by conducting number of experiments by varying multi-cluster resource configuration, speed heterogeneity of cluster resources and different workload sizes of LLNL and METACENTRUM workload logs. Each experiment for metaheuristic algorithms is repeated 10 times for same input and mean value is selected for comparison to remove randomness from collected results. The various parameters for metaheuristics are shown in Table-II. The best values for metaheuristic parameters are obtained after running various preliminary experiments.

Effect of workload size:

To see the effect of increase in the workload size on Makespan, Energy Consumption and Average Flowtime, the workload size is varied from 200-1000 jobs for both LLNL and METACENTRUM workloads. Workload size refers to the number of jobs in a workload set. The resource configuration RC1, speed heterogeneity SH1 and EC1 energy consumption values are fixed. From Fig. 2, Fig. 3 and Fig. 4; it can be observed that with the increase in the number of jobs in workload of LLNL-THUNDER; makespan, energy consumption and average flowtime of all policies increased. There is no sharp increase in makespan from workload size 200 to 800as shown in Fig. 2. This is due to low execution time and resource requirements of LLNL-THUNDER workload; multi-cluster system always has enough resources to complete the workload in minimized makespan. However at workload 1000, makespan rises due to increased execution times and resource requirements of jobs. Similar trend is observed with total energy consumption and average flowtime values as shown in Fig. 3 and Fig. 4 respectively for LLNL-THUNDER workload.

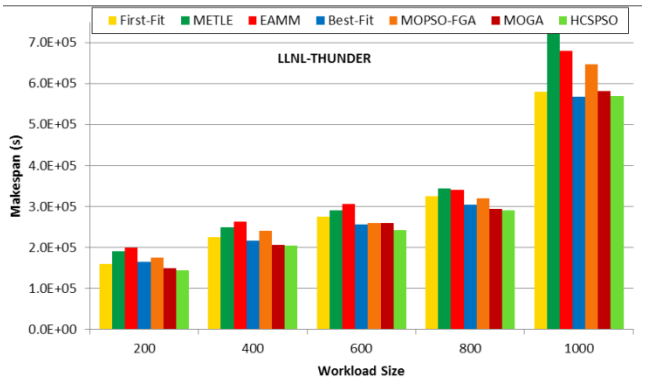

Fig.2.Effect of workload size on Makespan for LLNL-THUNDER

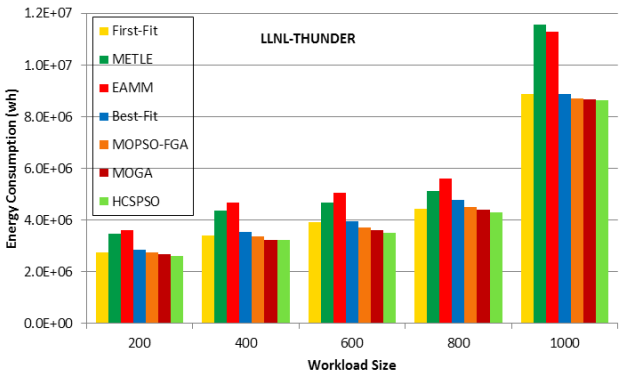

Fig.3: Effect of workload size on Energy Consumption for LLNL-THUNDER

However HCSPSO gives the minimized makespan, energy consumption and average flowtime in almost all workload sizes followed by MOGA policy.

Fig. 5, 6 and 7 show the effect of workload size scaling on makespan, energy consumption and average flowtime respectively for METACENTRUM workload. Trends of performance measures makespan, energy consumption and average flowtime are same as observed in LLNL-THUNDER workload. However overall makespan, energy consumption and average flowtime in METACENTRUM workload is higher than LLNL-THUNDER workload due to high execution time requirements of jobs in METACENTRUM. Again, HCSPSO has managed to achieve reduced performance values as compared other policies due to increased diversity leading to global best solutions.

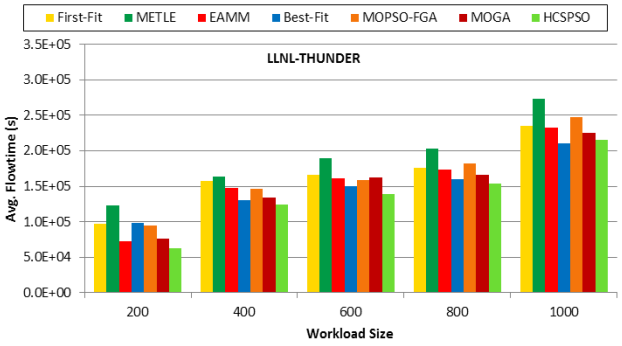

Fig.4. Effect of workload size on Avg. Flowtime for LLNL-THUNDER

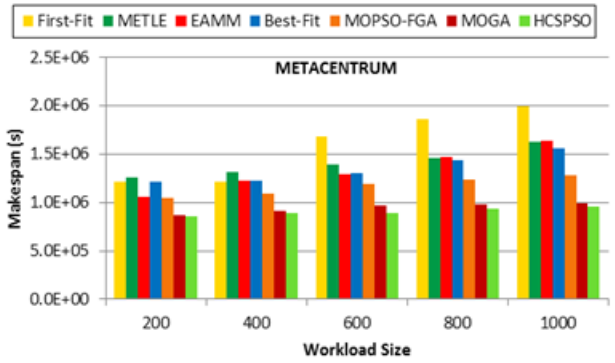

Fig.5. Effect of workload size on Makespan for METACENTRUM

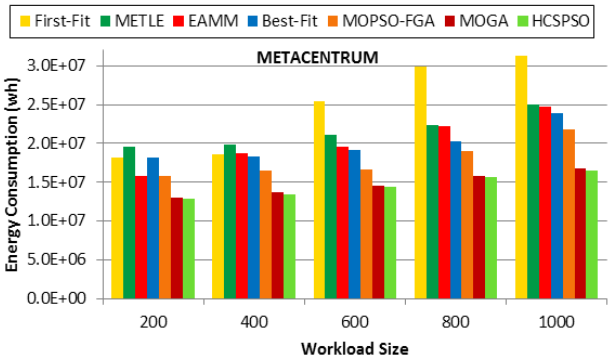

Fig. 6. Effect of workload size on Energy Consumption for METACENTRUM

Effect of Resource configuration variation:

To see the impact of decrease in the number of PEs in multi-cluster systems, three different multi-cluster resource configurations RC1(64,64,64,64), RC2(48,48,48,48) and RC3(32,32,32,32), as shown in Table I, have been selected. Workload size is fixed to 200 jobs and Speed heterogeneity is SH1 and corresponding energy consumption values are EC1. These resource configurations are different from each other in terms of the number of processing elements each cluster contains.

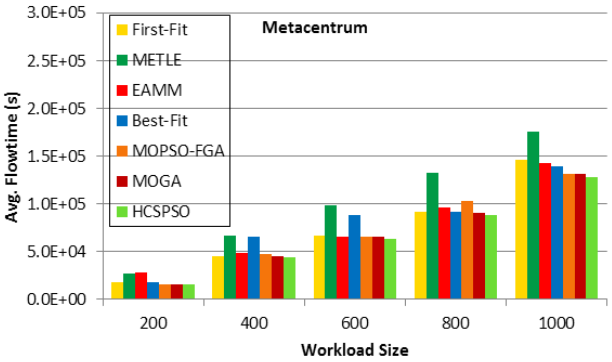

Fig. 7: Effect of workload size on Avg. Flowtime for METACENTRUM

Fig. 8, 9 and 10 show the effect of variation in resource configurations on Makespan, Energy Consumption and Average Flowtime respectively for LLNL-THUNDER workload. As the number of processing elements decreases, overall completion time increases due to decreased availability of resources as shown in Fig. 8.

Table-II: Metaheuristic Parameters

Parameter	Values
MOGA	
Population Size	60
Generations	80
Crossover rate	0.33
Mutation rate	0.10
MOPSO-FGA	
Population Size	40
Generations	50
Crossover rate	0.33
Mutation rate	0.10
HCSPSO	
Population Size	60
Generations	60
α_0 (CS parameter)	1.0
p_a (CS parameter)	0.25
ω (PSO parameter)	0.9
r1, r2 (PSO parameter)	0.65, 0.35
c1, c2 (PSO parameter)	2.0, 2.0

Fig. 9 shows that there has been decrease in the total energy consumption as number of processors in resource configurations is decreased. This is due to the fact that total idle energy, which contributes to nearly 30-40 % of total energy, is reduced as we move from RC1 to RC3. Average flowtime which represents the individual job flowtime is increased due to possible increase in the wait times of jobs as less number of free processors are available for allocation as shown in Fig. 10. Overall proposed HCSPSO policy achieves better results followed by MOGA policy in most of the test cases.

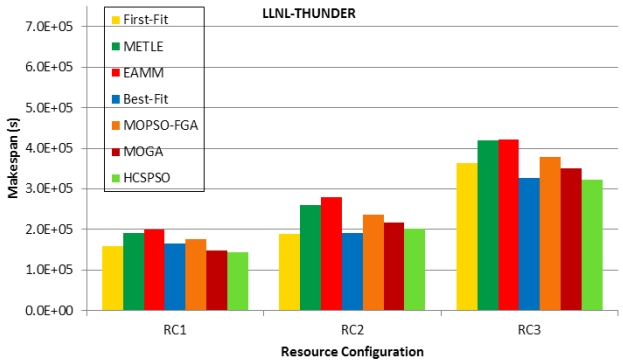

Fig. 8. Effect of Resource Configuration on Makespan for LLNL-THUNDER

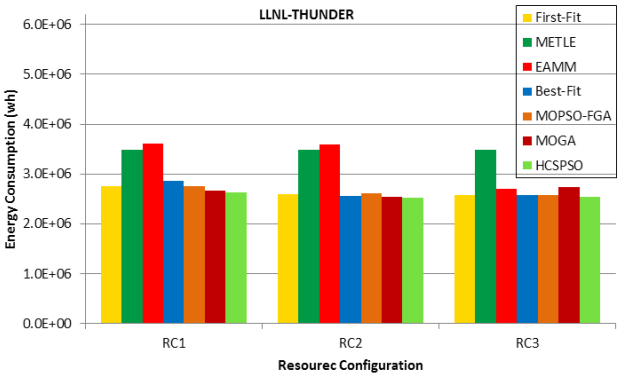

Fig. 9. Effect of resource configuration on Energy Consumption for LLNL-THUNDER

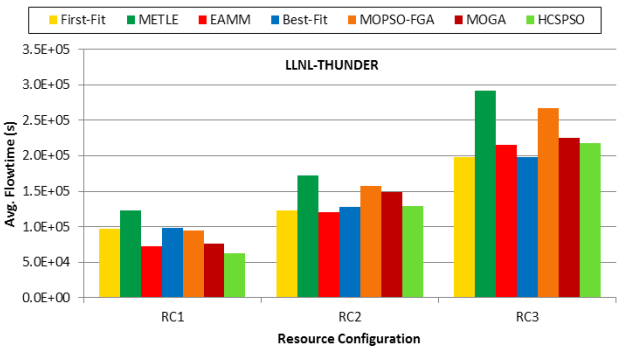

Fig. 10. Effect of resource configuration on Avg. Flowtime for LLNL-THUNDER

For METACENTRUM workload, similar trends have been observed for makespan, total energy consumption and average flowtime metrics when number of processing elements have been reduced by varying resource configurations as shown in Fig. 11, 12 and 13 respectively. HCSPSO produced better results at every resource configuration. There is little increase in makespan as number of processing elements have been reduced as shown in Fig. 11. This is due to the fact that METACENTRUM workload has low processor requirements. Energy Consumption and Average flowtime trends are same as recorded in LLNL-THUNDER workload.

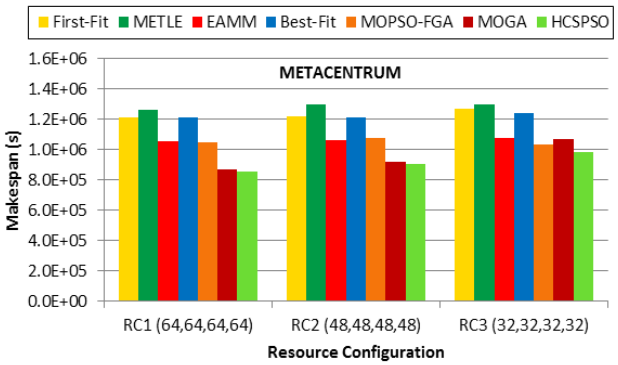

Fig. 11. Effect of Resource Configuration on Makespan for METACENTRUM

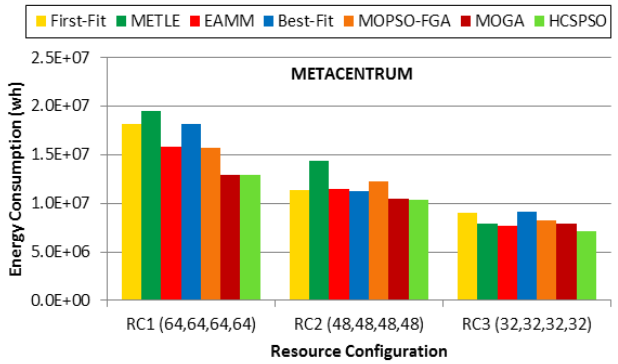

Fig. 12. Effect of Resource Configuration on Energy Consumption for METACENTRUM

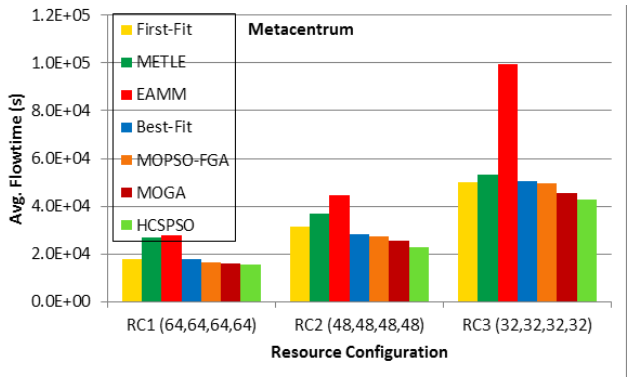

Fig. 13. Effect of Resource Configuration on Avg. Flowtime for METACENTRUM

Effect of speed heterogeneity:

To the see the effect of heterogeneity in computation speed of cluster resources in the multi-cluster system, three speed heterogeneity configurations viz. SH1, SH2 and SH3 are generated by changing the computation speed i.e. MIPS rating of cluster resources as shown in table. Speed heterogeneity is low in SH1, medium in SH2 and high in SH3 configuration. Associated energy consumption values for three speed heterogeneity configurations have been increased as shown in EC1,EC2 and EC3 respectively. Workload size is fixed to 200 jobs and resource configuration is fixed as RC1. Fig. 14 shows that as the computation speed of cluster resources increased, the makespan of all policies decreased due to reduced execution times of jobs. Although all the policies got benefitted due to increased resource computation speed, metaheuristic algorithms viz.

MOPSO-FGA, MOGA and HCSPSO have taken more benefit of resource heterogeneity because of their ability to exploit heterogeneous resources. Fig. 15 shows that although the makespan is reduced by increasing computation speeds, but energy consumption is little increased due to associated increased energy consumption values of PEs belonging to different speed heterogeneity configurations. Fig.16 indicates that with the increase in computation speed of resources, average flowtime decreases due to reduced job execution times.

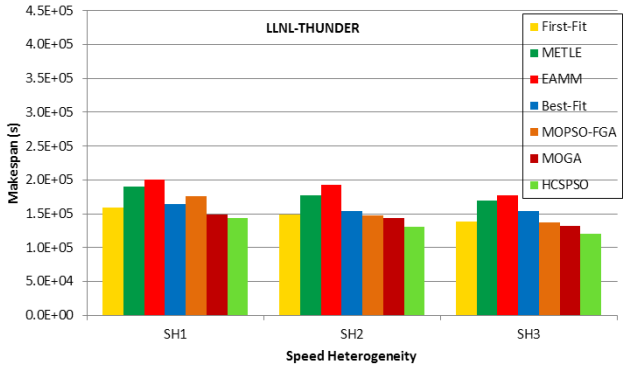

Fig. 14. Effect of Speed Heterogeneity on Makespan for LLNL-THUNDER

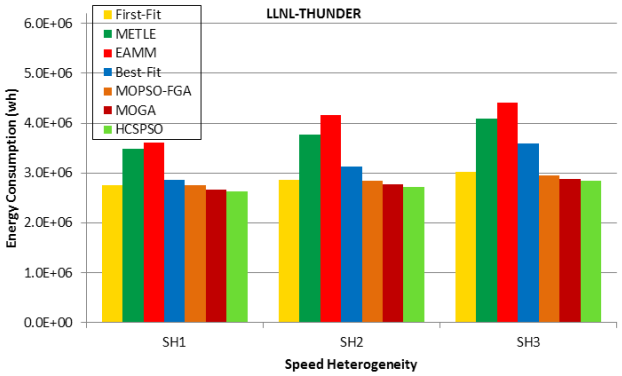

Fig. 15. Effect of Speed Heterogeneity on Energy Consumption for LLNL-THUNDER

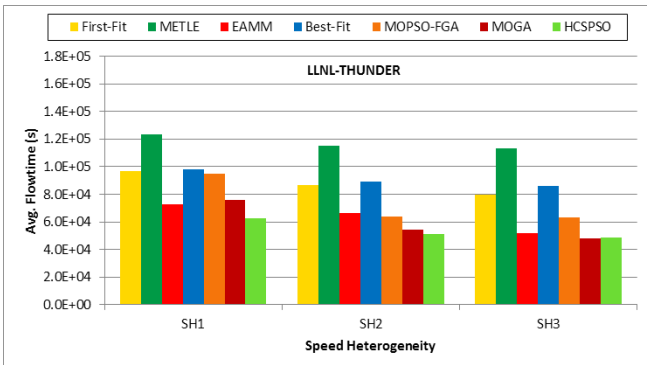

Fig. 16.Effect of Speed Heterogeneity on Avg. Flowtime for LLNL-THUNDER

Similar trends of effect of increased speed heterogeneity on makespan, energy consumption and average flowtime metrics have been observed in case of METACENTRUM workload as shown in Fig. 17, 18 and 19. The improvement in makespan, energy consumption and average flowtime is evidently more in case of metaheuristic techniques HCSPSO, MOGA and MOPSO-FGA as compared to First-Fit, METLE, EAMM and Best-Fit heuristics when speed heterogeneity is increased. HCSPSO showed the best performance over other algorithms due to increase in population diversity and hybridization of qualities of cuckoo search and particle swarm optimization algorithms.

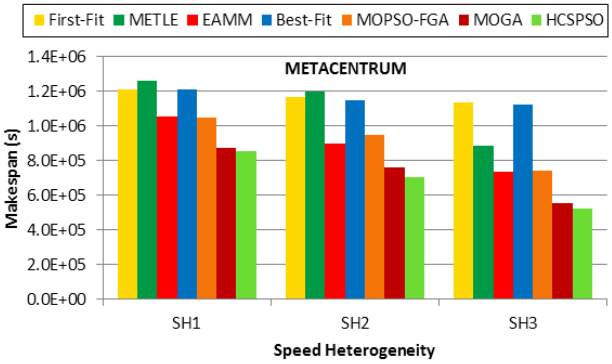

Fig. 17. Effect of Speed Heterogeneity on Makespan for METACENTRUM

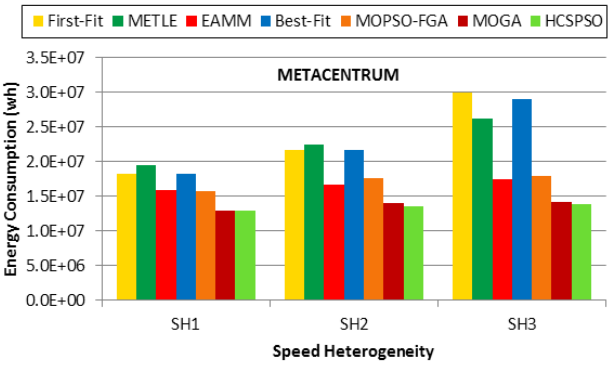

Fig. 18. Effect of Speed Heterogeneity on Energy Consumption for METACENTRUM

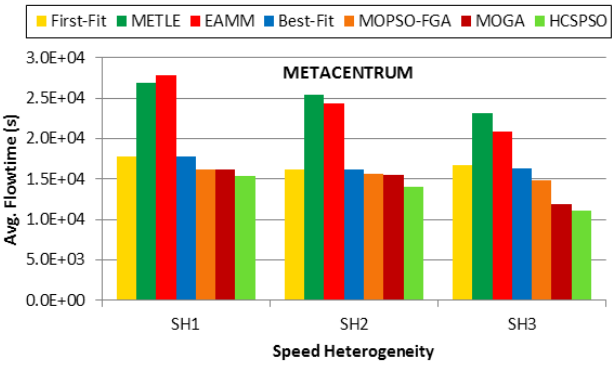

Fig. 19. Effect of Speed Heterogeneity on Avg. Flowtime for METACENTRUM

C. Summary of overall experimentation:

Table-III shows the median values of all the experimental results. Texts in bold show the values obtained from the best policy, and the underlined texts show the second best policy. % values (with + sign) calculated using (14) show the percentage increase in makespan, energy consumption and performance of other policies as compared to HCSPSO.

$$Percentage(\%)\ Increase = ((Algorithm\ Y - HCSPSO)/Algorithm\ Y)\ X\ 100 \quad (14)$$

In case of LLNL-THUNDER workload, First-Fit, METLE, EAMM, Best-Fit, MOPSO-FGA and MOGA respectively produced 11.06 %, 19.6%, 23.57%, 6.94%, 10.47% and

Table-III: Overall experimental results

	First-Fit	METLE	EAMM	Best-Fit	MOPSO-FGA	MOGA	HCSPSO
LLNL-THUNDER							
Makespan (s)	2.26E+05 (+11.06%)	2.50E+05 (+19.60%)	2.63E+05 (+23.57%)	2.16E+05 (+6.94%)	2.25E+05 (+10.47%)	2.09E+05 (+3.83%)	**2.01E+05**
Energy Consumption (W)	3.02E+06 (+6.29%)	4.10E+06 (+30.98%)	4.41E+06 (+35.83%)	3.56E+06 (+20.51%)	2.98E+06 (+5.03%)	2.92E+06 (+2.92%)	**2.83E+06**
Avg. Flowtime (s)	1.57E+05 (+17.83%)	1.72E+05 (+25.00%)	1.47E+05 (+12.24%)	1.30E+05 (+0.77%)	1.54E+05 (+16.23%)	1.47E+05 (+12.24%)	**1.29E+05**
METACENTRUM							
Makespan (s)	1.27E+06 (+32.32%)	1.30E+06 (+33.88%)	1.08E+06 (+20.42%)	1.24E+06 (+30.69%)	1.01E+06 (+14.90%)	9.51E+05 (+9.62%)	**8.60E+05**
Energy Consumption (W)	2.18E+07 (+40.83%)	2.06E+07 (+37.38%)	1.75E+07 (+26.29%)	1.87E+07 (+31.02%)	1.57E+07 (+17.83%)	**1.29E+07** (+0.00%)	**1.29E+07**
Avg. Flowtime (s)	4.48E+04 (+6.58%)	5.31E+04 (+21.19%)	4.84E+04 (+13.53%)	5.03E+04 (+16.80%)	4.64E+04 (+9.71%)	4.34E+04 (+3.57%)	**4.19E+04**

3.83% more makespan as compared to HCSPSO. Similarly these policies (in the same order) consumed more energy as indicated by increase of 6.29% , 30.98%, 35.83%, 20.51%, 5.03% and 2.92% respectively when compared to HCSPSO. These policies also resulted into 17.53 %, 25%, 12.24%, 0.77%, 16.23% and 12.24% more average flowtime than HCSPSO algorithm for LLNL-THUNDER workload.

In case of METACENTRUM workload; First-Fit, METLE, EAMM, Best-Fit, MOPSO-FGA and MOGA respectively produced 32.32%, 33.88%, 20.42%, 30.69%, 14.9% and 9.62% more makespan than HCSPSO. These policies in the same order produced 40.83%, 37.38%, 26.29%, 31.02%, 17.83% and 0% more energy consumption than HCSPSO. Increase of 6.58%, 21.19%, 13.53%, 16.8%, 9.71% and 3.57% was found in average flowtime in these policies respectively as compared to HCSPSO.

VI. CONCLUSIONS

Modern heterogeneous computing systems are continuously expanding to accommodate the ever increasing resource demand of users. Efficient schedulers are required to allocate the resources to HPC applications in order to meet the expectations of both user and system provider. However, the resource as well as application heterogeneity and communication contention becomes a bottleneck for effective HPC in HCS platforms. In this paper a hybrid multi-objective HCSPSO algorithm is proposed for scheduling data-parallel jobs in a multi-cluster HCS platform. HCSPSO effectively combines the qualities of both CS and PSO algorithms to generate overall optimal global solutions by providing diversity in population of scheduling solutions over the successive generations. Moreover the information exchange also took place between CS and PSO over the generations in the form of previous personal best solutions and overall global best solution. The experimentation work was conducted with two real supercomputing workload logs. Workload size, resource heterogeneity and speed heterogeneity configurations have been varied to obtain different results. The obtained experimental results have proved the ability of our proposed HCSPSO policy to dominate the results of several methods from the literature by obtaining lower overall completion times, minimum energy consumption and reduced average flowtimes. HCSPSO also offers scalability when tested with different number of jobs and number of resources. In the future, authors will compare proposed HCSPSO with other latest evolutionary metaheuristics with different supercomputing workloads.

REFERENCES

1. I. Foster, C. Kesselman,"The history of the grid computing",in Cloud Computing and Big Data, IOS Press, Amsterdam, 2013.
2. I. Foster, C. Kesselman., Tuecke, S. ,"The anatomy of the grid: Enabling scalable virtual organizations", International journal of high performance computing applications 15(3), pp. 200–222, 2001.
3. M. Akbari, H. Rashidi, S. H. Alizadeh, "An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems", Engineering Applications in Artificial Intelligence, 61, pp. 35–46, 2017.
4. A. I. D. Bucur, D.H.J Epema DHJ , "Scheduling policies for processor coallocation in multicluster systems", IEEE TPDS 18(7), pp. 958–972, 2007.
5. O. Sonmez, B. Grundeken,H. Mohamed, I. Alexandru, D. Epema, " Scheduling Strategies for Cycle Scavenging in Multicluster Grid Systems", 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID 2009. pp. 12-19.
6. O. Sonmez, H. Mohamed, D. Epema, D.(2010) , "On the Benefit of Processor Coallocation in Multicluster Grid Systems", IEEE Transactions onParallel and Distributed Systems, 21, pp. 778-789, 2010.
7. M. Netto, R. Calheiros, E. Rodrigues, R. Cunha, R. Buyya, "HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges", ACM Computing Surveys. 51 (1): 8:1–8:29, 2018.
8. G. Stavrinides, H. Karatza, "Scheduling Real-Time Jobs in Distributed Systems - Simulation and Performance Analysis", Conference: 1st International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2014)At: Porto, Portugal, 2014.
9. G. Stavrinides, H. Karatza, "Simulation-Based Performance Evaluation of an Energy-Aware Heuristic for the Scheduling of HPC Applications in Large-Scale Distributed Systems", Conference: the 8th ACM/SPEC, pp. 49-54, 2017.
10. P.B. Monk, A.K. Parrott, P.J. Wesson , "A parallel finite element method for electromagnetic scattering", COMPEL, Supp.A (13): pp. 237-242, 1994.
11. M. Nibhanupudi, C. Norton, B. Szymanski , " Plasma simulation on networks of workstations using the bulk synchronous parallel model", In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, 1995.
12. F. A. Salem, " A BSP Parallel Model for the Gottfert Algorithm over F2", Parallel Processing and Applied Mathematics ,pp. 217-224, 2004
13. P. Krusche, A. Tiskin , "Efficient Longest Common Subsequence Computation Using Bulk-Synchronous Parallelism", ICCSA ,3984, pp. 165-174, 2006.
14. M. Younis, S. Yang, "Hybrid Meta-Heuristic Algorithms for Independent Job Scheduling in Grid Computing", Applied Soft Computing. 72, 2018.
15. E. Gabaldon, F. Guirado, J. Lerida, J. Planes, " Particle Swarm Optimization Scheduling for Energy Saving in Cluster Computing Heterogeneous Environments",pp. 321-325, 2016.

16. E. Gabaldon, S.V. Almenara, F. Guirado, J. Lerida, J. Planes, " Energy efficient scheduling on heterogeneous federated clusters using a fuzzy multi-objective meta-heuristic",pp. 1-6, 2017.
17. P. Switalski, F. Seredynski, " Scheduling parallel batch jobs in grids with evolutionary metaheuristics", Journal of Scheduling Volume 18, Issue 4, pp. 345–357, 2015.
18. E. Gabaldon, J. Lerida, F. Guirado, J. Planes, "Blacklist multi-objective genetic algorithm for energy saving in heterogeneous environments", The Journal of Supercomputing, 2016.
19. H. Blanco, J. Llados, F. Guirado, J. L. Lerida , "Ordering and allocating parallel jobs on multi-cluster systems", In Proceedings of the 12th International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2012, pp. 196–206, 2012.
20. H. Blanco,J.L. Lerida,F. Cores, F.Guirado, " Multiple job co-allocation strategy for heterogeneous multi-cluster systems based on linear programming", J Supercomput 58(3):394–402, 2011.
21. P. Switalski, F.Seredynski, " A grid scheduling based on generalized external optimization for parallel job model", parallel processing and applied mathematics. Lecture Notes in Computer Science, 7204, 41–50,2012.
22. A. Bose, T.Biswas, P. Kuila, "A Novel Genetic Algorithm Based Scheduling for Multi-core Systems", S. Tiwari et al. (eds.), Smart Innovations in Communication and Computational Sciences, Advances in Intelligent Systems and Computing 851, 2019.
23. T.K. Ghosh, S. Das, " Solving Job Scheduling Problem in Computational Grid Systems Using a Hybrid Algorithm". In M. Sarfraz (Ed.), Exploring Critical Approaches of Evolutionary Computation, pp. 310-324. IGI Global, 2019.
24. T. Biswas, P. Kuila, A. Ray, "A novel scheduling with multi-criteria for high-performance computing systems: an improved genetic algorithm-based approach", Engineering with Computers, 2018.
25. M. Younis, S. Yang, Shengxiang, "A Genetic Algorithm for Independent Job Scheduling in Grid Computing", MENDEL - Soft Computing Journal, 23, pp. 65-72, 2017.
26. J. Kolodziej, "Security, energy, and performance-aware resource allocation mechanisms for computational grids", Future Generation Computer Systems Vol. 31,pp. 77–92, 2014.
27. P. S. Patel , "Multi-objective job scheduler using genetic algorithmin grid computing",Int J ComputAppl 92(14):34–43, 2014.
28. Z. Pooranian, M. Shojafar, R. Tavoli, M. Singhal, and A. Abraham, "A Hybrid Metaheuristic Algorithm for Job Scheduling on Computational Grids,"Informatica, vol. 37, no. 2, pp. 157, 2013.
29. Z. Poornian, "An efficient meta-heuristic algorithm for grid computing", Journal of Combinatorial Optimization, 2013.
30. J. Kolodziej, F. Xhafa , "Hierarchical genetic-based grid scheduling with energy optimization, Cluster Computing 16(3):1-19, 2012.
31. H. Liu, A. Abraham, A. E. Hassanien, "Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm", Future Generation Computer Systems 26 (8),pp. 1336–1343, 2010.
32. H. Yan, X. Q. Shen , X. Li , M.H. Wu , "An improved ant algorithm for job scheduling in grid computing", In IEEE International Conference on Machine Learning and Cybernetics, vol. 5, pp. 2957–2961, 2005.
33. A. Abraham,H. Liu, W. Zhang, T.G. Chang, "Scheduling jobs on computational grids using fuzzy particle swarm algorithm", InInternational Conference on Knowledge-Based and Intelligent Information and Engineering Systems. Springer, pp. 500–507, 2006.
34. L. Zhang,Y. Chen, R. Sun, S. Jing, B. Yang B, " A task scheduling algorithm based on PSO for grid computing",Int J ComputIntell Res 4(1):37–43, 2008.
35. J. Carretero, F. Xhafa, A. Abraham, "Genetic algorithm based schedulers for grid computing systems", International Journal of Innovative Computing, Information and Control 3 (6), pp.1–19, 2007.
36. T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, et al., "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", Journal of Parallel and Distributed computing 61 (6),pp. 810–837,2001.
37. D. B. Skillicorn, H.M.D. Jonathan, and M.F. William , "Questions and answers about BSP", Scientific Programming, 6(3):249–274, 1997.
38. J. L. Lérida, F. Solsona,P. Hernandez, F. Gine, M. Hanzich,J. Conde, " State-based predictions with self-correction on Enterprise Desktop Grid environments", Journal of Parallel and Distributed Computing 73(6): 777–789, 2013.
39. X.S. Yang, S. Deb, "Cuckoo search via Lévy flights", In IEEE World congress on nature and biologically inspired computing, pp. 210–214, 2009.
40. X.S. Yang, S. Deb, "Engineering optimisation by cuckoo search". Int J Math Model NumerOptim 1(4):330–343, 2010.
41. J. Kennedy, R. Eberhart, "Particle Swarm Optimization", Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948, 1995.
42. S. Srichandan, T. A. Kumar, S. Bibhudatta,"Task scheduling for cloud computing using multi-objective hybrid bacteria foraging algorithm",Future Computing and Informatics Journal,Volume 3, Issue 2,pp. 210-230, 2018.
43. M.F. Tasgetiren , Y.C. Liang , M. Sevkli,G.Gencyilmaz G, " Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem",Int J Prod Res 44(22):4737–4754, 2006.
44. Feitelson D (2005) Parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload.Accessed 05 June 2019.
45. Y. Li, Y. Liu, D. Qian, " A heuristic energy-aware scheduling algorithm for heterogeneous clusters", In 15th International Conference on Parallel and Distributed Systems (ICPADS), pp. 407–413, 2009.

AUTHORS PROFILE

Amit Chhabra works as an Assistant Professor in the department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar. His research areas include parallel job scheduling in multi-cluster, grid and cloud computing systems.

Gurvinder Singh works as Professor in the department of Computer Science in Guru Nanak Dev University, Amritsar. His research interestsincludejob scheduling, parallel and distributed computing systems viz. grid and clouds, Machine learning and Data science.

Karanjeet Singh Kahlon works as Professor in the department of Computer Science in Guru Nanak Dev University, Amritsar. His research interests includeComputer networks, Job scheduling in grid and cloud computing systems.