J Supercomput
DOI 10.1007/s11227-014-1373-9

A hyper-heuristic approach for resource provisioning-based scheduling in grid environment

Rajni Aron · Inderveer Chana · Ajith Abraham

Abstract Grid computing being immensely based on the concept of resource sharing has always been closely associated with a lot many challenges. Growth of Resource provisioning-based scheduling in large-scale distributed environments like Grid computing brings in new requirement challenges that are not being considered in traditional distributed computing environments. Resources being the backbone of the system, their efficient management plays quite an important role in its execution environment. Many constraints such as heterogeneity and dynamic nature of resources need to be taken care as steps toward managing Grid resources efficiently. The most important challenge in Grids being the job–resource mapping as per the users' requirement in the most secure way. The mapping of the jobs to appropriate resources for execution of the applications in Grid computing is found to be an NP-complete problem. Novel algorithm is required to schedule the jobs on the resources to provide reduced execution time, increased security and reliability. The main aim of this paper is to present an efficient strategy for secure scheduling of jobs on appropriate resources. A novel particle swarm optimization-based hyper-heuristic resource scheduling algorithm has been designed and used to schedule jobs effectively on available resources without violating any of the security norms. Performance of the proposed algorithm has also been evaluated through the GridSim toolkit. We have compared our resource scheduling

R. Aron (✉)
Computer Science and Engineering Department, LNMIIT, Jaipur, India
e-mail: rajni@lnmiit.ac.in

I. Chana
Computer Science and Engineering Department, Thapar University, Patiala, India
e-mail: inderveer@thapar.edu

A. Abraham
Machine Intelligence and Research Lab, Scientific Network for Innovation and Research Excellence, Auburn, WA, USA
e-mail: ajith.abraham@ieee.org

Published online: 20 January 2015

algorithm with existing common heuristic-based scheduling algorithms experimentally. The results thus obtained have shown a better performance by our algorithm than the existing algorithms, in terms of giving more reduced cost and makespan of user's application being submitted to the Grids.

Keywords Grid computing · Resource scheduling · Heuristic methods

1 Introduction

Grid computing has emerged as a computing platform to provide huge amount of resource sharing to large-scale scientific applications such as drug-design and protein folding. Due to heterogeneous and dynamic nature of the resources in Grid, resources are inexorably unreliable, which has a great effect on scheduling [1]. So, Grid resource management has become one of the most important key concerns in the field of Grid computing [2,3].

Grid resource management can be defined as a process consisting identification of requirements of the resources, matching resources to the applications followed by resource allocation and scheduling as well as monitoring the Grid resources finally over time to run Grid applications as efficiently as possible [46]. Grid resource management system is required to take resource management decisions which include resource provisioning and scheduling, while maximizing the Quality of Service (QoS) metrics delivered to the clients. Deployment of Grid systems involves the efficient management of heterogeneous, geographically distributed and dynamically available resources. These requirements introduce a number of challenging issues that need to be addressed such as resource provisioning-based resource scheduling and ensuring the scheduling of jobs on the trustworthy nodes. A major issue which needs due attention is to address the resource scheduling problem in Grid computing environment in a way ensuring secure execution of the jobs on ingredient resources leading to a more authentic performance. Resource provisioning is needed before scheduling of the resources or execution of Grid applications as Grid user often has a limited control over Grid Resources and the resource manager too is not always able to fulfill all the requirements due to the large number of resources as well as users requests.

It thus becomes a well-justified need to design an efficient resource provisioning-based scheduling algorithm. Grid scheduling is defined as the process of making scheduling decisions involving allocation of jobs to resources over multiple administrative domains [4]. Unlike traditional distributed systems, Grid computing environment provides a platform where resources and users work in different autonomous domains, thus making the mapping of jobs to appropriate resources for execution of application in Grid computing, a challenging NP-complete problem [11,44]. Heuristic methods often help to solve NP-complete problems. Heuristic approaches can easily be applied to Grid scheduling problems because Grid scheduling consists of various important issues such as heterogeneity of resources, dynamic and autonomous nature of Grid resources and finally the issue of different policies being followed by the resource providers and resource consumers for execution of their applications.

Hyper-heuristic approaches are more suitable to Grid environment as both resources and jobs are highly diverse and dynamic in nature. The term hyper-

heuristic describes choice-based heuristics in the context of combinatorial optimization.

Hyper-heuristic can be seen as a high-level methodology, which when given a particular problem instance or class of instances and a number of low-level heuristics automatically produces an adequate combination of the provided components to effectively solve the given problems [5].

The main motive of this work is to propose a hyper-heuristic-based scheduling algorithm being able to be applicable in Grid environment and hence scheduling resources to the preferred jobs leading to the delivery of optimum results to the Grid users. The main contributions of this paper are (1) a model of Grid resource scheduling, (2) particle swarm optimization (PSO)-based hyper-heuristic resource scheduling algorithm, (3) optimizing the cost and time for resource scheduling simultaneously, and (4) performance evaluation with respect to existing scheduling algorithms.

Rest of this paper is organized as further six sections. Section 2 presents related work. Section 3 deals with a description of Grid resource scheduling model. In Sect. 4, we present PSO-based hyper-heuristic resource scheduling algorithm for Grid environment. Section 5 shows the simulation study of various heuristic approaches and their comparison with the new proposed algorithm. Conclusion and future works have been presented in Sect. 6.

2 Related work

Resource provisioning and scheduling are major pillars of computational Grids and help it achieve high performance in its execution environment. Due the heterogeneous and dynamic nature of Grid resources, this basically has taken the form of a large-scale optimization problem. Observations suggest little emphasis on resource provisioning and security-based scheduling in Grid resource management as described in this section.

2.1 Resource scheduling without security

Abraham et al. [7] used nature's heuristics namely Genetic Algorithm (GA), Simulated Annealing (SA) and Tabu Search (TS) for scheduling of jobs on computational Grids. Authors have presented about better performance of GA over TS and SA for scheduling of the jobs to exact resources but hybrid heuristic algorithms perform better than GA approach as it minimizes the time required for scheduling the job.

In [39], a QoS-guided min-min heuristic is presented which can guarantee the QoS requirements of particular tasks and minimize the makespan at the same time. Carretero and Xhafa [21] have used GA for job scheduling in large-scale Grid applications. Authors have done several variations for GA operators to identify which worked best for the problem. Izakian et al. [40] used particle swarm optimization (PSO) for task scheduling in heterogenous computing systems and considered the makespan and flowtime as parameters. Authors did not used hyper-heuristic approach to design Grid scheduling algorithm and did not use the concept of resource provisioning before resource scheduling.

Abraham et al. [41] used PSO for scheduling job on computational Grids and extended the position and velocity of the particles in the conventional PSO from the real

vectors to fuzzy matrices. Zhao et al. [43] proposed SPSC, a flexible QoS-based service scheduling algorithm for service-oriented Grids. SPSC scheduling algorithm has been designed to provide QoS requirements by giving preference to user instead of resource provider. Above all, these works only considered limited scheduling objectives, more research is needed to give a flexible resource provisioning policy-based scheduling algorithm which supports both user's preference and resource provider's benefits. Resource provisioning policies have been discussed in our previous work [23].

Garg et al. proposed a model for meta-scheduling on utility Grids using linear programming and genetic algorithm. This model minimizes the cost for scheduling of an independent task. Authors have considered multiple and concurrent users who are competing for the resources in a meta-scheduling environment to minimize their cost [9]. Garg et al. [10] proposed three novel heuristics for parallel applications on utility Grids. The sensitivity of proposed heuristics on the basis of changes in user's preference, application's execution time and resource's pricing has been evaluated. A drawback of previously mentioned approaches being performing scheduling without any consideration of security and reliability as QoS parameters simultaneously.

Brun et al. [11] compared 11 static heuristics for mapping a class of independent tasks on heterogenous computing systems. A simulation model has been used for comparing static heuristics which returned the best optimum result. Gaoa et al. [13] developed two algorithms that used the predictive models to schedule jobs at both system level and application level. In application-level scheduling, GA is used to minimize the average completion time of jobs through optimal job allocation on each node. Xhafa et al. [14] surveyed the computational model and heuristic methods for Grid scheduling problems. The major drawback here being the resource scheduling algorithm based on resource provisioning policies not being designed considering the capability of node and the QoS expectation of the resource providers and resource consumers.

Gonzalez et al. [8] used ad-hoc (immediate and batch mode) scheduling methods to design hyper-heuristic approach for scheduling of jobs on the Grid nodes according to the job and Grid characteristics. A scheduling model for resource scheduling using heuristic methods has been designed by Bhanu et al. [6]. Longest Job Faster Resource (LJFR) heuristic and Shortest Job Faster Resource (SJFR) heuristic method have been used for resource scheduling in [6].

Authors did not consider the cost, makespan, security and reliability for independent job scheduling in the Grid environment. Techniques used in the traditional Grid scheduling have not been very successful in producing efficient and effective results to manage the resources. Our proposed implementation of hyper-heuristic-based resource scheduling algorithm minimizes the cost and makespan simultaneously. On the other hand, particle swarm-based hyper-heuristic has not been used for resource scheduling in the Grid earlier and we are proposing a new way while designing our resource scheduling algorithm.

2.2 Resource scheduling with security

The resource scheduling problem in Grid becomes more challenging as it is not only important to achieve the promising potentials of enormously distributed resources, but also using effective and efficient scheduling algorithms. Hence, a lot of algorithms

have been developed for scheduling jobs in a computational Grid and all of them aim to minimize the job completion time [30,45].

Different Grid scheduling approaches have been investigated and applied to different Grid scenarios and requirements. Some of the researchers have tried to schedule the jobs securely but to our best of the knowledge there is still a dearth of approaches covering resource provisioning-based scheduling. Kyriaki et al. [30] have considered multi-criteria job scheduling using accelerated genetic algorithm. Authors have considered security constraints for scheduling of jobs but resource provisioning-based scheduling has not been considered. Kolodziej et al. [31,32] have presented an approach for independent task scheduling with security requirements in Grid computing environment. Authors here have developed a scheduling model that enables the aggregation of task abortion and security requirements using a game-theoretic and genetic algorithm-based approach for optimizing the makespan and flowtime.

A fuzzy reputation-based ant algorithm for Grid scheduling has been designed in [42]. A fuzzy logic trust model for trust value aggregation through fuzzification has been used. Chen et al. [38] have used universal utility optimization function to design economic Grid resource scheduling algorithm by considering time and cost parameters.

In addition of these, fitness function of the genetic algorithm is dependent on the makespan of the solution ignoring the other scheduling criteria like cost and security. Using the proposed resource provisioning-based scheduling presented in this work, we can securely schedule the resources and avoid the run-time failures. In the above-mentioned approaches, security has been considered as a QoS parameter to provide QoS at the time of scheduling but scheduling based on QoS parameter-based resource provisioning policies have not been considered. These studies have many limitations (1) makespan is the only QoS parameter which has been considered for resource scheduling, (2) only user benefits have been considered ignoring the resource provider's benefits and (3) authors did not use the hyper-heuristic approach to solve the Grid resource scheduling problems. Hyper-heuristic is a good approach to solve various problems such as personal scheduling, timetabling, nurse scheduling and resource scheduling. Not being problem-specific like metaheuristic approach, it can be applied to any optimization problem.

The main problem in Grid environment is the selection of services as well as service providers in an appropriate way so as to achieve global Service Level Agreement (SLA) with minimum cost [33]. An integration of the security mechanisms with the scheduling algorithms still seems to be one of the most important and challenging problems in Grid computing [31].

The main aim of our study is to propose an efficient resource scheduling algorithm based on resource provisioning, taking into account cost and computation time as QoS parameters.

3 Grid resource scheduling model

Grid resource scheduling is the core of the Grid resource management systems. This process includes searching multi-administrative domains to use the available resources from the Grid infrastructure to satisfy the requirements of the user.

Grid scheduling being a two-step process: step one identifies the required set of resources as per the user requests whereas the second step maps the jobs on to the actual set of resources thus ensuring further a near-optimal satisfaction of QoS parameters. For example, if a person has to buy something from a shop the shopkeeper asks the buyer about his budget and then shows the items accordingly, the buyer now selects the most appropriate items among all the items shown that matches his budget and other specifications.

Figure 1 shows a Grid resource scheduling model.

- Each user tries to access the resources for application's execution through Grid portal. After this, the task of authorization and authentication is performed at the portal side.
- Resource Provisioning (RP) unit takes the information about the available resources from Resource Information Center (RIC) [23]. RIC collects all the information about the resource provider's resources and trust level of the resources from Resource Trust Manager (RTM). RTM will act as both resource manager and trust manager. Then, the RP unit performs preliminary provisioning for user's requests. On the completion of resource provisioning, the task of mapping to exact resources is performed.

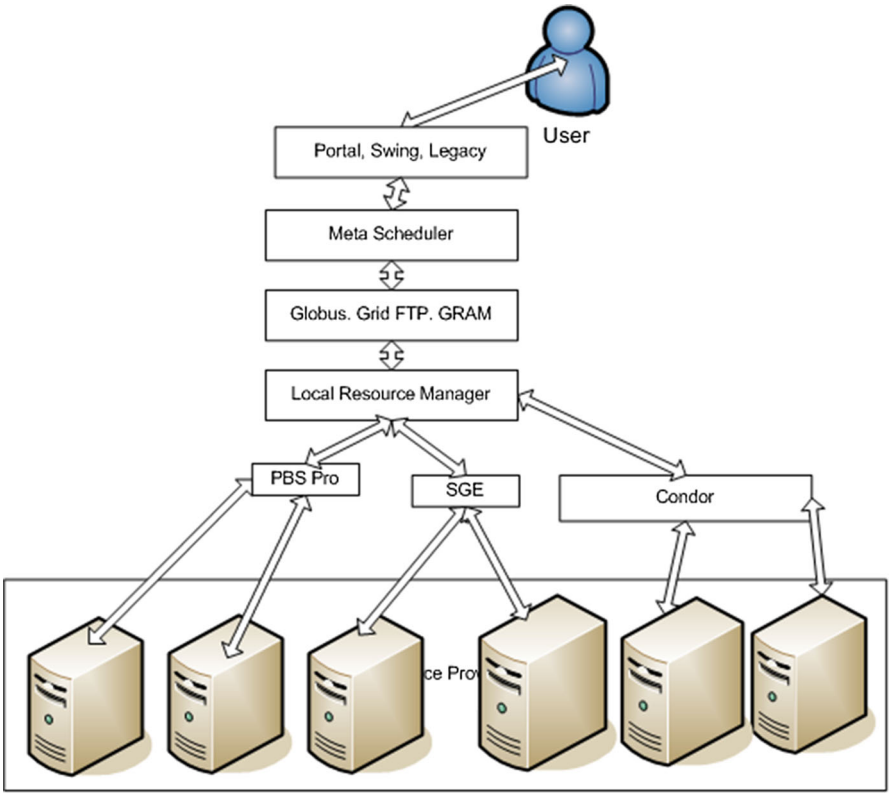

Fig. 1 Resource scheduling model

– Grid scheduler further takes all the information from RP unit which has a list of provisioned resources that are available in the Grid environment. According to the status of the resource, scheduler will consult with the Job Manger (JM) for job execution.
– Job Manager is a protocol engine for communicating with a range of different local resource schedulers using a standard message format [25]. Grid computing resources are typically operated under the control of a scheduler which implements allocation and prioritization policies while optimizing the execution of all submitted jobs leading to high efficiency and performance. Mapping of job to resources is then performed. Resources can be clusters of computers, memory spaces, storage capacity, files, attached peripheral devices, etc.

3.1 Requirements

There are some important characteristics which should be kept in mind while designing resource scheduling algorithm. The important requirements are as follows:

Efficiency Resource provisioning provides the facility to minimize the Grid overheads. Resource provisioning requires efficient management of the resources. It should also be efficient enough based on QoS parameter-based resource provisioning policies.

Efficient resource usage A resource scheduling algorithm should reduce the wastage of the resources. Jobs that are waiting for events (e.g., disk or user I/O, network latency, CPU usage, processor) should be handed over to the processor not to waste any of the resources. It should optimize the resource utilization by simultaneously optimizing cost and time.

Fair allocation The amount of resources allocated to each user should be independent of the number of jobs each user runs and a resource scheduling algorithm should be fair.

Adaptability and scalability A smart scheduler adapts as per the resources, i.e., whenever resources join or leave (dynamically), it manages the resources and jobs' execution process efficiently.

3.2 Problem formulation

To find the most suitable resource to a corresponding job is a tedious task and the problem of finding the best resource–job pair according to user's application requirement is a combinatorial optimization problem.

In this work, a resource scheduling problem has been designed considering the QoS expectations of both the resource providers and resource consumers. User wishes to minimize the cost whereas the resource provider wishes to minimize the makespan. In this problem, we have considered the most popular and extensively studied optimization criteria, i.e., the minimization of the makespan. Makespan is used to indicate the general productivity of the Grid systems. Smaller values of makespan indicate that the scheduler is planning the jobs in an efficient manner. Another optimization criterion is cost, which refers to the cost of the job execution on the particle resource along with the security-assurance cost.

To formalize the problem instance, we have used the Ali et al. computational model [35]. We have mathematically formalized the problem to get an optimal solution.

To consider this problem, we have taken a set of independent jobs $\{j_1, j_2, j_3, \ldots, j_m\}$ to map on a set of heterogenous and dynamic resources $\{r_1, r_2, r_3, \ldots, r_n\}$. $R = \{r_k | 1 \leq k \leq n\}$ is the collection of resources and n is the total number of resources. $J = \{j_i | 1 \leq i \leq m\}$ is the collection of jobs and m is the total number of jobs. The estimated time to compute value of each application/job on each resource is assumed to be supplied by the user. User gives the information, experimental data, job profiling and analytical benchmarking. The performance estimation for resource services is achieved using the existing performance estimation techniques such as analytical modeling [29] and historical information [28,31]. Under the expected time to compute (ETC) simulation model for problem formulation, we have considered the following constraints:

1. Each job to be scheduled for application's execution has a unique id.
2. Jobs are independent and indivisible.
3. Arrival of jobs for execution of application is random and jobs are placed in a queue of unscheduled jobs.
4. The computing capacity/speed of the resources is measured in multiple instruction per second (MIPS) as per Standard Performance Evaluation Corporation (SPEC) benchmark.
5. The processing requirement of job is measured in million instructions (MI).
6. Execution time for every job on resource is obtained from the ETC matrix. No of jobs $\times$ no of resources gives the size of the matrix and its components are defined as ETC(j_i, r_k). Rows of the ETC matrix demonstrate the estimated execution time for a job on each resource and the columns demonstrate the estimated execution time for a particular resource. ETC(j_i, r_k) is the expected execution time of job j_i and the resource r_k.

To enable more effective and security aware resource scheduling, it is desirable to know the security demand (SD) from Grid users at the time of job submission and the trust level (TL) assured by a resource provider at the Grid site. The first step is to issue an SD to all the available resource sites which is done by the user. The trust model requires assessing the resource site's trustworthiness, called the TL of a node. This information is taken from RIC unit. When an application is scheduled to execute on a resource, the trustworthiness of node reflects the reliability of the node's services. To address this problem, we are calculating trust of the node. The TL quantifies how much a user can trust a site for successfully executing a given job. A job is expected to be successfully carried out when SD and TL satisfy a security-assurance condition (SD $\leq$ TL) during the job mapping process [34].

We have defined the failure probability of a resource/machine as a function, which is dependent on the difference $\mathrm{SD}_i - \mathrm{TL}_k$. The formula (1) presented below expresses the failure probability of a resource scheduling r_k with trust level value TL_k, to a job j_i with a specific SD_i value [30]. In our model, a job could be delayed or dropped, if the site TL is lower than the job SD. The SD is a real fraction in the range [0, 1] with 0 representing the lowest and 1 the highest security requirement. The TL is in the same range with 0 for the most risky resource site and 1 for a risk-free or fully trusted site.

The negative exponent indicates that the failure probability of a scheduling grows with the difference $\mathrm{SD}_i - \mathrm{TL}_k$. The failure probability of executing a job, with a job SD on a site with TL, is modeled by an exponential distributed failure function as follows:

$$P_{\mathrm{f}}(j_i, r_k) = \begin{cases} 0 & \text{if } \mathrm{SD}_i \leq \mathrm{TL}_k \\ 1 - e^{-\alpha(\mathrm{SD}_i - \mathrm{TL}_k)} & \text{if } \mathrm{SD}_i > \mathrm{TL}_k \end{cases} \tag{1}$$

We have used a simple weighted sum function of makespan and cost to deal with their simultaneous optimization.

3.3 Objective function

In Grid scheduling, the main goal of the providers is to minimize the makespan whereas the goal of the user is to minimize the cost for Grid application. Fitness value is thus calculated as:

$$\text{Fitness function} = \theta \text{cost} + \delta \text{makespan} \tag{2}$$

where $0 \leq \theta < 1$ and $0 \leq \delta < 1$ are weights to prioritize components of the fitness function.

$$\text{cost} = \min(c(r_k, j_i)) \quad for\ 1 \leq k \leq n, 1 \leq i \leq m \tag{3}$$

Cost $c(r_k, j_i)$ is the cost of job j_i which executes on resource r_k in addition to security-assurance cost that is defined below;

$$c(r_k, j_i) = c_e(r_k, j_i) + c_s(r_k, j_i) \tag{4}$$

$$c_e(r_k, j_i) = \sum_{j_i \in J} \text{completion}(j_i, r_k) \Big/ \text{completion}_{m(j_i)} \times J \tag{5}$$

where as:

$$\text{completion}_{m(j_i)} = \max_{j_i \in J, r_k \in R} \text{completion}(j_i, r_k) \tag{6}$$

$$c_s(r_k, j_i) = \sum_{j_i \in J} P_f(j_i, r_k) \times \text{ETC}(j_i, r_k) \Big/ \text{ETC}_{m(j_i)} \times J \tag{7}$$

where as:

$$\text{ETC}_{m(j_i)} = \max_{j_i \in J, r_k \in R} \text{ETC}(j_i, r_k) \tag{8}$$

$$\text{makespan} = \min(F_{j_i}) j_i \in J. \tag{9}$$

Makespan is the finishing time F_j of the latest job and can be expressed as ETC job j_i on resource r_k. The completion time of a machine must be defined before calculating the makespan. Completion time indicates the time in which the machine/resource can

complete the execution of all the previous assigned jobs in addition to the execution time of job j_i on resource r_k, as defined below.

$$\widetilde{\mathrm{ETC}}(j_i, r_k) = (1 + P_f(j_i, r_k))\mathrm{ETC}(j_i, r_k) \tag{10}$$

$$\mathrm{completion}(r_k) = \mathrm{avail\_time}_{r_k} \pm \widetilde{\mathrm{ETC}}(j_i, r_k) \tag{11}$$

We can use the value of completion time to compute the makespan. This mapping is done with an objective of minimizing the cost and makespan simultaneously.

Algorithm 1: PSO based Hyper-heuristic with Great Deluge Resource Scheduling Algorithm

```
Data: Number of jobs and number of available
      resources.
Result: Mapping of the each job to the resources.
begin
    initialize Resource list[Number of Resources]
    initialize joblist[Number of Jobs]
    Input n= number of heuristics
    Input r = number of iterations
    Initialize a random feasible solution
    S= The number of particle in the population
    Initialize n and m
    for i = 1 To Population_size do
        P_velocity ← RandomVelocity()
        P_position ←
        RandomPosition(Population_size)
        P_p_best ← P_position
        For each particle , calculate the fitness
        if Fitness(P_p_best) ≤ Fitness(P_g_best) then
            P_g_best ← P_p_best
    while maximum iteration is not satisfied do
        for P ∈ Population do
            P_velocity ← UpdateVelocity(P_velocity,
            P_g_best, P_p_best)
            P_position ← UpdatePosition(P_position,
            P_velocity)
            if Fitness(P_position) ≤ Fitness(P_p_best)
            then
                P_p_best ← P_position
                if (P_p_best) ≤ Fitness(P_g_best) then
                    P_g_best ← P_p_best
    Return (P_g_best)
    Apply the heuristic
    while there are unscheduled jobs in the queue do
        for every resource is in resource list do
            get the next job from queue
            schedule the job on the resource on the
            basis of fitness
    Repeat each and every step till all the jobs are
    allocated
```

4 Hyper-heuristic-based resource scheduling algorithm

4.1 Pseudo code of algorithm

In this section, we present the pseudo code of PSO-based hyper-heuristic for resource scheduling in the Grid environment. Each particle in genome is a partial solution and is represented as a heuristic (e.g., select, move, swap, drop) or a sequence of heuristics. A low-level heuristic is any heuristic which is operated upon by the hyper-heuristic.

Low-level heuristics can be simple or complex and are implemented as follows: (1) job selection and scheduling: the heuristics select job from the unscheduled list and schedule it in to the best available resource. (2) Try for the best combination of all jobs and resources until the best combination is found. (3) Move job j_i from its current resource/schedule. (4) Swap jobs: select the jobs randomly which can swap. (5) Remove a randomly selected job from job pool already scheduled. This is the only heuristic which will move the search into an infeasible region because any job may be unscheduled.

We make sure that the search can move back into its feasible region by un-scheduling job that has other valid resources so that it can move into the next iteration. The low-level heuristics are then applied so as to find an optimal solution of the problem instance.

The objective of PSO is to find the best low-level heuristic that generates the best solution for resource scheduling problem. The selection process of low-level heuristic in hyper-heuristic stops after a pre-defined number of iterations. We set a fixed number of iterations to keep the computation time low. The particle rejects the new solution if it is poorer than the current solution. The pseudo code of PSO-based hyper-heuristic is given in Algorithm 1.

- First of all, scheduler will collect the information about resources and jobs from the user.
- A resource list is then obtained from the resource provisioning unit after provisioning of user's requests [23]. Once the resource list has been obtained, a job list and a random feasible solution are initialized.
- The task to choose the best heuristic from low-level heuristics is started.
- We have a number of particles, each of which represents a hyper-heuristic agent supplied with an initial solution in the solution space and an access to the evaluation function.
- Particle's position and velocity would be randomly initialized.
- It will then select a low-level heuristic at each particle position and compute its fitness function
  Fitness(P_{p_best}).
- If at P_{position}, Fitness(P_{p_best}) is better than
  Fitness(P_{g_best}) then P_{g_best} takes the value of P_{p_best}.
- We will try to find the Fitness value at best global position of the particle.
- After a particle has been chosen from the population, its position and velocity would be updated using Eqs. 11 and 12. Then, its fitness at the new position is calculated and compared with its previous position.

– If it is better than the local best value then we will assign particle's current position to the local best value.
– Now, we will compare fitness at P_{p_best} and P_{g_best}. If the fitness at P_{p_best} is better than at P_{g_best} then we will assign the value of P_{p_best} to P_{g_best}.
– After selection of a low-level heuristic, it is then applied to the problem. Resource scheduling is performed till there are no unscheduled jobs in the queue.

5 Performance evaluation and discussion

GridSim toolkit provides facilities of modeling, simulation of resources and network connectivity with different capabilities, configurations and domain. It also supports primitives for application composition, information services for resource discovery, interfaces for assigning application tasks to resources and manages their execution. Following are the reasons for the GridSim toolkit to be used for evaluation [24].

– It allows modeling of heterogeneous resources.
– Resources capability can be defined in the form of MIPS as per SPEC benchmark.
– There is no limit on the number of application jobs that can be simulated.
– Multiple user entities can submit tasks for execution simultaneously.
– It supports simulation of both static and dynamic schedulers.
– Application tasks can be heterogeneous and can be CPU or I/O intensive.
– Statistics of all or selected operations can be recorded and analyzed using GridSim statistic analysis methods.

For experimental results, heterogeneous resources are considered. In general, each resource may contain a different number of machines, and each machine may have one or more than one processing elements (PE) with different MIPS. In our results, we have assumed that each application/task which is submitted to the Grid may require varying processing time and input size and such type of task is defined in the form of Gridlets. A Gridlet is a package that contains all information related to the job and its execution management details such as job length, I/O operations and size of input/output files. The processing requirement of Gridlets is measured in multiple instructions (MI). Table 1 shows the characteristics of resources and Gridlets, that we have used for all our experiments.

For our evaluation, we have derived a suitable workload from real machine traces. These traces have been obtained from Grid workload archive website.[1] 5,000 User applications are generated according to the Lublin workload model [36]. The model specifies the arrival time, number of CPUs required and execution time μ of each application. This model is derived from existing workload traces for rigid jobs and incorporates correlations between job runtimes, job sizes, and daytime cycles in job-interarrival times. Using this generated workload, we have generated ETC matrix which is computed as the ratio of workload and computing capacity of machine vectors. No of jobs $\times$ no of resources gives the size of the matrix and its components are defined as ETC(j_i, r_k). Rows of the ETC matrix demonstrate the estimated execution time for

[1] More information about the real trace used can be obtained from the Grid Workload Archive at http://gwa.ewi.tudelft.nl/pmwiki/.

Table 1 Scheduling parameters and their values

Parameter	Value
Number of resource	150–250
Number of gridlets	5,000
Length of job	1,000–6,000
Bandwidth	3,000 or 7,000 B/S
Number of machine per resource	1
Number of PEs per machine	1–5
PE ratings	10–60 MIPS
Cost per job	3 G–5 G
File size	100 + (10–30 %) MB
Job output size	250 + (10–40 %) MB

a job on each resource and the columns demonstrate the estimated execution time for a particular resource. ETC(j_i, r_k) is the expected execution time of job j_i and the resource r_k. Each job can execute on each resource, and the estimated execution times of each job on each resource are known.

ETC matrices are classified into consistent and inconsistent matrices. Consistent matrix means that whenever a resource r_k executes the job j_i faster in comparison to r_l then the resource r_k executes all the jobs faster than r_l. Inconsistent matrix means that r_q may be faster in job execution than r_s for some cases and slower for others [35]. Resource heterogeneity represents the variation that is possible among the execution times for a given job across all the resources. The variation of the application's execution time on different resources can be high or low. A high variation in execution time of the same application is generated using the gamma distribution method. In the gamma distribution method, a mean task execution time and Coefficient of Variation (CV) are used to generate ETC matrices [35]. The mean task execution time of an application is set to μ and a CV value of 0.9 is used. Similarly, the low variation in the execution time is generated using uniform distribution with mean value of μ and a CV value of 0.3. The prices of resources are generated using Weibull distribution with parameters $\alpha = 0.3$ and $\beta = 0.7$.

5.1 Performance evaluation criteria

In this section, to evaluate the performance of a particle swarm-based hyper-heuristic for resource scheduling algorithm, we have defined the performance evaluation criteria. Two matrices, namely makespan and cost for evaluating the performance, have been selected. The former indicates the total execution time whereas the latter indicates the cost per unit resources that are consumed by the users for the execution of their applications. The makespan and cost are measured in seconds and Grid dollars (G$), respectively.

5.2 Results

In addition, a comparison of makespan and cost of the proposed algorithm vs existing heuristic algorithms GA [9,13,15,16], GA–TS [5] and Bacterial Foraging

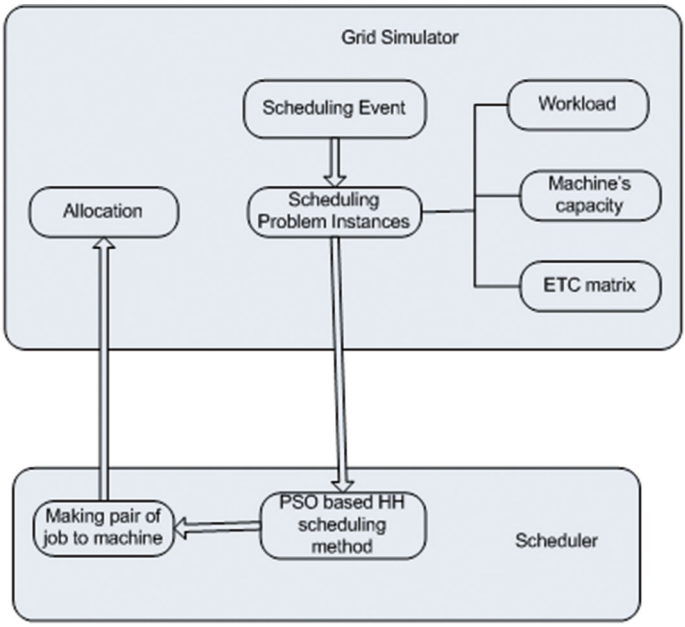

Fig. 2 Flowchart of scheduling in simulator

Optimization-based Hyper-Heuristic (BFOHH) [37] resource scheduling algorithm has been presented. To evaluate the performance of the proposed approach, we have investigated the effects of different number of applications. In all experiments, a comparison for consistent and inconsistent matrix has been done.

The main flow of scheduling in GridSim toolkit can be briefly described as follows. When a scheduling event is started, an instance of the scheduling problem is created by the simulator which is based on the current jobs and available resource pools as shown in the Fig. 2. The instance contains (a) workload; (b) computing capacity of machines; (c) prior load of machines and (d) the ETC matrix. The defined instance is then passed on to the scheduler which computes the planning of jobs to resources.

5.2.1 Performance for the high- and low-heterogeneous case

In this case, we evaluate the makespan and cost of the Grid applications in two different scenarios as (i) same number of applications/jobs is sent and (ii) different numbers of applications are sent. The pricing of resources may or may not be related to CPU speed. Thus, minimization of both makespan and cost of an application may conflict with each other depending on the price of the resources. In this analysis, low resource heterogeneity is simulated by each resource having a random number of PEs between 1 and 4. Figures 3 and 4 show the makespan of PSO-based hyper-heuristic vs GA, SA, GA–TS and BFOHH algorithms for inconsistent and consistent with low machine heterogeneity, respectively.

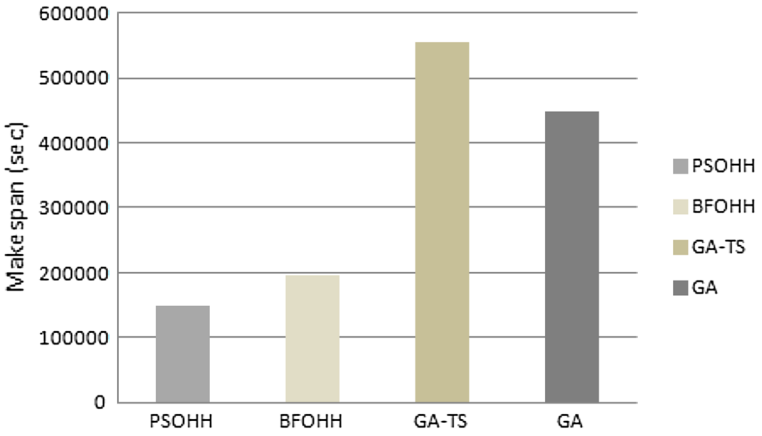

Fig. 3 Comparison result for inconsistent and low machine heterogeneity

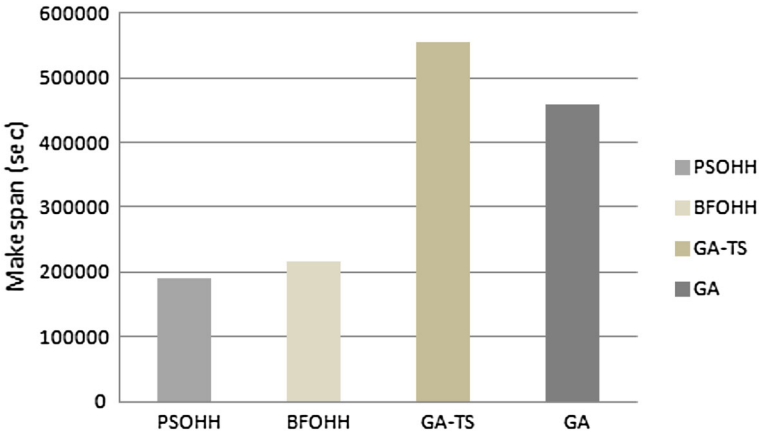

Fig. 4 Comparison result for consistent and low machine heterogeneity

Figures 5 and 6 show the cost of PSO-based hyper-heuristic vs GA, SA, GA–TS and BFOHH algorithms for inconsistent and consistent with low machine heterogeneity, respectively. When having low resource heterogeneity, PSOHH outperforms all the other approaches for both consistent and inconsistent matrices. The other three heuristics have higher cost in comparison to PSOHH for application execution.

The most important characteristic applicable to real-world scenarios is about how each algorithm responds to different heterogeneity of resources. A comparison of different makespans for both high and low resource/machine heterogeneity has been shown. The high resource heterogeneity is simulated by resources having the number of PEs between 7 and 30. Figures 7 and 8 show the effect on makespan by the four heuristics in case of high resource heterogeneity with inconsistent and consistent matrices. It can be seen from the figures that the makespan is lowest in case of PSOHH

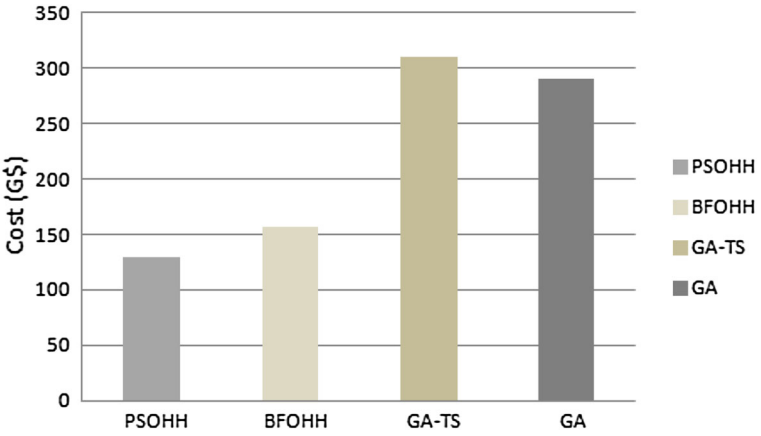

Fig. 5 Comparison result for inconsistent and low machine heterogeneity

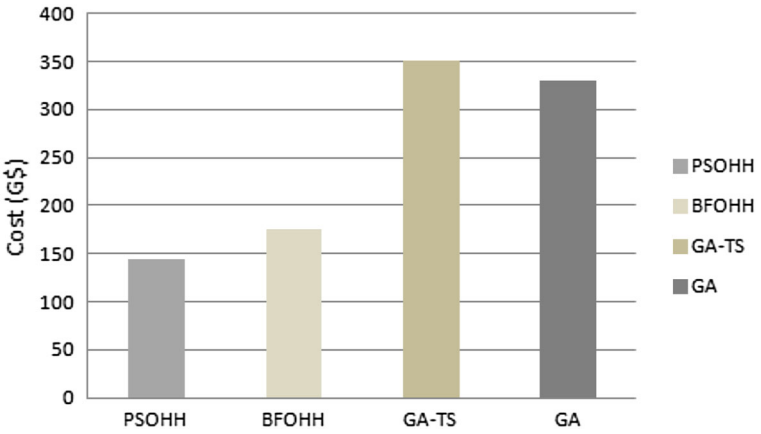

Fig. 6 Comparison result for consistent and low machine heterogeneity

for both the consistent and inconsistent matrices. This is because of the high variation in execution time across various resources as the resource list that is obtained from the resource provisioning unit is already filtered. This also demonstrates the effectiveness of the PSOHH in managing the time requirement of the user.

Figures 9 and 10 show the cost comparison for inconsistent and consistent matrices with high machine heterogeneity.

By analyzing the results in the Figs. 3, 4, 5, 6, 7, 8, 9, 10, we can conclude that PSO-based hyper-heuristic outperforms all the other approaches in cases of both low and high machine heterogeneity.

The results show that in case of GA, SA and GA–TS algorithms, if we send the same number of applications/jobs to the Grid, makespan and cost increase whereas in the case of PSO-based hyper-heuristic both makespan and cost decrease.

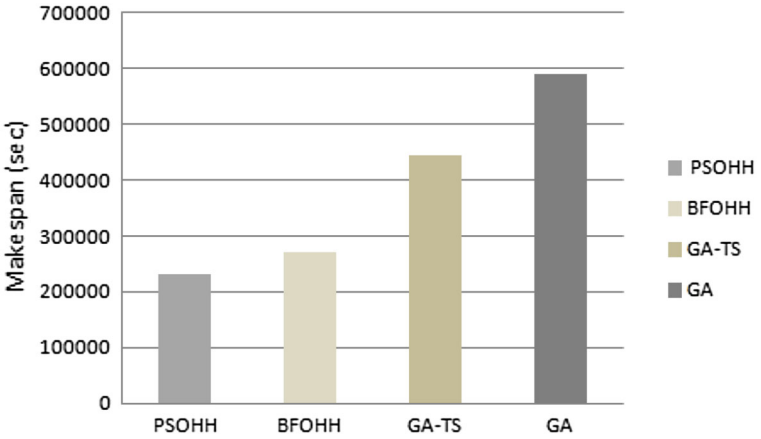

Fig. 7 Comparison result for inconsistent and high machine heterogeneity

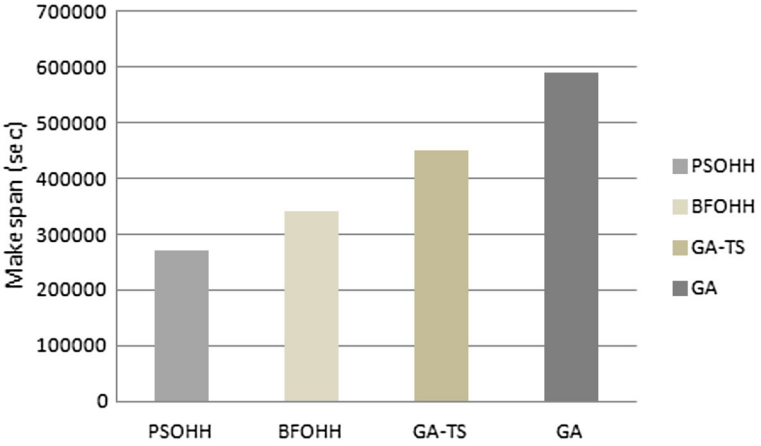

Fig. 8 Comparison result for consistent and high machine heterogeneity

5.2.2 Effect of the number of jobs

We have also performed experiments to determine the effect of an increase in the number of applications on cost and makespan. We have a 100-node simulated Grid with 2,000 jobs being sent to the Grid. From the experimental results shown in Figure 11, we can conclude that the time taken to execute an application can be reduced using PSO-based hyper-heuristic algorithm.

Figure 12 shows that cost per application increases as the number of submitted applications increases. The existing algorithm-based application's execution resulted in a schedule which is expensive in comparison to PSO-based hyper-heuristic algorithm. From all the experimental results, we observed that application execution using PSO-based hyper-heuristic algorithm provides the following advantages: the makespan is much lower in comparison to the GA, SA and GA–TS. The time variation in appli-

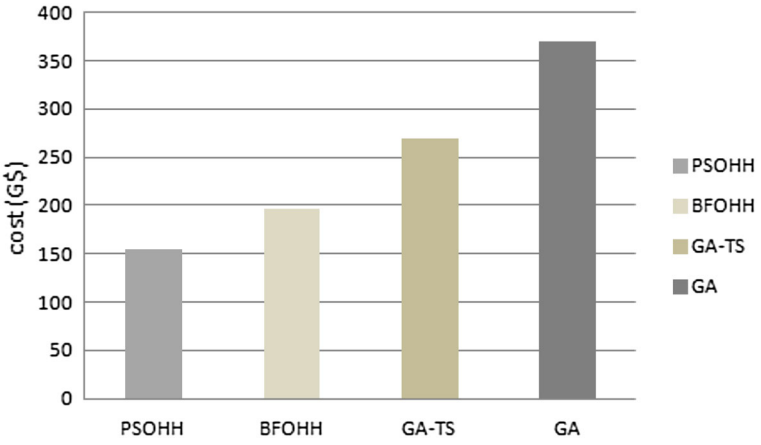

Fig. 9 Comparison result for inconsistent and high machine heterogeneity

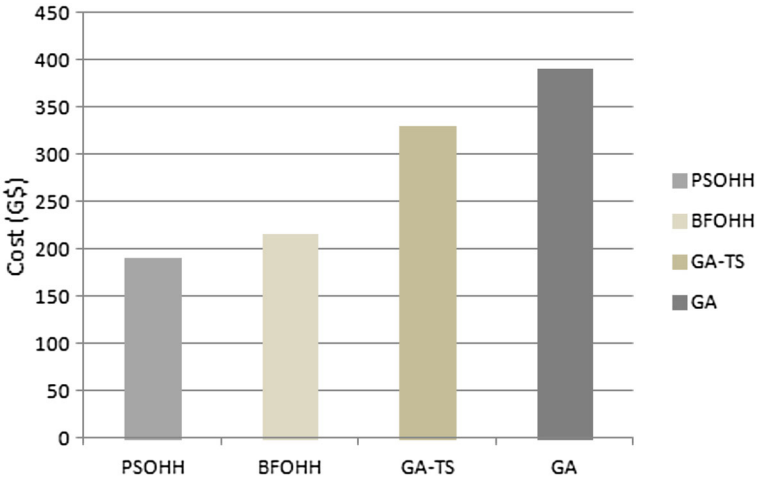

Fig. 10 Comparison result for consistent and high machine heterogeneity

cation's execution is about 4–7 %, which is much less in comparison to the 40–70 % variation in the existing algorithms using the same set of applications. The overall cost for user's application execution is less.

5.3 Effect of the number of resources

Figure 13 shows the effect of increasing the number of resources, while keeping the number of jobs being sent to the Grid constant. In this experiment, 700 jobs were sent to the Grid with varying number of resources. The results depict that by increasing the number of resources, the execution time increases thus decreasing the performance of the Grid. PSOHH and GA–TS perform better when the number of resources is less in

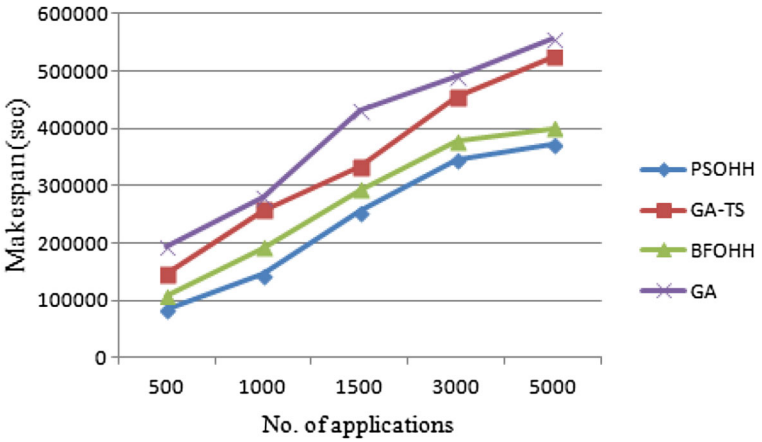

Fig. 11 Effect of the number of application on the makespan

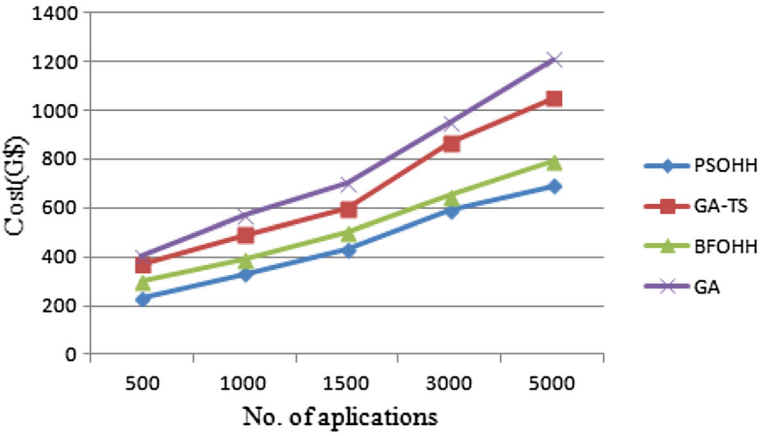

Fig. 12 Effect of the number of application on the cost

comparison to the number of jobs. The cost of application execution using PSOHH is much less in comparison to the execution cost using existing scheduling algorithms as shown in Fig. 14. As the cost variations within the Grid resources are not significant (i.e., 4G$ with 0.5G$) so the cost benefits of only 5–7 % were noticed. However, more benefits can be anticipated if the variations are higher.

5.4 Statistical analysis of results

In this section, we employed statistical method, namely the Coefficient of Variation (CV), to analyze the statistical significance of the results. The coefficient of variation is defined as the statistical measure of the dispersion of data around the average value. For comparison between datasets with different units or widely different means, we should use the coefficient of variation instead of the standard deviation. It

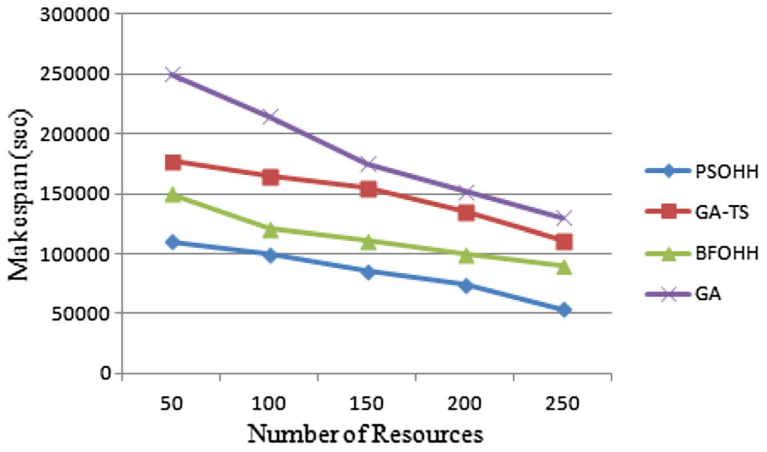

Fig. 13 Effect of the number of resources on the makespan

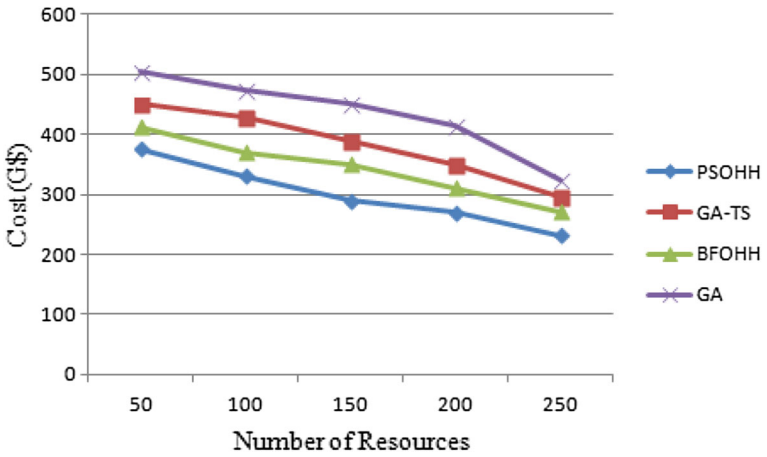

Fig. 14 Effect of the number of resources on the cost

expresses the variation of the data as a percentage of its mean value and is calculated as follows:

$$\text{CV} = (\text{standard deviation}/\text{mean}) \times 100 \tag{12}$$

The CV statistic is very useful in the analysis of the data series. It can also provide a general analysis of performance of the method used for generating the data. In Figs. 15 and 16, we examined the CV of the makepsan and cost of application's execution of each scheduling algorithm.

It can be observed that in all instances, the values of CVs are in the range 0–2 % which confirms the stability of the proposed algorithm. If the coefficient of variation is small, it means that BFO-based hyper-heuristic resource scheduling algorithm is more effective in scheduling of independent parallel jobs in the cases where the number of

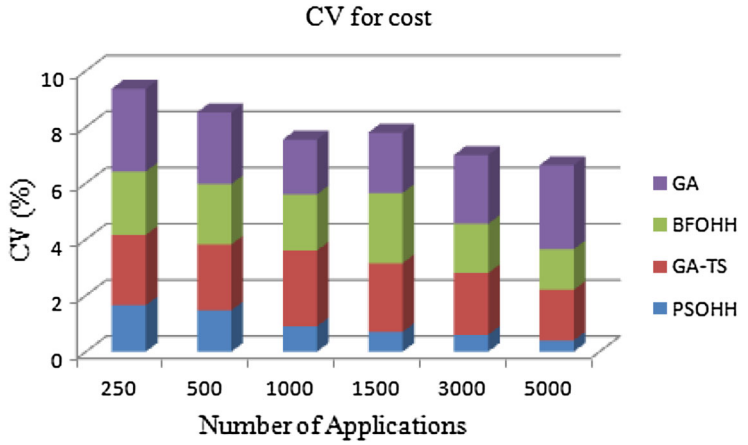

Fig. 15 Coefficient of variation for the cost with each algorithm

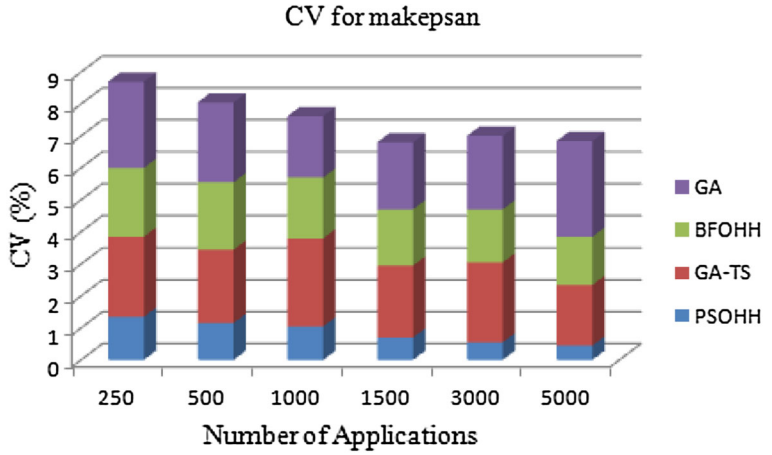

Fig. 16 Coefficient of variation for the makespan with each algorithm

application has changed. As the number of applications is increasing, the CV value decreases. It shows that our proposed algorithm outperforms other existing algorithms for large number of applications. Thus, we can say that as the system with small coefficient of variation value is more balanced, our proposed algorithm achieved the best results in the Grid with regard to both makespan and cost as QoS parameters.

5.5 Discussion

A good scheduling algorithm should schedule jobs to achieve high Grid performance while satisfying various user demands in an unbiased fashion. A hyper-heuristic approach for secure resource provisioning-based resource scheduling algorithm is designed. We have compared the performance of the PSO-based hyper-heuristic

resource scheduling algorithm with well-known scheduling algorithms such as GA, SA and GA–TS. We analyzed the performance of the proposed algorithm with variation in both the number of jobs and the number of resources, which are expected to vary in the real Grid environment. We evaluated the algorithm's performance with respect to makespan and cost. Makespan allows the evaluation of the algorithm which results in better scheduling in the sense of the duration of job execution, while the cost allows the comparison for resource selection. The proposed algorithm helps to achieve high performance and simultaneously and also to satisfy the user's requirements. In the experiments conducted, PSO-based hyper-heuristic resource scheduling algorithm clearly demonstrates its ability to provide better performance with respect to the existing Grid scheduling algorithms. As Grid computing has emerged for solving scientific, engineering and large-scale problems, it can be concluded that resources scheduling is one of the main challenging issues of Grid computing. Metaheuristic is highly adaptive in Grid computing environment but does not provide good solutions for more number of jobs in heterogeneous environment. Considering all these criteria and simulation results, it can be concluded that PSO-based hyper-heuristic resource scheduling algorithm provides a better solution for consistent and inconsistent matrices in cases of both low and high machine heterogeneity. It also outperforms all the existing scheduling algorithms in cases of varying jobs and resources thus providing near-optimal solution of Grid scheduling problems.

6 Conclusions and future works

In this paper, we have proposed particle swarm optimization-based hyper-heuristic resource scheduling algorithm for scheduling of jobs in Grid environment so as to minimize the cost and time by minimizing the makespan. Trust has been defined for the evaluation of the target nodes to provide security. To improve the efficiency of PSOHH, security has been calculated by probability of resource failure. We have presented simulation results, demonstrating that a hyper-heuristic approach for secure resource provisioning-based scheduling is effective in reduction of the overall cost and makespan of Grid applications. The results show that PSO-based hyper-heuristic approach gives better result in comparison to the existing common heuristic scheduling algorithms at different resource utilization levels. Attaining optimum scheduling results and management of the resources becomes considerably easier through resource provisioning.

The proposed algorithm not only minimizes the time and cost but also minimizes the makespan and security cost. In future, we wish to incorporate the concept of load balancing during scheduling of the resources. The performance of Grid would be shown by considering the average response time as a metric that is not considered in this work. Current results have been gathered through simulation on Gridsim but in future the same results would be verified on actual Grid resources present at Centre of Excellence (CoE) in Grid Computing at Thapar University.

Acknowledgments We would like to thank all anonymous reviewers for their comments and suggestions for improving the paper. We would like to thank Parteek Gupta for helping in improving the language and expression of a preliminary version of this paper. We are also grateful to the Grid Workloads Archive group

for making the Grid workload traces available. We also thank Dr. Dror Feitelson for maintaining the Parallel Workload Archive and all organizations and researchers who made their workload logs available.

References

1. Foster I, Kesselman C (2004) The Grid: blueprint for a future computing infrastructure. Morgan Kaufmann Publishers, USA
2. Aron R, Chana I (2010) Resource provisioning and scheduling in Grids: issues, challenges and future directions. In: Proceeding of IEEE International Conference on Computer and Communication Technology. MNNIT, Allahabad
3. Aron R, Chana I (2011) Resource provisioning for Grid: a policy perspective. In: Proceeding of Springer International Conference on Contemporary Computing (IC31). JP University, Noida
4. Khateeb AA, Abdullah R, Rashid AN (2009) Job type approach for deciding job scheduling in Grid computing systems. J Comput Sci 5(10):745–750
5. Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E, Qu R (2009) Hyper-heuristics: a survey of the State of the Art. University of Nottingham, technical report
6. Bhanu SMS, Gopalan NP (2008) A hyper-heuristic approach for efficient resource scheduling in Grid. Int J Comput Commun Control 3(3):249–258
7. Abraham A, Buyya R, Nath B (2000) Nature's heuristics for scheduling jobs on computational Grids. In: The 8th IEEE Conference on Advanced Computing and Communications. Cochin, India
8. Gonzalez JA, Serna M, Xhafa F (2007) A hyper-heuristic for scheduling independent jobs in computational Grids. In: International conference on software and data technologies, ICSOFT
9. Garg S, Konugurthi P, Buyya R (2008) A linear programming driven genetic algorithm for meta-scheduling on utility Grids. In: Proceedings of the 16th International Conference on Advanced Computing and Communication (ADCOM 2008, IEEE Press, New York, USA). Chennai, India
10. Garg SK, Buyya R, Siegel HJ (2010) Time and cost trade-off management for scheduling parallel applications on utility grids. Future Gener Comput Syst 26(8):1344–1355 (ISSN: 0167–739X, Elsevier Science, Amsterdam, The Netherlands)
11. Braun TD, Siegel HJ, Beck N, Boloni LL, Maheswaran M, Reuther AI, Robertson JP et al.(2001) A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J Parallel Distrib Comput 61(6):810837
12. Jun L, Chunlin L, Qingqing L (2010) A research about independent tasks scheduling on tree-based grid computing platforms, 2nd International Workshop on Intelligent Systems and Applications. Institute of Computer Science. Wuhan University of Technology, Wuhan, pp 1–4
13. Gaoa Y, Rongb H, Huangc JZ (2005) Adaptive Grid job scheduling with genetic algorithms. J Future Gener Comput Syst 21(1)
14. Xhafa F, Abraham A (2010) Computational models and heuristics methods for Grid scheduling problems. FGCS 26:608–621
15. Kim S, Weissman JB (2004) A genetic algorithm based approach for scheduling decomposable data Grid applications. In: International Conference on Parallel Processing. pp 406–413
16. Konugurthi PK, Ramakrishnan K, Buyya R (2007) A heuristic genetic algorithm based scheduler for clearing house grid broker, Technical Report, GRIDS-TR-2007-22. Grid Computing and Distributed Systems Laboratory. The University of Melbourne, Australia
17. Golconda K, Ozguner F (2004) A comparison of static QoS-based scheduling heuristics for a meta-task with multiple QoS dimensions in heterogeneous computing. Proceedings of 18th International Symposium on Parallel and Distributed Processing
18. Chakhlevitch K, Cowling P (2008) Hyperheurictics: recent developments. In: Cotta C, Sevaux M, Sorensen K (eds) Adaptive and multilevel metaheuristics, studies in computational intelligence, vol 136. Springer, pp 3–29
19. Dueck G (2002) New optimisation heuristics for the great deluge algorithm and the record-to-record travel. J Comput Phys 104:86–92 (Systems Magazine, pp 52–67)
20. Martino VD (2004) Sub-optimal scheduling in a Grid using genetic algorithms. In: Parallel and nature-inspired computational paradigms and applications. Elsevier Science Publishers, pp 553–565
21. Carretero J, Xhafa F (2006) Use of genetic algorithms for scheduling jobs in large scale Grid applications. Technol Econ Dev Econ 12(1):11–17

22. Cowling P, Kendall G, Han L (2002) An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In: Proceedings of the IEEE Congress on Evolutionary Computation. pp 1185–1190
23. Aron R, Chana I (2012) Formal QoS policy based Grid resource provisioning framework. J Grid Comput
24. Buyya R, Murshed M (2002) GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing, concurrency and computation: practice and experience (CCPE), vol 14. Wiley Press, New York, pp 1175–1220, ISSN: 1532–0626
25. http://www.globus.org/toolkit/docs/5.2/5.2.0/gram5/key/
26. Cowling P, Kendall G, Soubeiga E (2001) A hyper-heuristic approach to scheduling a sales summit, selected papers of proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling, vol. 2079. Springer LNCS, pp 176–190
27. Burke EK, Kendall G, Landa Silva JD O'Brien R, Soubeiga E (2005) An ant algorithm hyperheuristic for the project presentation scheduling problem, proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC). Edinburgh, UK, pp 2263–2270
28. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks. IV. pp 1942–1948
29. Tao F, Zhao D, Hu Y, Zhou Z (2008) Resource service composition and its optimal-selection based on particle swarm optimization in manufacturing Grid system. In: IEEE Transactions on industrial informatics, vol. 4, no. 4
30. Gkoutioudi KZ, Karatza HD (2012) Multi-criteria job scheduling in Grid using an accelerated genetic algorithm. J Grid Comput. doi:10.1007/s10723-012-9210-y
31. Kolodziej J, Xhafa F (2012) Integration of task abortion and security requirements in GA-based meta-heuristics for independent batch grid scheduling. In: Computers and mathematics with applications, Elsevier. doi:10.1016/j.camwa.2011.07.038, 63 350364
32. Kolodziej J, Xhafa F (2011) Meeting security and user behaviour requirements in Grid scheduling. Simul Model Pract Theory 19:213–223. doi:10.1016/j.simpat.2010.06.007
33. Menasce DA, Casalicchio E (2004) QoS in Grid computing. IEEE Internet Comput J 8(4)
34. Song S, Hwang K, Kwok YK (2006) Risk-resilient heuristics and genetic algorithms for security-assured grid scheduling. IEEE Trans Comput 55:703–719
35. Ali S, Siegel HJ, Maheswaran M, Hensgen D, Ali S (2000) Representing task and machine heterogeneities for heterogeneous computing systems. Tamkang J Sci Eng 3(3):195–207
36. Lublin U, Feitelson D (2003) The workload on parallel supercomputers: modeling the characteristics of rigid jobs. J Parallel Distrib Comput 63(11):1105–1122
37. Aron R, Chana I (2013) Bacterial Foraging based Hyper-heuristic for Resource Scheduling in Grid Computing. Future Gener Comput Syst 29(3):751–762. doi:10.1016/j.future.2012.09.005
38. Chen J (2010) Economic Grid resource scheduling based on utility optimization. IITSI, pp 522–525
39. He X, Sun X, Laszewski G (2003) A QoS Guided Min-Min Heuristic for Grid Task Scheduling. J Comput Sci Technol 18(4):442–451
40. Izakian H et al. (2009) A novel particle swarm optimization approach for grid job scheduling. Information Systems, Technology and Management. pp 100–109
41. Abraham A, Liu H, Zhang W, Chang TG (2006) Scheduling jobs on computational Grids using fuzzy particle swarm algorithm. In: Proceedings of 10th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems. England, pp 500–507
42. Zhou Z, Deng W, Lu L (2009) A fuzzy reputation based ant algorithm for Grid scheduling, cso. International Joint Conference on Computational Sciences and Optimization, vol. 1. pp 102–104
43. Zhao L, Ren Y, Li M, Sakurai K (2010) SPSE: a flexible QoS-based service scheduling algorithm for service-oriented Grid. In: 24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010. Atlanta, Georgia, pp 1–8
44. Kolodiej J, Khan SU, Gelenbe E, Talbi EG (2013) Scalable optimization in grid, cloud, and intelligent network computing. Concurr Comput 25(12):1719–1721
45. Liu Z, Qu W, Liu W, Li Z, Xu Y (2014) Resource preprocessing and optimal task scheduling in cloud computing environments. In: Practice and Experience, Concurrency and Computation
46. Buyya R, Abramson D, Giddy J, Stockinger H (2002) Economic models for resource management and scheduling in grid computing. Concurr Comput 14(13–15):1507–1542