J Supercomput (2013) 66:262–283
DOI 10.1007/s11227-013-0903-1

QoS based resource provisioning and scheduling in grids

Rajni Aron · Inderveer Chana

Published online: 14 March 2013

Abstract As Grid computing has emerged as a technology for providing the computational resources to industries and scientific projects, new requirements arise. Nowadays, resource management has become an important research area in the Grid computing environment. To provision the appropriate resource to a corresponding application is a tedious task. So, it is important to check and verify the provisioning of the resource before the application's execution. In this paper, a resource provisioning framework has been presented that offers a resource provisioning policy, which caters to provisioned resource allocation and resource scheduling. The framework has been formally specified and verified. Formal specification and verification of the framework helps in predicting possible errors before the scheduling process itself, and thus results in efficient resource provisioning and scheduling of Grid resources.

Keywords Grid computing · Quality of service · Resource provisioning · Resource scheduling

1 Introduction

Grid computing has provided the facility of resource sharing in a coordinates fashion to provide the non-trivial Quality of Service (QoS) [1]. The base of the Grid computing is the resource. To manage the resources in an efficient manner is not an easy task. As the Grid scales up, resource management complexity also increases and efficient resource management techniques are desired. To increase the efficiency of resource management systems, we have concentrated on resource provisioning and resource

R. Aron (✉) · I. Chana
Computer Science & Engineering Department, Thapar University, Patiala, India
e-mail: rajni@thapar.edu

I. Chana
e-mail: inderveer@thapar.edu

scheduling [2]. Before execution of any application, first of all resource provisioning is required. Resource provisioning allows the users and providers to access the specified resources according to the availability. Resource scheduling can be done effectively after resource provisioning as the user desires to discover the resources according to the requirements of the job, does resource provisioning, and then maps the resources to that job.

This paper addresses the enforcement of resource provisioning framework, QoS parameter(s) based resource provisioning policies for efficient provisioning of the resources and the resource scheduling algorithm for execution of the user's application. Most of the existing Grid models have only a security policy and there is no resource provisioning policy for efficient resource sharing. QoS parameters based resource provisioning policies such as a cost based resource provisioning policy is particularly useful for Grid scheduling because most often, resource providers have different resource policies for provisioning, but their main aim is to have maximum profit. If they use the same policy, then profit would be maximized and cost of resources for the user will automatically be less. So, the Grid resource provisioning and resource scheduling play a vital role in building an effective and a well-organized Grid environment [3].

The motivation of our work stems from the challenges in managing, sharing, and efficient utilization of the Grid resources. In real life situations, there are many constraints such as: (i) providing the guaranteed quality of service and (ii) minimizing the cost and time consumption for the verification of the policy before execution of any application, etc. The main aim of this work is to provision the resources before scheduling in an efficient manner and then execute the application so as to return optimal results to the user. This paper presents a resource provisioning framework. This framework results from common aspects of allocation of those resources for execution of application, requirements of job execution and QoS requirements for provisioning.

This paper is structured as follows: Sect. 2 describes the related work. A resource provisioning framework has been presented in Sect. 3. QoS parameters based resource provisioning policies, resource provisioning protocol, and resource scheduling are discussed. In Sect. 4, we have verified the resource provisioning framework. Section 5 discusses the experimental setup and results. We have provided the conclusion in Sect. 6.

2 Related work

Resource provisioning and scheduling in a Grid environment is challenging due to the dynamic and heterogeneous resources over geographical area. Most research deals with resource management systems in a Grid computing environment because the base of the Grid is the resource. Resource management systems are divided into the two most important parts, i.e., resource provisioning and resource scheduling.

2.1 Resource provisioning

Resource provisioning is done prior to scheduling. After provisioning of the resources, mapping of resources to jobs, i.e., resources scheduling is done. Resource

provisioning can be done in two ways: economic based approaches and noneconomic based approaches. In economic based approaches, resource provisioning is done on the basis of cost and time. In the noneconomic based approaches, resource provisioning is done on the basis of load balancing, round robin, first-come first-serve, etc. QoS-GRAF framework for QoS based Grid allocation with failure provisioning has been designed by Dasgupta et al. [16]. By considering Service Level Agreement (SLA) based service differentiation and failure provisioning, a linear relaxation based algorithm has been designed to improve the revenue. In this framework, only those jobs can be submitted to the resources that have multiple dependencies and differentiated QoS provisioning, but applications' execution deadline has not been considered, whereas our work considers both cost and deadline simultaneously at the time of implementation of the resource provisioning.

DRAGON [20] and GLARE [21] frameworks provide provisioning in a heterogeneous Grid environment. As per the DRAGON framework, network infrastructure is deployed that allows dynamic provisioning of network resources in order to establish deterministic paths in direct response to end-user requests. It also allows advanced e-science applications to dynamically acquire dedicated and deterministic network resources to link computational clusters, storage arrays, visualization facilities, remote sensors, and other instruments into globally distributed and application-specific topologies. GLARE provides distributed registries for activity types, activity deployments, and services that perform registration, provisioning, monitoring, and automatic deployments of new activities on different Grid computers in VO.

Raicu et al. [17] have proposed dynamic resource provisioning architecture using the existing system *falkon*. Here, allocation and deallocation policies are presented, and these policies are evaluated using metrics such as provisioning latency and accumulated CPU time. They have not considered time, cost, security, and reliability as QoS parameters for resource provisioning. The design of a gateway that provisions resources to dead-line applications relying on information given by current resource management services may be complex. It basically depends upon scheduling decisions that are far from optimal. So, when providers and brokers use conflicting policies, the number of migrations can be high [18]. Foster et al. [19] described a General-purpose Architecture for Reservation and Allocation (GARA) that supports flow-specific QoS specification, immediate and advance reservation, online monitoring, and control of both individual resources and heterogeneous resource ensembles. GARA does not support the concept of an agreement protocol and establishing a SLA for various resources.

2.2 Resource scheduling

Abraham et al. used nature's heuristics namely the Genetic Algorithm (GA), Simulated Annealing (SA), and Tabu Search (TS) for scheduling of jobs on computational Grids. They have shown that GA performs better than TS and SA for scheduling of the jobs to exact resources but hybrid-heuristic algorithms perform better than the GA approach as it minimizes the time required for scheduling the job [22]. Fidanova et al. designed ant colony optimization based grid task scheduling. They have used the concept of the Monte Carlo system to execute the application in the Grid computing environment [23]. Lorpunmanee et al. have developed a general framework

of grid scheduling using dynamic information. They have designed an ant colony optimization algorithm to improve the decision of scheduling [24]. These scheduling methods try to minimize the execution time/makespan of the applications and as such are suitable for Grids. However, in Grids, there is another important parameter other than execution time, i.e., cost. The meta-heuristic approach has been used in all these methods, but the hyperheuristic would have performed better in comparison to the metaheuristic.

Garg et al. presented a linear programming based metascheduling model for utility Grids and a genetic algorithm. This model minimizes the cost for scheduling of an independent task and considers multiple and concurrent users, which are competing for the resources in a metascheduling environment so as to minimize their cost [25]. To execute parallel applications on utility Grids, three heuristics have been proposed by Garg et al. [26]. They evaluated the sensitivity of the proposed heuristics on the basis of changes in the user's preference, application's execution time, and resource's pricing. Kolodziej et al. [27, 28] have presented an approach for independent task scheduling with security requirements in the Grid computing environment. They have developed a scheduling model that enables the aggregation of task abortion and security requirements. They have used game-theoretic and a GA based approach for optimizing the makespan and flowtime. The major drawback here was that the resource scheduling algorithm using the concept of resource provisioning has not been designed along with consideration of the QoS expectations of the resource providers and resource consumers. In our paper, we have obtained the provisioned set of resources through resource provisioning policies.

2.3 Our contributions

Our contribution in this paper is twofold. Firstly, a Grid resource provisioning framework for Grid systems has been defined incorporating resource provisioning. Verification of the QoS parameters based resource provisioning policies is done by the resource provisioning manager. XML schema has been used to design the resource provisioning policy, and its implementation has been shown in the resource provisioning framework.

Secondly, to execute the application on the ingredient resource according user's requirement, the Bacterial Foraging Optimization (BFO) based hyperheuristic resource scheduling algorithm has been proposed and implemented. This paper then formulated a combinatorial optimization model to produce near-optimal schedules for the independent parallel scheduling problem described in the Sect. 4. The proposed algorithm is used in the scheduling of independent parallel jobs in the Grid environment so as to simultaneously minimize the cost and the makespan. Optimization technique for optimizing the cost and makespan for resource scheduling simultaneously through a fitness function has been discussed. Finally, we have analyzed empirically the performance of the proposed resource provisioning based scheduling approach using the Aneka Grid Cloud computing platform.

Our proposed implementation of the resource provisioning based resource scheduling algorithm minimizes makespan and cost simultaneously, which has not been done by any of the techniques used earlier for Grid scheduling. None of the

techniques which have been used earlier were successful in producing efficient and effective results to manage the resources.

3 Resource provisioning framework

Resource providers give the facility of resource provisioning to user for optimum results and better services to avoid the violations of service level guarantees. The implementation of this framework will enable the user to analyze customer requirements and define processes that will contribute to the achievement of a product or service that is acceptable to their resource consumer.

3.1 Requirements for resource provisioning framework

Firstly, it is important to identify the resource provisioning's requirement before accessing and sharing of resources in the Grid environment. The following requirements have been identified by Dusseau et al. in [4].

- Efficiency: Resource provisioning provides the facility to minimize the Grid overheads. So, it requires efficient management of the resources.
- Efficient Resource Usage: It reduces the wastage of the resources. Jobs that are waiting for events (e.g., disk or user I/O, network latency, CPU usage, processor) should hand over the processor so that they do not waste any resources.
- Fair Allocation: The amount of resources allocated to each user should be independent of the number of jobs and provisioning of the resources should be fair.
- Adaptability and scalability: A smart scheduler adapts as per the resources, i.e., it manages the jobs' execution process efficiently even when the resources join or leave(dynamically).

3.1.1 Mode of operation

Figure 1 illustrates the resource provisioning framework. First of all, authenticated users will try to access the resource through the Grid portal for execution of the application. User fills the Service Level Agreement (SLA) form to send the request of the resources for the execution of the application. Then the Resource Provisioning Manger (RPM) performs the tasks, i.e., managing the SLAs, maintaining information about the resources and dynamically updating the resources' status. RPM checks the information about policy, which are stored in the policy repository and takes the information about the available resources from Resource information Center (RIC) as shown in Fig. 1. RPM checks for availability of the resources according to policy conditions and then provisions the resources to user's application [2]. The task of scheduling will then be performed. After getting the list of provisioned resources, the mapping of the job/application to the corresponding resource is done and the result will be again sent back to the user.

The main aim of resource provisioning framework is to provision the resources to Grid user with Quality of Service. This framework exhibits the way in which resource provisioning can be done in the Grid environment.

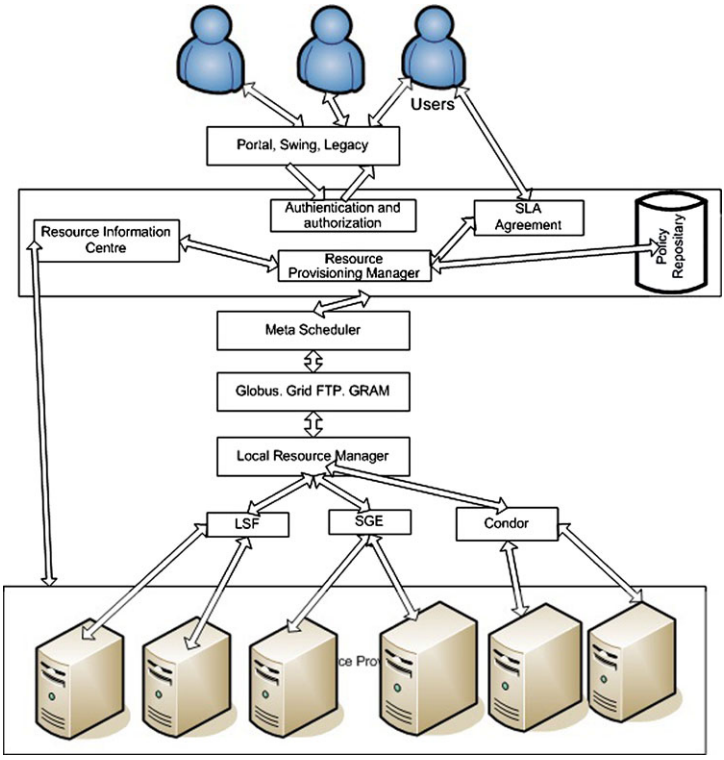

Fig. 1 Resource provisioning framework [2]

3.1.2 Resource provisioning protocol

In this section, a resource provisioning protocol has been described. The detailed requirements have to be gathered for designing a system, which furnish the solution for the desired problems and fulfills its goals. As these requirements need to be studied and analyzed from different perspectives, Unified Modeling Language (UML) [5] has been used.

Use case: user login In the first use case, we have demonstrated the successful registration of the user. The user will try to login to access the resources, and then he will be asked to submit the basic information on the registration page and password. Administrator confirms the registration of the user as shown in Fig. 2.

After the authentication, the user demands the resources for the application's execution. The resource information center will pass information about the availability of the resources to resource provisioning manager. RPM will provision the resources to the user and then the user will be able to access the resources as shown in Fig. 3.

Use case: successful execution of resource provisioning In this use case, we have shown the successful execution of the resource provisioning in Fig. 4. After the login through the portal, the user will try to access the resources. The user has to fill his

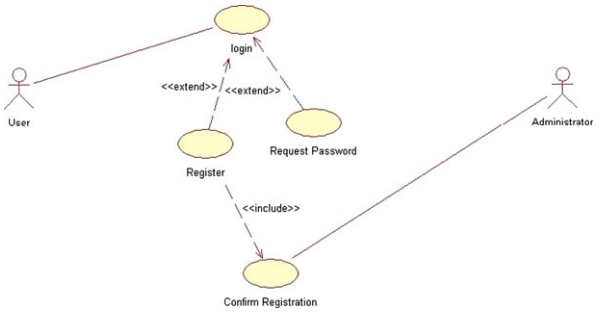

Fig. 2 Use case for user authentication

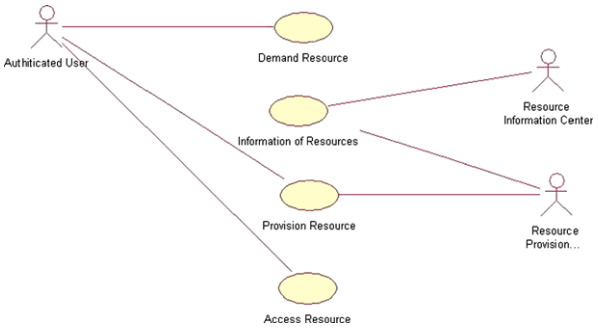

Fig. 3 Use case of resource provision

requirements in the form of SLA. After going through the SLA form, the resource provisioning manager tries to take the information of the available resources from the resource information unit and simultaneously will check the policy conditions. If the policy condition matches, then RPP will provision the resources to users.

Use case: resource provisioning manager cancels In this sequence provision model, we have shown the request of the resources for the execution of application through resource provisioning. After the login through the portal, the user will try to access the resources. The user has to fill his requirements in the form of SLA. After going through the SLA form, the resource provisioning manager tries to take the information of available resources from the resource information unit and simultaneously will check the policy conditions. If policy conditions do not match, then RPP will not provision the resources to users as shown in Fig. 5.

3.2 QoS based resource provisioning policy

Resource Provisioning framework provisions the resources on the basis of the QoS based resource provisioning policy. This policy standard is based on ISO:9000-2000, RFC 4745 [33].

3.2.1 Time based resource provisioning policy (TRPP)

In this policy, resource provisioning has been made on the basis of time, and takes static information as input. Users will submit their deadlines and the resource manager checks in the resource provider list that who can satisfy the requirements of the

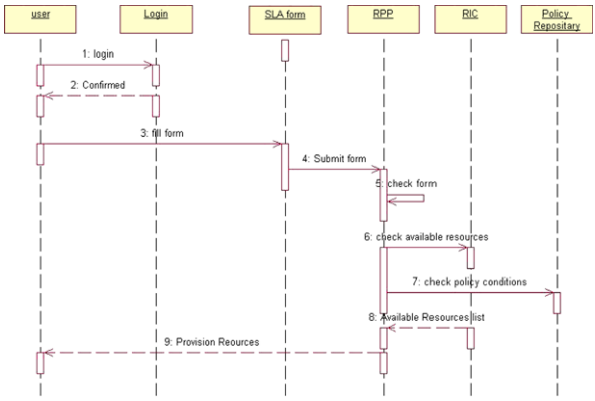

Fig. 4 Successful execution of resource provisioning

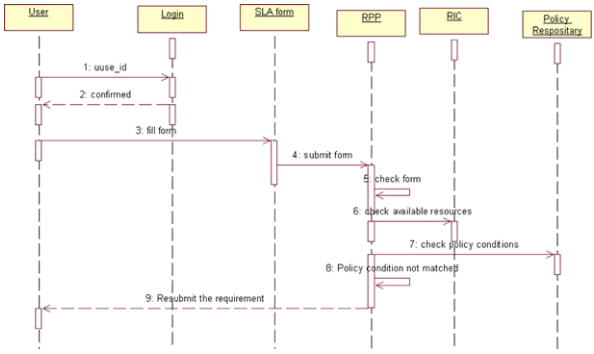

Fig. 5 RPP canceled the provisioning due to mismatch of policy

user. Time is calculated as the difference of the start time and deadline time. Then the resource manager will provision the resources to the user. The main aim of the Time based Resource provisioning policy (TRPP) is to minimize the time.

```
<?xmlversion="1.0"encoding="UTF-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attribute FormDefault="unqualified">
<xs:elementname="Time">
<xs:annotation>
<xs:documentation>
This is Time based Resource Provisioning
Policy
</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:elementname="application execution
service">
<xs:complexType>
```

```
<xs:attributename="resource"type="xs:string"
use="required"/>
<xs:attributename="type"type="xs:string"
use="optional"/>
</xs:complexType>
</xs:element>
<xs:elementname="compute QoS parameter Time"
minOccurs="0">
<xs:complexType>
<xs:attributename="startTime"type="xs:
dateTime"
use="required"/>
<xs:attributename="endTime"type="xs:
dateTime"
use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Similarly, other QoS parameters based resource provisioning policies such as cost, security, and reliability have been described in [2].

3.3 Grid resource scheduling

Grid scheduling is the core of the Grid resource management systems. It basically implies mapping jobs to the available ingredient resources. This process includes searching multiadministrative domains to use available resources from the Grid infrastructure in order to satisfy the requirements of the user [6]. Grid scheduling is a two-step process. In the first step, the required set of resources is identified as per the user requests and in the second step, the jobs are mapped onto the actual set of resources, thus further ensuring near optimal satisfaction of QoS parameters [11]. After getting the set of provisioned resources, the task of scheduling is performed in the resource provisioning framework.

3.3.1 Statement of the problem

To find the best 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. There are many scenarios in which the need of scheduling of independent jobs arises, so we have considered independent parallel applications to handle the realistic scenarios. They are also suitable to Grid systems as the Grid users submit jobs or monolithic applications in an independent manner to the system. Also, Grid systems are used for massive parallel processing, in which large amounts of data are processed independently [29]. Both the user's and resource

provider's point of view have been kept in mind while considering the scheduling problem as the user wants to minimize the cost, whereas the resource provider wants to minimize the makespan.

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 heterogeneous 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. The user gives the information, experimental data, job profiling, and analytical benchmarking. The performance estimation for resource services is achieved by using the existing performance estimation techniques such as analytical modeling [30] and historical information [31, 32]. In this problem formulation, we have considered some constraints such as: (i) each job to be scheduled for the application's execution has a unique id, (ii) jobs are independent and indivisible, and (iii) arrival of jobs for execution of application is random and jobs are placed in a queue of unscheduled jobs. Execution time for every job on resource is obtained from the ETC matrix. The number of jobs * number of resources gives the size of the matrix and its components are defined as $\mathrm{ETC}(j_i, r_k)$. 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. $\mathrm{ETC}(j_i, r_k)$ is the expected execution time of job j_i and the resource r_k.

3.3.2 Objective function

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

$$FitnessFunction = \theta \cos t + \delta makespan \tag{1}$$

$$\mathrm{cost} = \min\big(c(r_k, j_i)\big) \quad \text{for } 1 \leq k \leq n,\ 1 \leq i \leq m \tag{2}$$

$$makespan = \min(F_{j_i}) \quad \text{for } j_i \in J \tag{3}$$

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

The cost and makespan specified in Eqs. (2) and (3) have been used for the definition of an objective function for the independent parallel job scheduling as defined in Eq. (1).

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

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

whereas

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

Cost $c(r_k, j_i)$ is the cost of job j_i, which executes on resource r_k. Equation (5) denotes $c_e(r_k, j_i)$ user's application execution cost. In Eq. (6), the $completion_{m(j_i)}$ denotes the maximal completion time of the user's job. Makespan is the finishing time F_j of the latest job and can be expressed as Expected Time to Compute (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.

$$completion(r_k) = avail\_time_{r_k} \pm ETC(j_i, r_k) \tag{7}$$

The value of completion time has been used to compute the makespan. This mapping is done with an objective of minimizing the cost and makespan simultaneously.

3.4 Bacterial foraging based hyperheuristic resource scheduling algorithm

For mapping and execution of the job to the corresponding resource is done by using the proposed BFO based hyperheuristic resource scheduling algorithm.

3.4.1 Bacterial foraging optimization

The Bacterial Foraging Optimization (BFO) algorithm was proposed by Passino [9, 10]. It is a population based numerical optimization algorithm based on the foraging behavior of *Escherichia coli* bacteria. In the foraging theory, the objective of the animal is to search and obtain nutrients in a fashion that energy intake per unit time (E/T) is maximized. Foraging is a process in which a group of bacteria moves in search of food in a region; they decide whether or not to enter into a possible food region, and then search for a new food region so as to get high quality of nutrients. The bacterial foraging process consists of three main mechanisms: chemotactic, swarming, reproduction, and elimination-dispersal event. Chemotactic is the process of simulating the movement of *E. coli* bacteria, which is carried in a flagella, through swimming and tumbling. The cell also repels a nearby cell in the sense that it consumes nearby nutrients, and so it is not physically possible to have two cells at the same location. A bacterium in times of stress releases attractants to signal the bacteria to swarm together. After chemotactic steps, a reproduction step is taken. Fitness value of bacteria is sorted in an ascending order. The least healthy bacteria eventually dies while each of the healthier bacteria (those yielding lower value of the objective function) asexually splits into two bacteria, which are then placed in the same location. This keeps the swarm size constant. The elimination event may occur due to sudden changes like a significant local rise of temperature or a part of them may move to other regions in the environment that will affect the behavior of bacteria heavily. The elimination and dispersal event destroys the performance of the chemotactic event, but dispersal may place bacteria near good sources of food.

3.4.2 Hyperheuristic

A hyperheuristic operates at a higher level of abstraction. It selects a low-level heuristic that should be applied at any given time, depending upon the characteristics of the

region of solution space currently under exploration [7]. The main aim of the hyper-heuristic is not to compete with the state of the art problem specific approaches, but to provide a general framework, which is able to deliver solutions of good quality for a wide range of optimization problems [8].

3.4.3 Scheduling algorithm

In this section, we have presented the flowchart of bacterial foraging based hyper-heuristic for resource scheduling in the Grid environment. The region of the BFO search space is the possible combination of low-level heuristics. Each bacterium in genome is a partial solution and is represented as a heuristic or a sequence of heuristics. A low-level heuristic is operated upon by the hyperheuristic. Low-level heuristics can be simple or complex and can be implemented as follows: (1) First of all, select the job to be scheduled. The heuristic selects a job from the list of unscheduled jobs and schedule it to the best available resource that is filtered from the resource provisioning list. (2) Move job j_i from its current resource to some other resource. (3) Randomly select a job and swap it with some other job. (4) Finally, remove a randomly selected job from the job pool already scheduled [11].

Figure 6 shows the flowchart of BFO based hyper-heuristic resource scheduling algorithm. First, all the required parameters like the resource provisioned list, job list, and number of heuristics, etc., are initialized. Then we choose a low-level heuristic from the set of low-level heuristics to apply on the scheduling problem. Now, the task of low-level heuristic selection is performed. BFO can be used as a top level heuristic to manage the overall performance and quality of all the mechanisms. The main objective of the BFO is to find the best low-level heuristic that generates the best solution for the resource scheduling problem. We will apply the chemotactic, reproduction and elimination-dispersal loop to select the low-level heuristic, and for this we have calculated the fitness function. After selection of the heuristic, it will than be applied to the problem. We will perform this process until all the jobs are allocated. We then get the result.

4 Verification of the framework

For our work, we chose formal specification and verification of our framework has been done using Z formal specification language and the model checking technique. The proposed framework has been formally specified and verified using the Z formal specification language and model checking technique. Formal specification of the resource provisioning framework has been described in [2]. In this paper, we have formally verified the framework. Model checking is a verification technique that yields results much more quickly than theorem proving. It is based on the idea of exhaustive exploration of the reachable state space of a system [12]. Verification of our framework has been done using a spin model [13] due to the following reasons:

– an intuitive, program-like notation for specifying design choices unambiguously, without implementation detail.
– a powerful, concise notation for expressing general correctness requirements, and

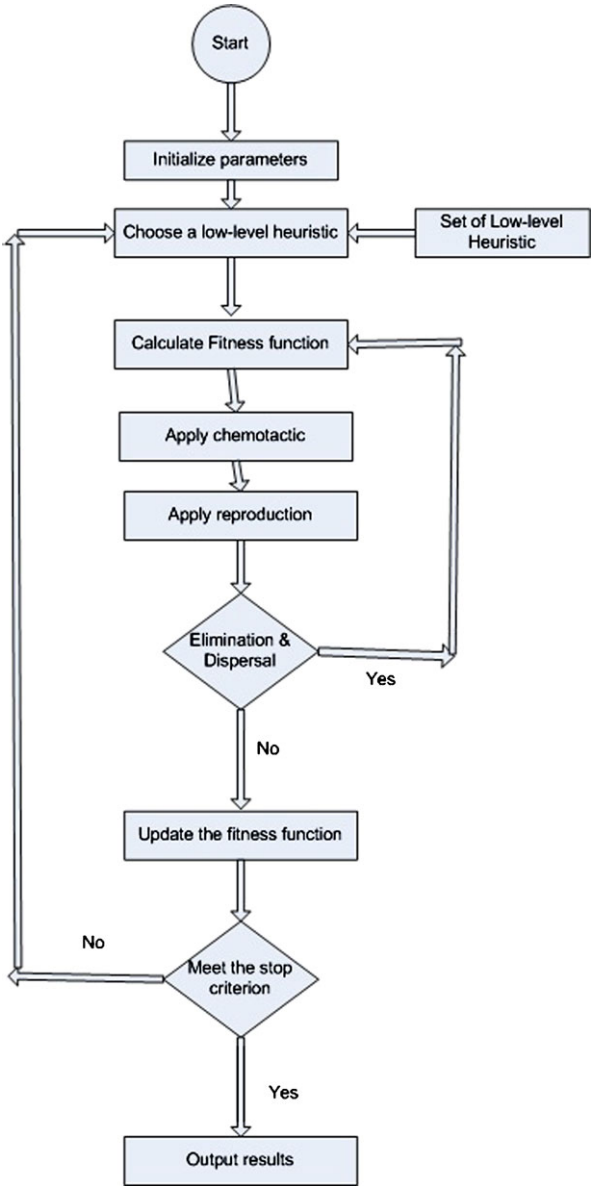

Fig. 6 Flowchart of the BFO based hyperheuristic resource scheduling algorithm

– a methodology for establishing the logic consistency of the design choices from (1) and the matching correctness requirements from (2).

Spin accepts specification in the Promela (a Process Meta Language) and Promela program consists of processes, messages, and variables, which are exchanges through FIFO pipelines. The exhaustive verifications performed by SPIN are conclusive. This makes SPIN a good choice for distributed systems. It accepts correctness claims specified in the syntax of Linear Temporal Logic (LTL) and converts that formula

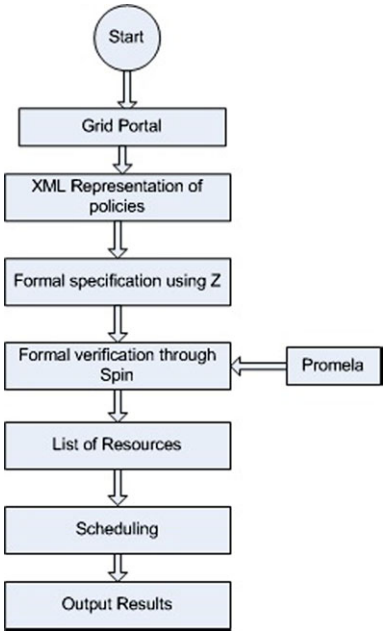

Fig. 7 Flowchart of resource provisioning framework

into Buechli automation [13]. Spin also checks the liveliness and deadlock of the systems. We have proved several specification properties with regard to the requirements of resource consumers. By construction of the protocol, the user can get appropriate resources for the application's execution by using the resource provisioning policy and vice versa. In addition, they can use different metrics for the selection of the resource: cost, time, security, and reliability are currently supported during the service level agreement. The properties, given as linear temporal logic formulae, can be converted to Buchli-automata automatically. The Buechli-automata are then evaluated for all states of the model. Usually, the property holds if the negated form of the corresponding Buechli-automaton does not terminate during the verification.

Figure 7 shows the flow and working of the resource provisioning framework. First of all, the user tries to access the resources through the Grid portal and QoS parameters based resource provisioning policies represented in the XML format for data storage. After the selection of the resource provisioning policy, the resource provisioning manager formally verifies the policy and we get the list of provisioned resources. Using the set of provisioned resources, the task of resource scheduling is performed. Figures 8, 9, 10 show the stages of the resource provisioning framework. In Fig. 8, we have shown the SLA form that is filled by the user for the application's execution in the Grid environment. Resource type, application type, and type of resource provisioning policy is offered by the resource provisioning manager as shown in Fig. 9. Figure 10 shows the complete overview of the SLA.

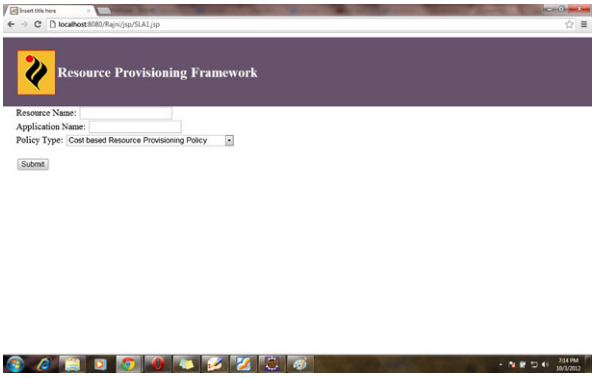

Fig. 8 Service level agreement

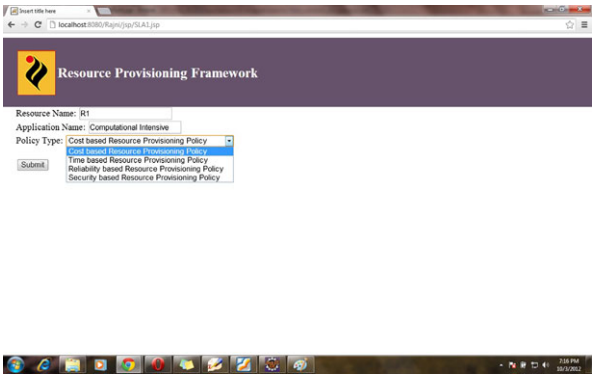

Fig. 9 Service level agreement of resource provisioning

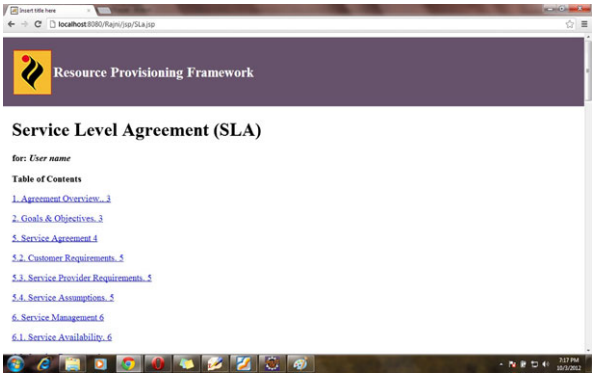

Fig. 10 Overview of service level agreement of resource provisioning

5 Experimental setup and results

Aneka [14] is a software platform and a framework for the development of distributed applications in the cloud. It harnesses the computing resources of a heterogeneous network of workstations, clusters, grids, servers, and data centers, on demand.

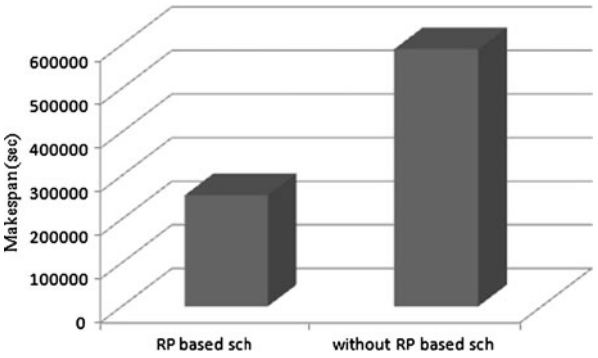

Fig. 11 Comparison of the makespan of RP based scheduling and without RP based scheduling

Aneka provides developers with a rich set of APIs for transparently exploiting such resources and expressing the business logic of applications by using the preferred programming abstractions. System administrators leverage a collection of tools to monitor and control the cloud, which can be a public virtual infrastructure available through the Internet, a network of computing nodes in the premises of an enterprise, or their combination.

5.1 Performance metrics

In this section, we have defined a performance evaluation criteria to evaluate the performance of a resource scheduling algorithm. We selected two matrices, namely makespan and cost for evaluating the performance. The former indicates the total execution time where as 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

To validate our algorithm, 5000 jobs/applications and 150–250 resources have been considered. We have presented the result using the Aneka Grid Cloud computing platform so as to test the performance of the hyper-heuristic based algorithm. We have used Basic Local Alignment Search Tool (BLAST) [15] application to run our algorithm. BLAST looks for similarities between a given sequence of genes and those stored into classified databases. For simplicity, we have used the Parameter Sweep Model in order to automatically perform multiple BLAST queries against the same database over a distributed infrastructure.

Testcase 1

In first test case, we have evaluated the makespan and cost of Grid applications in two scenarios as (i) the same number of applications/jobs are sent and (ii) different number of applications are sent. The pricing of resources may or may not be related to CPU speed. Thus, minimization of both makespan (execution time) and cost of an application may conflict with each other depending on the price of the resources. Figures 11 and 12 show the makespan and cost of resource provisioning

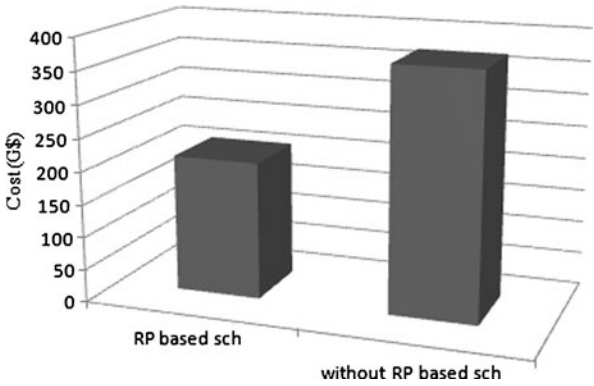

Fig. 12 Comparison of the cost of RP based scheduling and without RP based scheduling

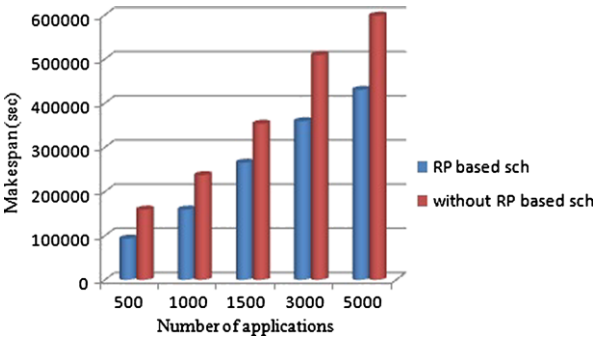

Fig. 13 Effect of change in number of application submitted on makespan

based scheduling vs. nonresource provisioning based scheduling, respectively. The results show that in the case of nonresource provisioning based scheduling, if we send the same number of applications/jobs to the Grid, makespan and cost increases where as in the other case, both makespan and cost decreases. This is expected as the application execution is done using the proposed algorithm which is based on QoS parameter (s) based resource provisioning policies. After the formal verification of the resource provisioning policies, we get filtered resources, which enables it to take an optimal decision.

Testcase 2

We have also performed experiments to determine the effect of increasing the number of applications on the cost and makespan. From the experimental results shown in Fig. 13, we can conclude that the time taken to execute an application reduces by using the resource provisioning based scheduling approach. Figure 14 shows that cost per application increases as the number of submitted application increases. The nonresource provisioning based scheduling approach resulted in a schedule, which is expensive in comparison to resource provisioning based scheduling approach as the number of applications increases. The reason is that the nonresource provisioning based scheduling approach does not consider the effect of other applications in the meta-scheduler at the time of job submission but in the resource provisioning

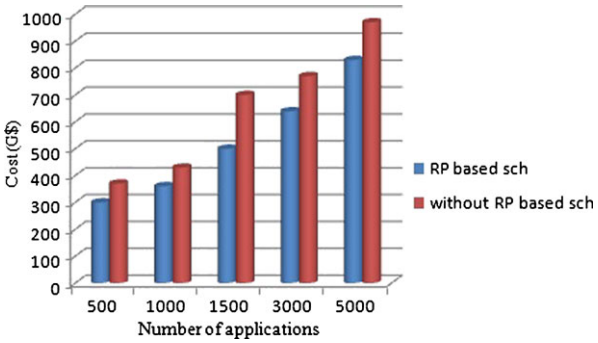

Fig. 14 Effect of change in number of application submitted on cost

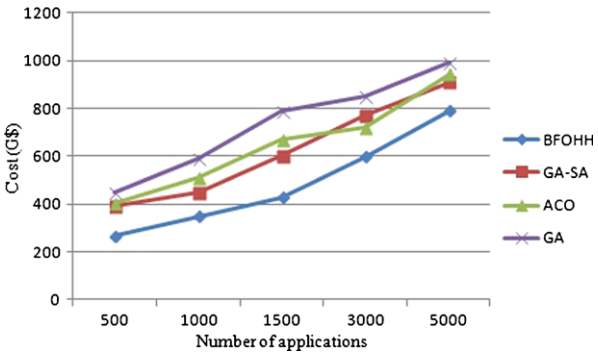

Fig. 15 Comparison of the cost for scheduling algorithms

based scheduling approach, it is considered according to both user's and resource providers's perspectives.

Testcase 3

We have also performed experiments to determine the effect of increasing the number of applications on the cost and makespan. Figure 15 shows that cost per application increases as the number of submitted applications increases. The other scheduling heuristic algorithm resulted in a schedule, which is more expensive in comparison to the bacterial foraging based hyperheuristic resource scheduling algorithm as the number of applications increases. As the cost variations within the Grid resources are not significant (i.e., 5G $ with 0.5G $) so the cost benefits of only 7–11 % were noticed. However, more benefits can be anticipated if the variations are higher. Thus, BFOHH outperforms all the existing scheduling algorithms when the application execution cost is high, which is an important case for a large-scale Grid computing environment.

Testcase 4

In this case, the proposed algorithm has been implemented in the Aneka Grid Cloud computing platform. The proposed algorithm has been compared with the other existing scheduling algorithms such as ant colony optimization, genetic algorithms, and GA–SA. In this case, we have varied the number of applications. It shows the effect of increasing the number of applications. The makespan of application execution using BFOHH is much less in comparison to the other existing scheduling algorithms. The reason is that this is because of the high variation in execution time

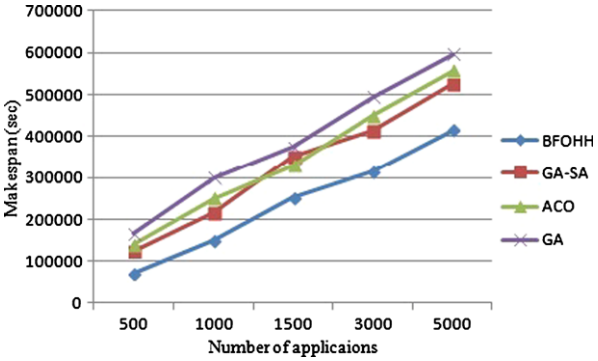

Fig. 16 Comparison of the makespan for scheduling algorithms

across various resources as the resource list that is obtained from the resource provisioning unit is already filtered. From the experimental results shown in Fig. 16, we can conclude that the time taken to execute an application reduces by using the proposed resource scheduling algorithm.

5.3 Framework validation

In the proposed framework, resource scheduling has been done on the basis of the QoS parameter(s) based resource provisioning policies, which was not considered traditionally. Figures (in the experimental results section) show that QoS based provisioned approach generates better results and the resource provisioning based scheduling algorithm is able to schedule the job on the ingredient resources more efficiently. The resource provisioning framework has been validated against the features of the existing resource provisioning frameworks and its result has been depicted in Table 1.

6 Conclusion

In this paper, a resource provisioning framework has been implemented. We allow both users and providers to take benefits from Grid resources and policy. In this paper, we have presented a resource provisioning framework for Grid computing environment. The objective is to minimize the complexity of provisioning for job execution in Grid computing. With the use of a uniform policy, the number of migrations and complexity of policies can be less, and thus the performance will be high. Specification and verification of the formal Grid resource provisioning framework has been done with the help of the Z specification language and model checking using the spin tool. After resource provisioning, we get the set of provisioned resources and after that the mapping of the resources to the corresponding job is done by using the resource scheduling algorithm. The algorithm has been implemented to run the Blast application in Aneka Grid Cloud computing platform. The experimental results shows that the performance of the proposed algorithm outperforms the existing algorithms in terms of cost and time.

Table 1 Comparison of Resource Provisioning Framework with existing frameworks

RPF/ Features	RPF	Dragon	Glare	GARA	QoS-GRAF	RAA/Falkon	Intra Grid	Ali's Framework
Usage	Resource provisioning Framework provides the facility of provisioning of the resources.	It discussed about the Nw architecture which can provide dynamically provisioning.	It provides the facility of service provisioning.	It supports flow-specific QoS specification, immediate and advance reservation and online monitoring and control of both individual resources and heterogeneous resource ensembles.	It optimized business matrices based on SLA driven pricing policies.	It is the combination of DRP & falkon.	It enables the deployment of applications across multiple Grids.	It worked before resource provisioning to discover resources.
Middleware	GT4	The model and capabilities are platform independent. Can be applied to different middleware like GT	Globus	GRAM	The model and capabilities are platform independent. Can be applied to different middleware like GT	PBS	SGE	GT 2.0
Resource Provisioning based Scheduling	Yes	NO	No	No	No	No	No	No
Policies	CRPP, SRPP, TRPP, RRPP	No Policies	No Policies	No Policies	No Policies	Allocation & De-allocation policies	Conservative Backfilling & Multiple site partition policies	Reservation Policy
QoS Parameters	Submission burst, Cost, Time, Security and Reliability	No parameters	Security, Reliability	Reservation	Revenue	Time	Response Time	Advance Reservation
Validation	Formal Methods, Z formal specification Language	No	No	No	No formal method verification	No	No	No

7 Future directions

In the future, the performance of the Grid would be shown by considering the scalability as a metric that is not considered in this framework. The proposed framework can be further enhanced by introducing dynamic load balancing techniques at the time of scheduling. The existing solutions have been designed and tested only for resource provisioning and scheduling challenges. In the future, this can also be extended for addressing the other resource management challenges such as re-scheduling and resource monitoring, etc. It will be interesting to enhance the proposed algorithm to schedule jobs with different application models such as workflows and bag-of-tasks.

References

1. Foster I, Kesselman C (2004) The grid: blueprint for a future computing infrastructure. Morgan Kaufmann, San Mateo
2. Aron R, Chana I (2012) Formal QoS policy based grid resource provisioning framework. J Grid Comput 10(2):249–264
3. Rajni A, Chana I (2010) Resource provisioning and scheduling in grids: issues, challenges and future directions. In: International conference on computer and communication technology (ICCCT'10), MNNIT, Allahabad, 17–19 September 2010, pp 306–310
4. Dusseau ACA (1998) Implicit co scheduling: coordinated scheduling with implicit information in distributed systems. PhD thesis, University of California at Berkeley
5. Rumbaugh J, Jacobson I, Booch G (2004) The unified modeling language reference manual, 2nd edn. Addison-Wesley, Pearson Education, Upper Saddle River. PGrady booch, object-oriented analysis and design
6. 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
7. Cowling P, Kendall G, Soubeiga E (2001) A hyper-heuristic approach to scheduling a sales summit. In: Proceedings of the 3rd international conference on the practice and theory of automated timetabling. Lecture notes in computer science, vol 2079. Springer, Berlin, pp 176–190
8. Gonzalez JA, Serna M, Xhafa F (2007) A hyper-heuristic for schedulingin dependent jobs in computational grids. In: International conference on software and data technologies (ICSOFT)
9. Passino KM (2002) Biomimicry of bacterial foraging for distributed optimization and control. IEEE Control Syst Mag 22(2):52–67
10. Liu Y, Passino KM (2002) Biomimicry of social foraging bacteria for distributed optimization: models, principles and emergent behaviors. J Optim Theory Appl 115(3):603–628
11. Aron R, Chana I (2012) Bacterial foraging based hyper-heuristic for resource scheduling in grid computing. Future generation of computer systems. Elsevier, Amsterdam. doi:10.1016/j.future.2012.09.005
12. Hesselink WH (2004) Introduction to the model checker spin, 4th October 2004. Online at http://wenku.baidu.com/view/ed1d002d453610661ed9f446.html
13. Holzmann GJ (1997) The model checker SPIN. IEEE Trans Softw Eng 23:279–295
14. Vecchiola C, Chu X, Buyya R (2009) Aneka: a software platform for .NET-based cloud computing. In: Gentzsch W, Grandinetti L, Joubert G (eds) High speed and large scale scientific computing. IOS, Lansdale, pp 267–295
15. Blast. www.ncbi.nlm.nih.gov/BLAST/
16. Dasgupta G, Dasgupta K, Purohit A, Viswanathan B (2006) QoS-GRAF: a framework for QoS based grid resource allocation with failure provisioning. In: Proceedings of 14th IEEE international workshop on QoS (IWQOS'06), 19–21 June, New Heaven, CT, USA, pp 281–283
17. Raicu I, Zhao Y, Dumitrescu C, Foster I, Wilde M (2007) Dynamic resource provisioning in grid environments. In: TeraGrid conference, June 2007
18. Assuncao MD, Buyya R (2009) Performance analysis of allocation policies for intergrid resource provisioning. Inf Softw Technol 51(1):42–55

19. Foster I, Fidler M, Royd A, Sander V, Winkler L (2004) End-to-end quality of service for high-end applications. Comput Commun J 27(14):1375–1388
20. Lehman T, Sobieski J, Jabbari B (2006) DRAGON: a technique for service provisioning in heterogeneous grid networks. Communications Magazine, IEEE 44(3):84–90
21. Siddiqui M, Villazon A, Hofer J, Fahringer T (2005) GLARE: A grid activity registration, deployment and provisioning framework. In: Proceedings of ACM/IEEE conference on supercomputing, 12–18 November 2005
22. 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
23. Fidanova S, Durchova M (2006) Ant algorithm for grid scheduling problem. Lecture notes in computer science, vol 3743. Springer, Berlin, pp 405–412
24. Lorpunmanee S, Sap MN, Abdullah AH, Chompoo-inwai C (2007) An ant colony optimization for dynamic job scheduling in grid environment. J Comput Inform Sci Eng 1(4):207–214
25. 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), Chennai, India. IEEE Press, New York, pp 14–17
26. 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
27. Kolodziej J, Xhafa F (2012) Integration of task abortion and security requirements in GA-based meta-heuristics for independent batch grid scheduling. Comput Math Appl 63:350–364
28. Kolodziej J, Xhafa F (2011) Meeting security and user behaviour requirements in grid scheduling, simulation modelling practice and theory. Int J Fed Eur Simul Soc 19:213–226
29. Kolodziej J, Xhafa F (2011) Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population. Future Gener Comput Syst 27(8):1035–1046
30. Nudd G, Kerbyson D, Papaefstathiou E, Perry S, Harper J, Wilcox D (2000) Pace—a toolset for the performance prediction of parallel and distributed systems. Int J High Perform Comput Appl 14(3):228–251
31. Smith W, Foster I, Taylor V (1998) Predicting application run times using historical information. In: Proceedings of IPPS/SPDP'98 workshop on job scheduling strategies for parallel processing, FL, USA
32. Hotovy S (1996) Workload evolution on the Cornell theory center IBM SP2. In: Proceeding of job scheduling strategies for parallel processing workshop, pp 27–40
33. Schulzrinne H, Tschofenig H, Morris J, Cuellar J, Polk J, Rosenberg J (2007) Common policy: a document format for expressing privacy preferences. RFC 4745