Chapter 5

Verification and Validation of the Proposed Solution

The previous chapter discussed in detail the problem formulation of resource scheduling and the design of the proposed Bacterial Foraging based Hyper-heuristic resource scheduling algorithm.

This chapter focuses on the verification and validation of the proposed framework. Formal verification of QoS based resource provisioning policies has been done through Z formal specification language. The proposed resource scheduling algorithm has been implemented in GridSim toolkit. Statistical analysis of simulation output has been performed in order to assess the accuracy of the estimated performance indices by using coefficient of variation. The framework has also been validated from various other perspectives similar to the existing frameworks like Dragon, GARA, Intra Grid etc.

At the outset, this chapter outlines the details of verification and the implementation of the proposed work through Z formal specification language. Then the experimental results obtained through GridSim have been discussed from various perspectives such as effect of resource heterogeneity, number of applications, number of resources, etc. Finally, the framework has been validated.

5.1 Verification of Resource Provisioning and Scheduling Framework

Formal specification can serve as a single, reliable reference point for those who investigate the customer's needs, those who implement programs to satisfy those needs and those who test the results [203]. Resource provisioning and scheduling framework and its policy (stated in chapter 3)can be verified using a formal language like Z specification. In Z [203][204], schemas are used to describe both static and dynamic aspects of a system. Z decomposes specifications into manageably sized modules, called schemas: that are divided into 3 parts: (i) a state, (ii) a collection of state variables and their values, (iii) operations that can change its state.

Resource provisioning policies deal with the resources and the consumers. In cost, time, security and reliability based resource provisioning policy, the resources are provisioned to those consumers whose requirements such as *cost*, *deadline*, *security* and *reliability* are fulfilled by the resource provisioning manager. The set of all resources and consumers are introduced as *basic types* of the specification:

$$[RESOURCE, CONSUMER].$$

The first aspect of the system to describe is its *state space*:

ResourceProvisioningPolicy

$$list : \mathbb{P}\, RESOURCE$$
$$provision : RESOURCE \nrightarrow CONSUMER$$

$$list = rajniprovision$$

In this schema, list and provision are variables. A relationship between the values of the variables has also been established.

- *list* is the set of resources recorded for allocation;
- *provision* is a function which when applied to certain resources, gives the user associated with them who fulfills all the conditions of the policy.

For example:

$$list = \{ \text{Resource1}, \text{Resource2}, \text{Resource3} \}$$
$$provision = \{ \text{Resource1} \mapsto \text{consumer 2}, \text{Resource2} \mapsto \text{consumer 3}, \text{Resource3} \mapsto \text{consumer1} \}.$$

In the schema Add Resource for Provisioning, the resources which are not in the list, are added in Grid environment, for provisioning to users' applications. Resource manager selects the resource providers who have not submitted the resources to the Grid environment earlier.

AddResourceforProvisioning

$\Delta ResourceProvisioningPolicy$
$consume? : CONSUMER$
$resource? : RESOURCE$

$resource? \notin list$

$provision' = provision \cup \{resource? \mapsto consumer?\}$

$list' = list \cup \{resource?\}.$

In Find Resource for Provisioning schema, resource manager finds the resources for provisioning the execution of users' applications.

FindResourceforProvisioning

$\Xi ResourceProvisioningPolicy$
$resource? : RESOURCE$
$consumer! : CONSUMER$

$resource? \in list$

$consumer! = provision(resource?)$

$list' = list$

$provision' = provision$

In this section, test cases for resource provisioning have been considered which are as follows:
Case 1: Initial state of the resource provisioning policy

InitResourceProvisioningPolicy

ResourceProvisioningPolicy

$list =$

This schema describes resource provisioning policy in initial state when the set *list* is empty: in consequence, the function *provision* is empty too.

Case 2: Resource provisioning operation is successful

$REPORT ::= ok \mid already\_in\ list \mid not\_in\ list.$

Success

$result! : REPORT$

$result! = ok$

This case shows a correct implementation of QoS based resource provisioning policy. It faithfully records resources, provisions them and gives the list of resource consumers associated with them which fulfill conditions of CRPP, TRPP, SRPP and RRPP policies. Following cases result in errors:

- Resource provider tries to add a resource already in the list of the Grid environment.

- Resource manager tries to find the resource for provisioning which is not in the list.

Case 2.1: Add same resource for provisioning in the list

AlreadyList

$\Xi ResourceProvisioningPolicy$
$resource? : RESOURCE$
$result! : REPORT$

$resource? \in list$
$result! = already\_list$

In this case, when the resource providers try to add the resources which are already in the list of resources, the schema shows this result.

Case 2.2: Find that resource which is not in the list

NotinList

$\Xi ResourceProvisioningPolicy$
$resource? : RESOURCE$
$result! : REPORT$

$resource? \notin list$
$result! = not\_list$

Now, the case in which the resource manager tries to find that resource which is not in the list of resource allocation is shown. Detail of resources has been shown in Table 5.1. Array of resources and consumers are:

$$resources : \quad ARRAY[1\,..\,]RESOURCE;$$
$$consumers : \quad ARRAY[1\,..\,]CONSUMER;$$

These 'infinite' arrays are taken for the sake of simplicity of Grid environment. In a real Grid environment, the schema is used to calculate and specify a limit on the number of entries of resources.

$\mathbb{N}_1$ is strictly positive integer to *Resource* or *Consumer*:

$$resources : \mathbb{N}_1 \rightarrow RESOURCE$$
$$consumer : \mathbb{N}_1 \rightarrow CONSUMER.$$

The element $resource[i]$ of the array is simply the value $resources(i)$ of the function, and the assignment $resources[i] := v$ is exactly described by the specification

$$resources' = resources \oplus \{i \mapsto v\}.$$

The right-hand side of this equation is a function which takes the same value as *resources* everywhere except at the argument i, where it takes the value v.

*ResourceProvisioningPolicy*1

$$resource : \mathbb{N}_1 \rightarrow RESOURCE$$
$$consumer : \mathbb{N}_1 \rightarrow CONSUMER$$
$$hwm : \mathbb{N}$$

$$\forall i, j : 1 \,..\, hwm \bullet$$
$$\quad i \neq j \Rightarrow resources(i) \neq resources(j)$$

The state space of the program of resource provisioning as a schema is described. There is another variable *hwm* (for 'high water mark') which shows the number of arrays in use. The idea of this representation is that each resource is linked with the consumers in the corresponding element of the array *consumers*. This can be documented with a schema *Abs* that defines the *abstraction relation* between the abstract state space *ResourceProvisioningPolicy* and the concrete state space *ResourceProvisioningPolicy*1:

Abs

$$ResourceProvisioningPolicy$$
$$ResourceProvisioningPolicy1$$

$$list = \{\, i : 1 \,..\, hwm \bullet resources(i) \,\}$$

$$\forall i : 1 \,..\, hwm \bullet$$
$$\quad provision(resources(i)) = consumers(i)$$

Here, Add Resource for Provisioning1 as concrete state is defined.

*AddResourceForProvisioning*1

$\Delta ResourceProvisioningPolicy1$
$resource? : RESOURCE$
$consumer? : CONSUMER$

$\forall i : 1 \,..\, hwm \bullet resources? \neq resources(i)$

$hwm' = hwm + 1$
$resources' = resources \oplus \{hwm' \mapsto resource?\}$
$consumers' = consumers \oplus \{hwm' \mapsto consumer?\}$

This schema describes an operation which has the same inputs and outputs as *AddResourceforprovisioning*, but operates on the concrete instead of the abstract state. It is a correct implementation of *AddResourcefor provisioning*, because of the fact:

- Whenever *AddResourceforprovisioning* is legal in some abstract state, the implementation *AddResourcefor provisioning*1 is legal in any corresponding concrete state.

Precondition: Add Resource Provisioning in List
The operation *AddResourceforProvisioning* is legal if its precondition $resource? \notin list$ is exactly satisfied. If this is so, the predicate

$$list = \{ i : 1 \,..\, hwm \bullet resources(i) \}$$

from *Abs* tells us that *resource*? is not one of the element $resources(i)$:

$$\forall i : 1 \,..\, hwm \bullet resource? \neq resources(i).$$

This is the pre-condition of *AddResourceforProvisioning*1.

The second operation, *FindResourceforProvisioning*, is implemented by the following operation, again described in terms of the concrete state:

*FindResourceforProvisioning*1

Ξ*ResourceforProvisioning*1

resource? : *RESOURCE*

consumer! : *CONSUMER*

$\exists\, i : 1 \,..\, hwm \bullet$

$\quad resource? = resources(i) \land consumer! = consumers(i)$

The predicate indicates that there is an index i at which the *resources* array contains the input *resource*?, and the output *consumer*! is the corresponding element of the array *consumers*.

Precondition: Find Resource for Provisioning

For this schema to be possible, *resource*? must in fact appear somewhere in the array *resources*: this is the pre-condition of the operation. The pre-conditions of the abstract and concrete operations are in fact the same: i.e the input *resource*? is known. The output is correct because for some i, $resource? = resources(i)$ and $consumer! = consumers(i)$, so

$$
\begin{aligned}
& consumer! = consumers(i) \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{[spec. of } \mathit{FindResourceforProvisioning}1\text{]} \\
& \quad = provision(resources(i)) \qquad\qquad\qquad\qquad\qquad \text{[from } \mathit{Abs}\text{]} \\
& \quad = provision(resource?). \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{[spec. of } \mathit{FindResourceforProvisioning}1\text{]}
\end{aligned}
$$

The existential quantifier $\exists$ i: in the description of *FindResourceforProvisioning*1 leads to a loop in the program code, searching for a suitable value of i:

$$
\begin{aligned}
& \mathit{FindResourceforProvisioning}(resource : RESOURCE; \\
& consumer : CONSUMER); \\
& \quad i : INTEGER; \\
& \quad i := 1; \\
& \quad resources[i] \neq resourcei := i + 1; \\
& \quad consumer := consumers[i] \\
& ;
\end{aligned}
$$

The idea of this representation is that the resource providers give the facility of resource provisioning to the user for optimum results, better services and avoid the violations of the service level guarantees. The implementation of this policy enables the users to analyze customer requirements and define processes that contribute to the achievement of a product or service that is acceptable to their resource consumers.

5.2 Simulation Model: GridSim Toolkit

Due to the inherent heterogeneity of Grid computing environment, it is difficult to evaluate the performance of the proposed schema in a controlled manner. The simulation model enables us to perform repeatable experiments and the cost incurred by performing experiments on real infrastructure would be prohibitively expensive. In addition, Grid testbed is limited to a few resources and domains. Thus for experimental results, GridSim toolkit [205] has been used. GridSim toolkit provides facilities for modeling and simulation of resources and network connectivity with different capabilities, configurations and domain. It also supports primitives for application composition, information services for resource discovery and interfaces for assigning application tasks to resources and managing their execution. The following are the reasons for the GridSim toolkit to be used for evaluation [205].

- It allows modeling of heterogeneous resources.
- Resources capability can be defined in the form of Millions instructions Per Second (MIPS) as per Standard Performance Evaluation Corporation (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.
- Statistics of all or selected operations can be recorded and they can be analyzed using GridSim statistics analysis methods.
- It supports simulation of both static and dynamic schedulers.
- Application tasks can be heterogeneous and they can be CPU or I/O intensive.

For experimental results, heterogeneous resources have been considered. In general, each resource may contain a different number of machines, and each

machine may have one or more than one processing element with different MIPS. In results, it has been assumed that each application/task which is submitted to the Grid may require varying processing time and input size and such type of tasks are 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, the size of input/output files, etc. The processing requirement of Gridlets is measured in Multiple Instructions (MI).

5.2.1 Performance Metrics

In this section, a performance evaluation criteria to evaluate the performance of a resource provisioning and scheduling framework has been defined. Two matrices, namely submission burst and cost for evaluating the performance of QoS parameter(s) based resource provisioning policy have been selected. The former indicates the total time obtained since the start of Gridlet's waiting in the queue till the resources are provisioned where as the latter indicates, the cost per unit resources that are consumed by the users for execution of their applications. The submission burst and cost are measured in seconds and Grid dollars (G$) (in-built units) respectively.

5.2.2 Experimental Results

To validate the QoS based resource provisioning approach, 250 jobs and 75 resources have been considered. An average of thirty to fifty runs have been considered in order to guarantee statistical correctness. In addition, a comparison of submission burst with QoS enabled resource provisioning vs QoS disabled resource provisioning has been presented. In order to evaluate the performance of the proposed approach, the effects of different values for the parameters of QoS based and non-QoS based resource provisioning have been investigated. In all the experiments, a comparison of QoS based resource provisioning with non-QoS based resource provisioning has been done.

Test Case 1: In the first test case, the submission burst time and cost of Grid applications have been evaluated 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 submission burst time and cost of an application may conflict with each other depending on the price of the resources. Figure 5.1, 5.2 show the submission

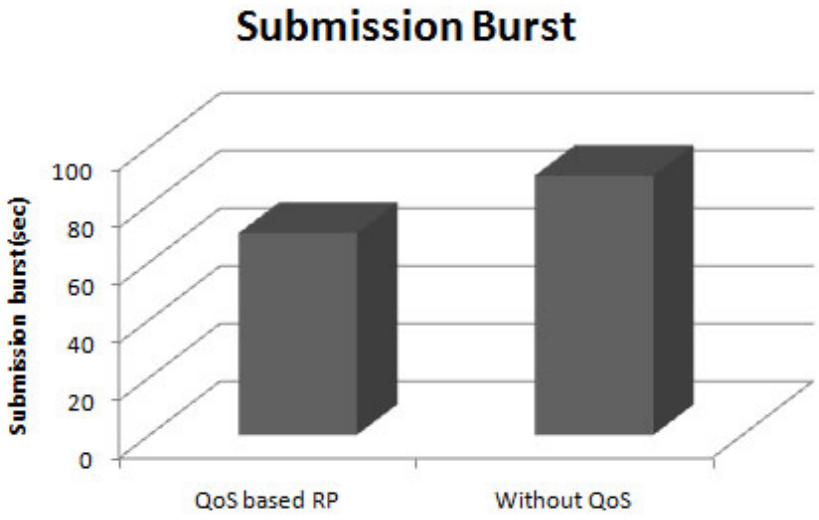

Figure 5.1: Comparison of the submission burst of QoS based resource provisioning QoS vs without QoS

burst and cost of QoS based resource provisioning vs non-QoS based resource provisioning respectively. The results show that in case of non-QoS based resource provisioning, if the same number of applications/jobs are sent to the Grid, submission burst and cost increases while in the other case, they decreases. This is expected as the QoS based resource provisioning approach is keeping track of the state of all resources at each point of the time which enables it to take an optimal decision than non-QoS based resource provisioning approach.

Experiment to determine the effect of increasing the number of applications on the cost and submission burst has also been performed. From the experimental results shown in Figures 5.3, 5.4, it can be concluded that the time taken to submit an application reduces by using QoS based resource provisioning approach. Figure 5.4 shows that cost per application increases as the number of submitted applications increases. Non-QoS based resource provisioning resulted in a schedule which is expensive in comparison to QoS based resource provisioning approach as the number of applications increases. The reason is that non-QoS based resource provisioning approach does not consider the effect of other applications in the meta scheduler at the time of job submission but in QoS based resource provisioning approach, RPM considers the effect of applications in meta scheduler before execution of application according to both user and resource providers's perspectives. In resource provisioning and scheduling framework, RPM considers the cost, time and other parameters of other applications before job submission

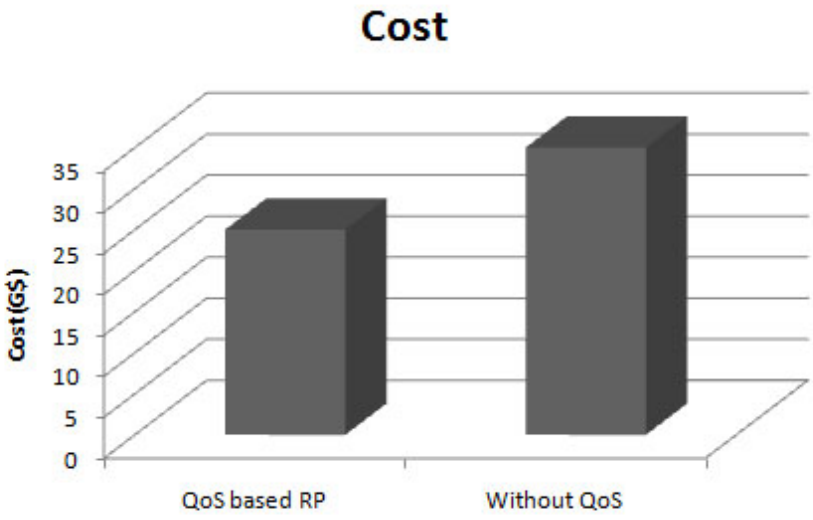

Figure 5.2: Comparison of the cost of QoS based resource provisioning vs without QoS

to meta-scheduler.

Test Case 2: The second test case evaluates the submission burst time and cost for job submission based on QoS based provisioned approach and best-effort approach. Best-effort approach does not support QoS and same priorities for all users and also there is no assurance about resource provisioning. Figures 5.5, 5.6 show the submission burst and cost of the application's execution using the best-effort and QoS based provision approach at different resource utilization levels. At 50% resource utilization level, the best effort submission burst id is 10 to 15% higher than QoS based provision approach as shown in Fig 5.5. This time variation with resource utilization is quite significant. The best-effort submission burst time sharply increases as the resource utilization reaches at 80% but the provisioned approach based submission burst time is less. The cost of the best-effort and provisioned approach differs less significantly in comparison to submission time at the resource utilization levels. This is because when the number of applications is large, they are scheduled on left out resources which may not be very cost effective. The cost of resources may or may not vary with resource's configuration.

From the above results, it has been observed that application's execution using the proposed QoS based resource provisioning approach provides the following advantages: The provisioned based submission burst time is 60% lower than the best effort approach. The time variation in execution of applications is about 5-10 %, compared to non QoS based resource provisioning of 50- 60 % using the same

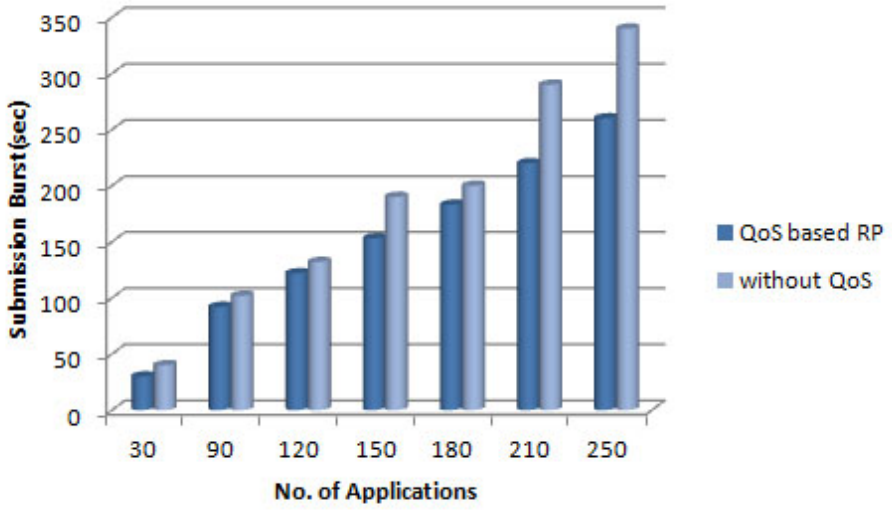

Figure 5.3: Effect of change in number of application submitted on submission burst

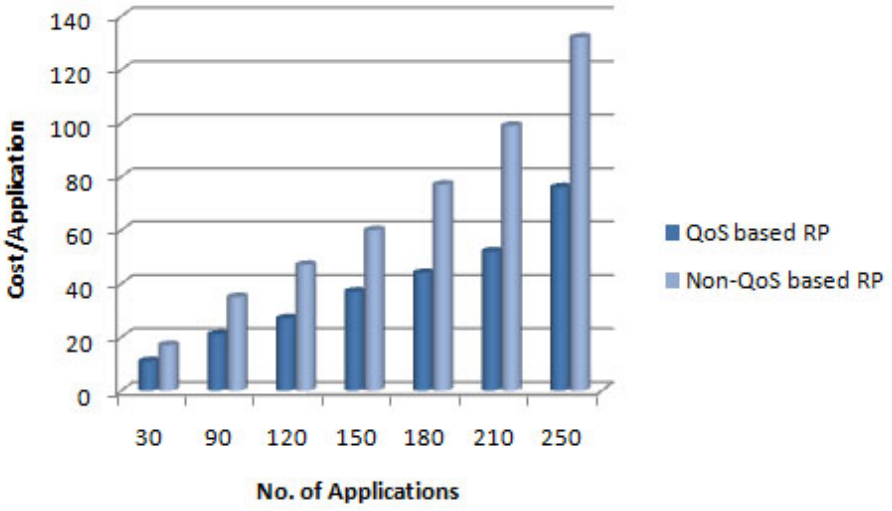

Figure 5.4: Effect of change in number of application submitted on cost

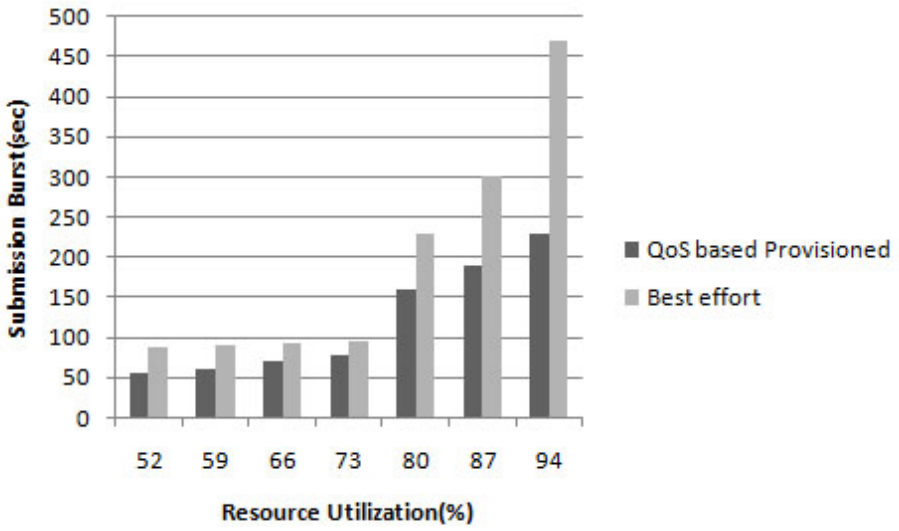

Figure 5.5: Submission burst of best effort and QoS based provisioned approach with resource utilization

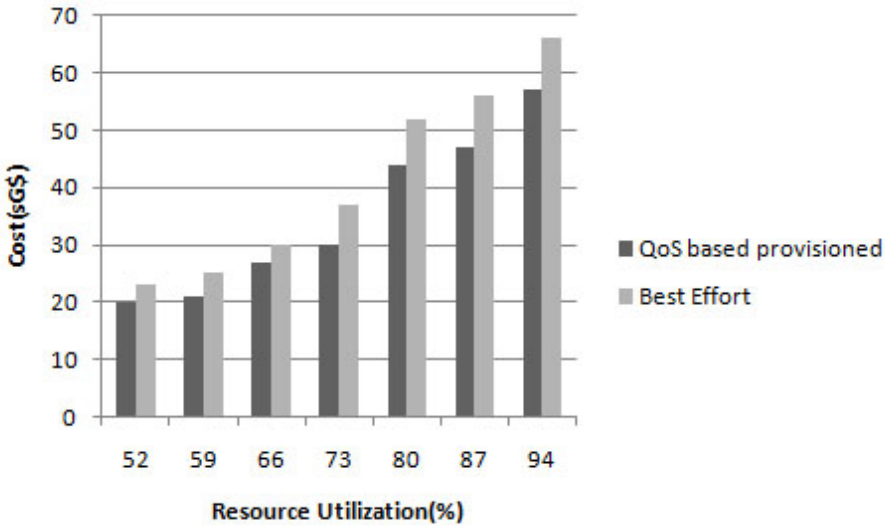

Figure 5.6: Cost of best effort and QoS based provisioned approach with resource utilization

set of applications. This time variation is quite significant. It also maintains the resource utilization and cost for user's application execution.

5.3 Experimental Scenario

The evaluation of the strategies has been performed using GridSim. Table 5.1 shows the characteristics of resources and Gridlets, which have been used for all the experiments. Table 5.2 below provides the proposed scheduling algorithm parameters names and their values taken in the experiments. User applications have been modeled as independent parallel applications which are computation intensive. Thus the data dependency among the tasks in the parallel applications is negligible. Each task is parallel and is hence considered to be independent of any other subtask. For evaluation, a suitable workload from real machine traces have been derived. These traces have been obtained from Grid workload archive website [1]. 5000 user applications are generated according to the Lublin workload model [206]. 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, an ETC matrix has been generated which is computed as the ratio of workload and computing capacity of machine vectors. No of jobs * 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. Each job can execute on each resource, and the estimated execution times of each job on each resource is 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 [197]. 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

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

Table 5.1: Scheduling parameters and their values

Parameter	Value
Number of Resource	150-250
Number of Gridlets	5000
Length of Job	1000 - 6000
Bandwidth	3000 or 7000 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$-5G$
File size	100 + (10-30%) MB
Job Output size	250 + (10-40%)MB

task execution time and Coefficient of Variation (CV) are used to generate ETC matrices [197]. 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 α=0.3 and β=0.7.

Table 5.2: BFO parameters and its values

Parameter	Value
Population size	100
Chemotaxis steps	20
Swim length	4
Reproduction steps	4
Ci (step size)	0.05
Elimination and dispersal with probability	.25

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 Figure 5.7. The instance contains: (a) workload (b) computing capacity of machines; (c) prior load of machines and (d) the ETC

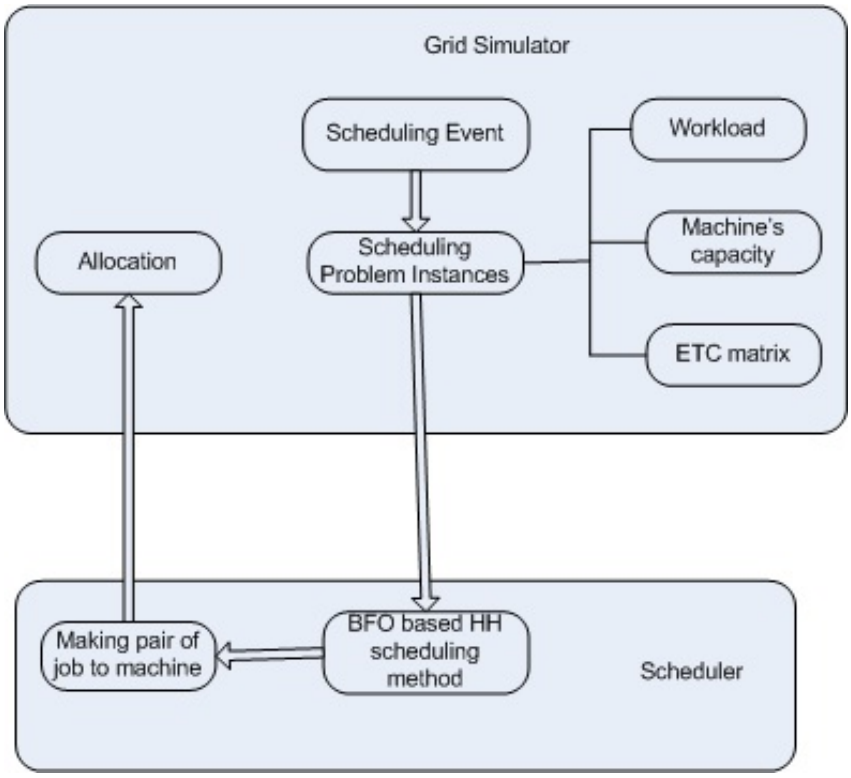

Figure 5.7: Flowchart of scheduling in Simulator

matrix. The defined instance is then passed on to the scheduler which computes the planning of jobs to resources.

5.3.1 Performance Evaluation Criteria

In this section, the performance evaluation criteria has been defined to evaluate the performance of bacterial foraging based hyper-heuristic for resource scheduling. Two matrices, namely makespan and cost have been selected and calculated using the equations 4.2 and 4.3 (described in Chapter 3) 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.

5.4 Analysis of Results

To validate the proposed algorithm, 5000 jobs/applications and 150 - 250 resources have been considered. An average of fifty runs has been used in order to guarantee

statistical correctness. The simulation results have been presented using Ali et al. [197] simulation model with the GridSim discrete event simulation so as to test the performance of hyper-heuristic based algorithm. In addition, a comparison of makespan and cost of the proposed algorithm with existing heuristic algorithms such as GA, SA, GA-TS has been presented. In order to evaluate the performance of the proposed approach, the effects of different number of applications have been investigated . In all the experiments, a comparison for consistent and inconsistent matrix has been done.

5.4.1 Test case 1: Performance for the low Heterogeneous case

In this case, makespan and cost of the Grid applications for low resource/machine heterogeneity with inconsistent and consistent matrices have been evaluated. 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. The most important characteristic applicable to real-world scenarios is that how each algorithm responds to different heterogeneity of jobs and resources. A comparison of both cost and makespan for low resource/machine heterogeneity has been shown for consistent and inconsistent matrices.

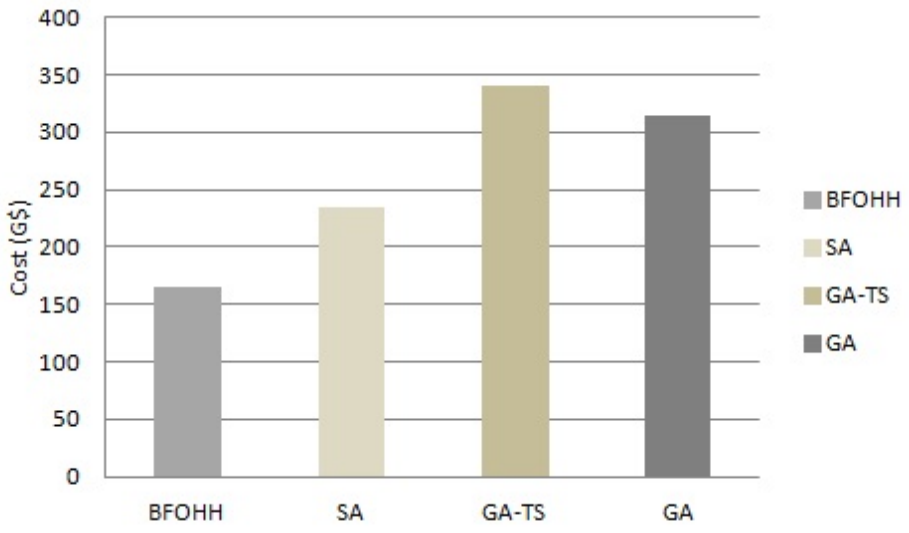

Figure 5.8: Cost comparison result for inconsistent and low machine heterogeneity

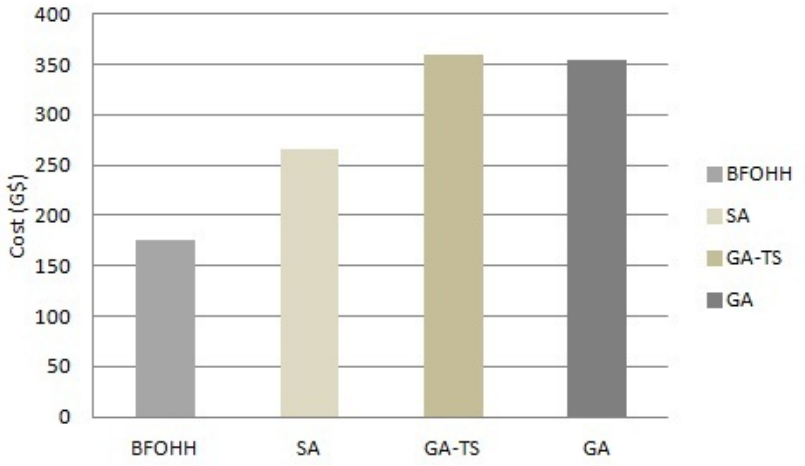

Figure 5.9: Cost comparison result for consistent and low machine heterogeneity

The low resource heterogeneity is simulated by resources having the number of PEs between 1 and 5. Figures 5.8 and 5.9 show the effect on cost by the four heuristics in case of low resource heterogeneity with inconsistent and consistent matrices. Figure 5.8 shows that the lowest cost was achieved in case of Bacterial Foraging based Hyper-heuristic (BFOHH) resource scheduling algorithm where as GA-TS resulted in the highest cost. A similar trend can be seen in case of low resource heterogeneity with consistent matrix. When having low resource heterogeneity, BFOHH outperforms all the other approaches for both consistent and inconsistent matrices. The other three heuristics have higher cost in comparison to BFOHH for application execution.

Figures 5.10 and 5.11 show the makespan comparison for inconsistent and consistent matrices with low machine heterogeneity. It can be seen from the figures that the makespan is lowest in case of BFOHH for both the consistent and inconsistent matrices. This is because of the low 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 BFOHH in managing the time requirement of the user. The results show that in case of GA, SA and GA-TS algorithms, if the same number of applications/jobs are sent to the Grid, makespan and cost increases whereas in the case of bacterial foraging based hyper-heuristic resource scheduling algorithm both makespan and cost decreases.

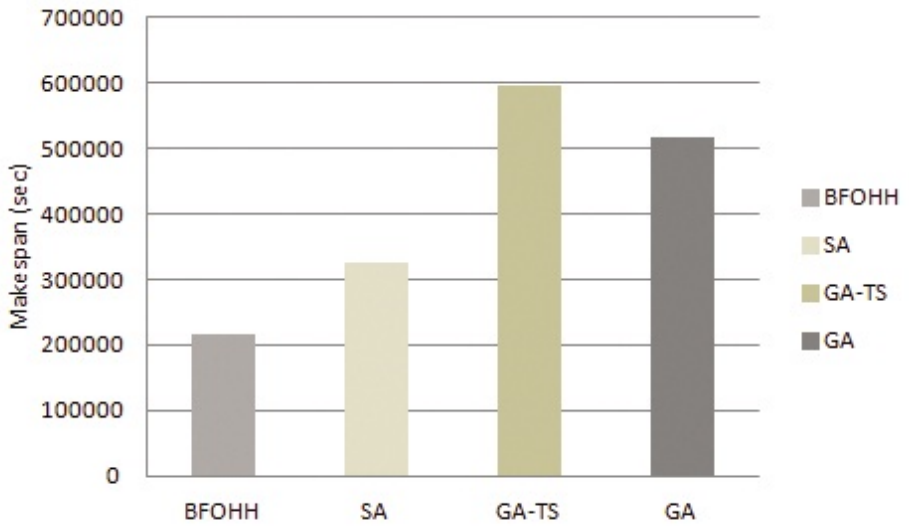

Figure 5.10: Makespan comparison result for inconsistent and low machine heterogeneity

5.4.2 Test case 2: Performance for the high heterogeneous case

In this case, makespan and cost of the Grid applications for high resource/machine heterogeneity case with inconsistent and consistent matrices have been evaluated. In this case, high resource heterogeneity is simulated by each resource having a random number of PEs between 7 and 30.

Figures 5.12 and 5.13 show the makespan comparison for inconsistent and consistent cases with high machine heterogeneity. As can be seen from Figures 5.12 and 5.13, the performance of BFO based hyper heuristic is far better than that of existing scheduling algorithms i.e GA, SA and GA-TS for the case of inconsistent and high machine heterogeneity. Figures 5.14 and 5.15 show the cost comparison for inconsistent and consistent ETC matrix with high machine heterogeneity. The cost reduction in GA was the least in comparison to other algorithms. It can be concluded from the figures that when the same number of applications are sent, the cost and makespan are the least for the proposed algorithm. This is because more applications are scheduled on cheaper and efficient resources, due to the ability of BFOHH to find optimal resources. By analyzing the results in the Figures 5.8 - 5.15, it can be concluded that BFO based hyper-heuristic outperforms all the other approaches in the cases with both low or high machine heterogeneity.

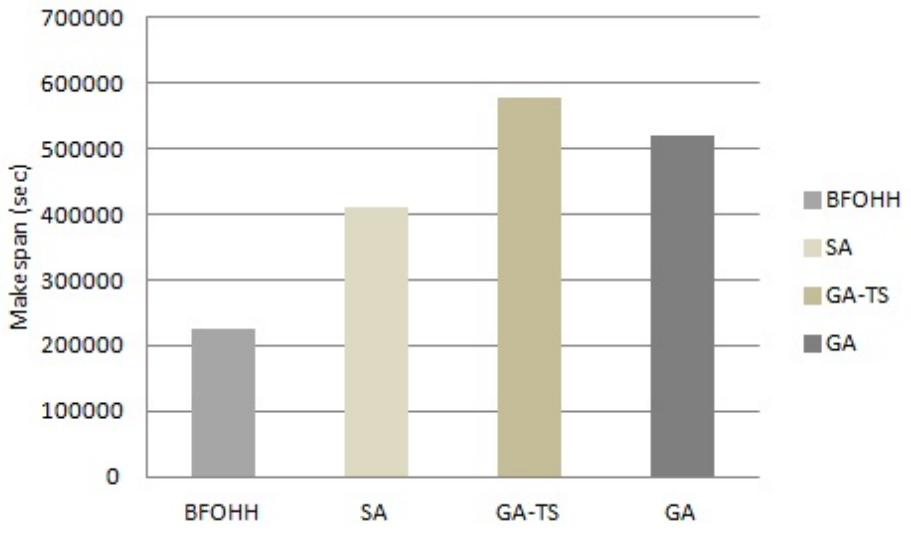

Figure 5.11: Makespan comparison result for inconsistent and low machine heterogeneity

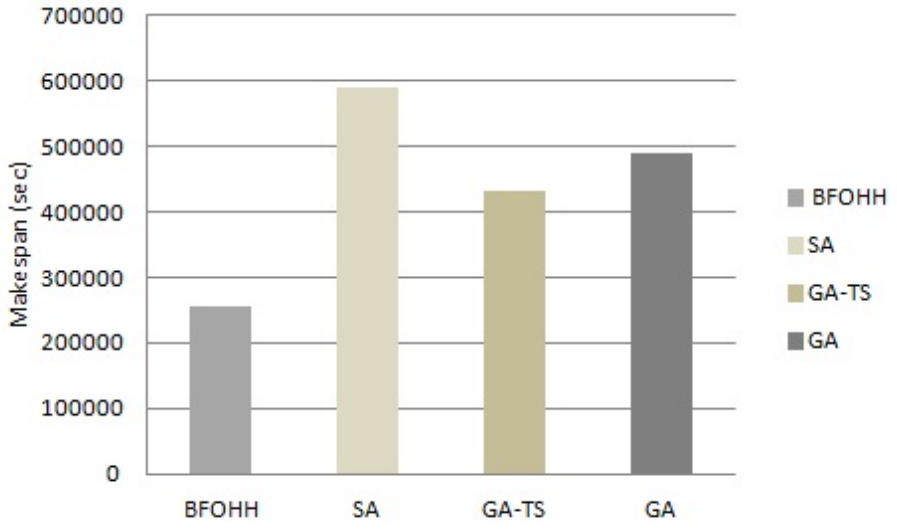

Figure 5.12: Makespan comparison result for inconsistent and high machine heterogeneity

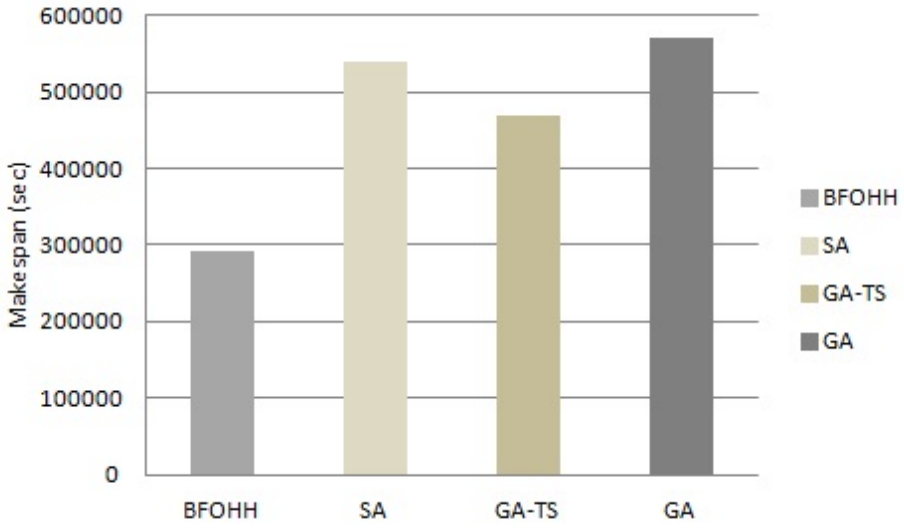

Figure 5.13: Makespan comparison result for consistent and high machine heterogeneity

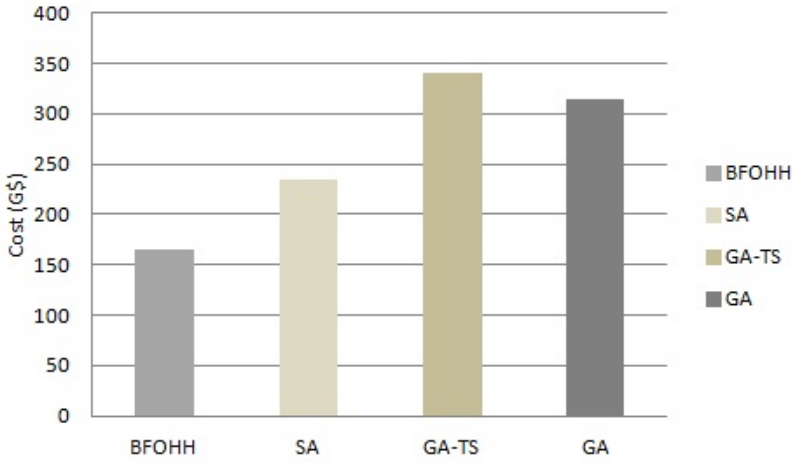

Figure 5.14: Cost comparison result for inconsistent and high machine heterogeneity

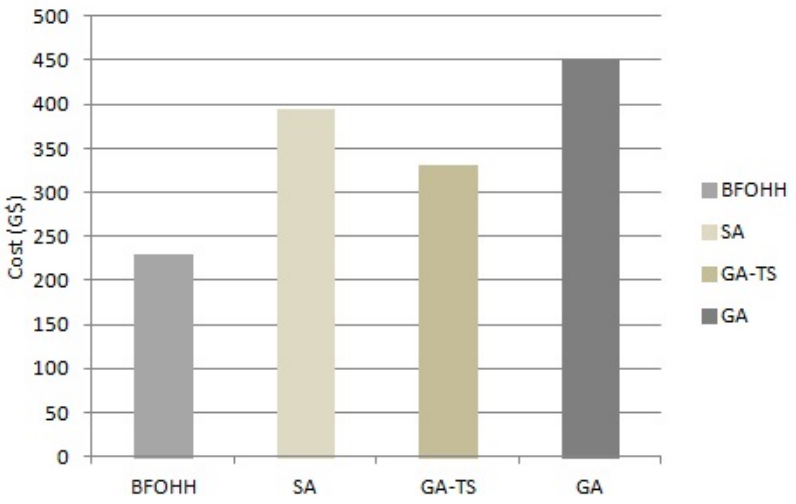

Figure 5.15: Cost comparison result for consistent and high machine heterogeneity

5.4.3 Test case 3: Effect of the number of resources and its capacity

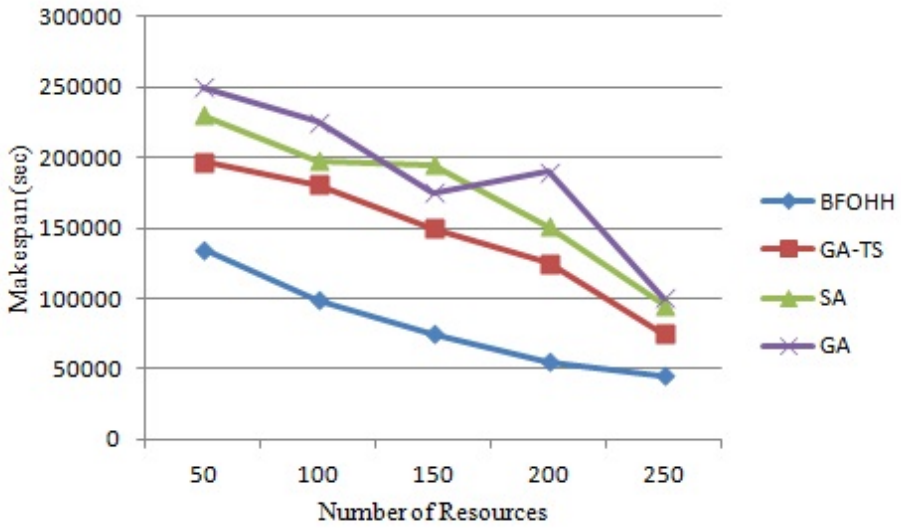

Figure 5.16: Effect of the number of resource on the makespan

Figure 5.16 shows the effect of increasing the number of resources, while keeping the number of jobs being sent to the Grid constant. In this experiment, 1000 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. BFOHH and GA-TS performs better

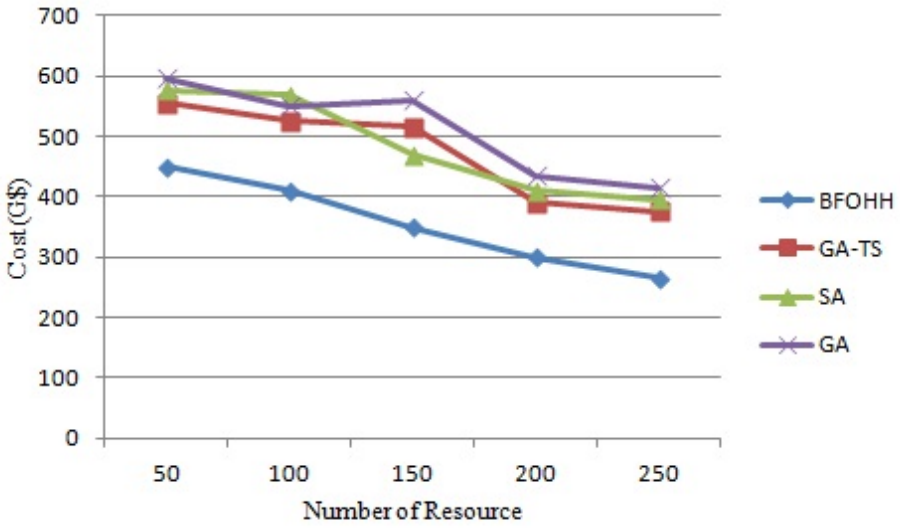

Figure 5.17: Effect of the number of resource on the cost

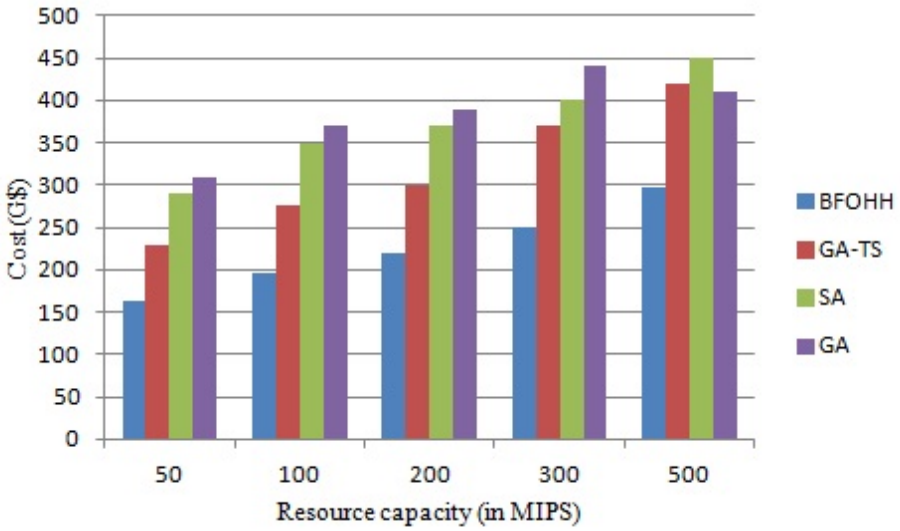

Figure 5.18: Effect of the resource capacity on the cost

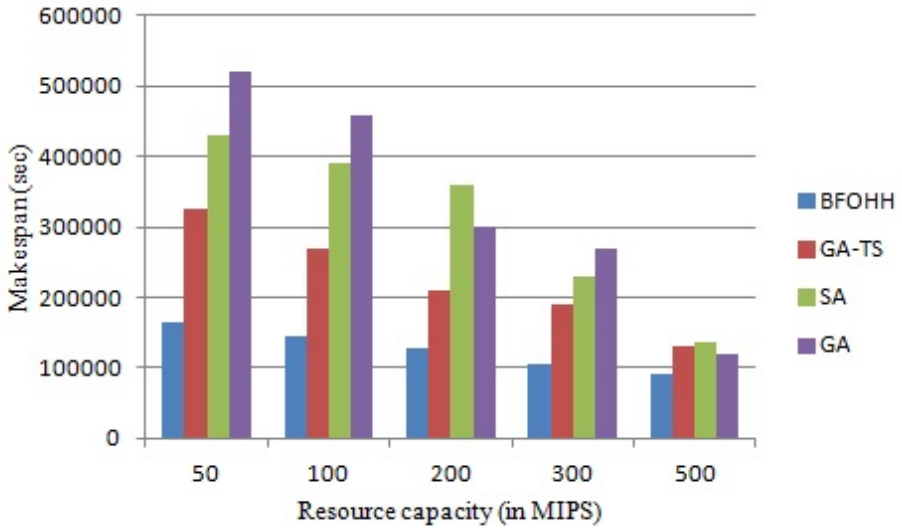

Figure 5.19: Effect of the resource capacity on the makespan

when the number of resources is less in comparison to the number of jobs. As shown in Figure 5.16, the makespan time decreases for all the algorithms in the same proportion on an increase in the number of resources. This observation indicates that our algorithm gives an equally good performance in comparison to that given by other algorithms.

The cost of application execution using BFOHH is much less in comparison to the execution cost using existing scheduling algorithms. 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. Figures 5.18 and 5.19 show cost and makespan of the user application by varying the capacity of the resources respectively. In this case, the cost of the resource in terms of its speed has been considered. An increase in the capacity of the resource will also increase its cost but with a reduction in the execution time. Figure 5.19 depicts that an increase in the resource's capacity will decrease the makespan thus increasing the overall performance of our algorithm in comparison to GA, SA and GA-TS. 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.

5.4.4 Test case 4: Effect of the number of jobs

Experiments have also been performed to determine the effect of an increase in the number of applications on cost and makespan. A hundred-node simulated

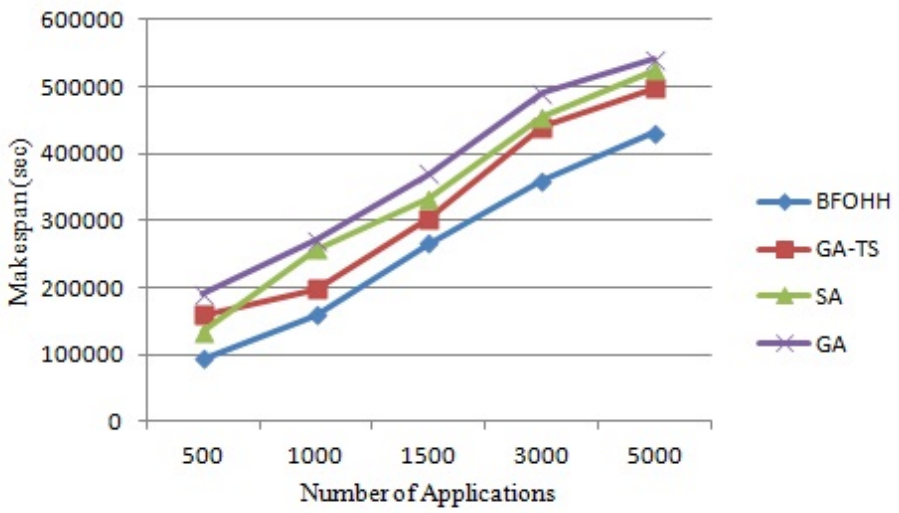

Figure 5.20: Effect of the number of applications on the makespan

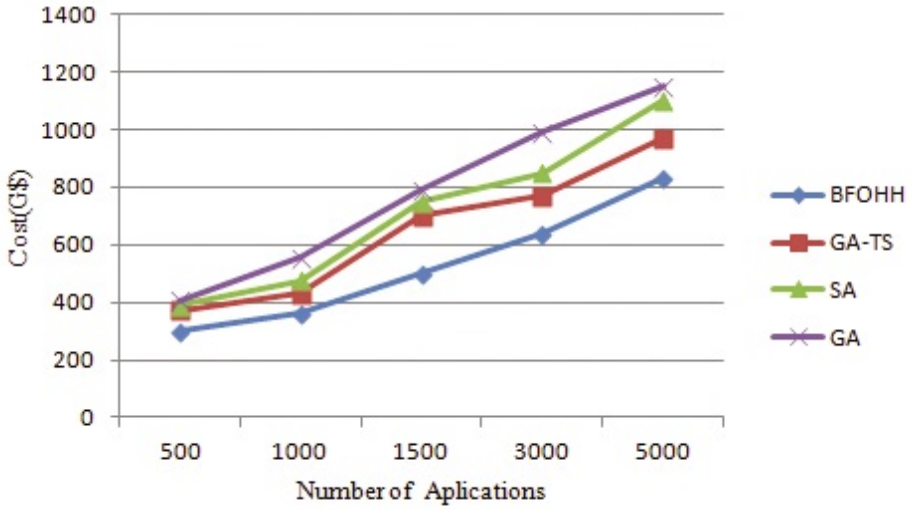

Figure 5.21: Effect of the number of applications on the cost

Grid with two thousand jobs being sent to the Grid has been considered. From the experimental results shown in Figure 5.20, it can be concluded that the time taken to execute an application reduces by using BFO based hyper-heuristic algorithm. Figure 5.21 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 BFO based hyper-heuristic algorithm. From all the experimental results, it was observed that application's execution using BFO 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 applications execution is about 5-11 %, which is much less in comparison to the 50-70 % variation in the existing algorithms using the same set of applications. The overall cost for user's application execution is less.

5.4.5 Statistical Analysis of Results

In this section, statistical method, namely the Coefficient of Variation (CV), has been employed 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, coefficient of variation is used instead of the standard deviation. It expresses the variation of the data as a percentage of its mean value, and is calculated as follows:

$$CV = (standard\ deviation/mean) * 100 \tag{5.1}$$

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 Figures 5.22 and 5.23, the CV of the makepsan and cost of application's execution of each scheduling algorithm has been thoroughly examined.

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. As 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 application changes. As the number of applications increases, the CV value decreases. It shows that our proposed algorithm outperforms other existing algorithms for large number of applications. As, the system

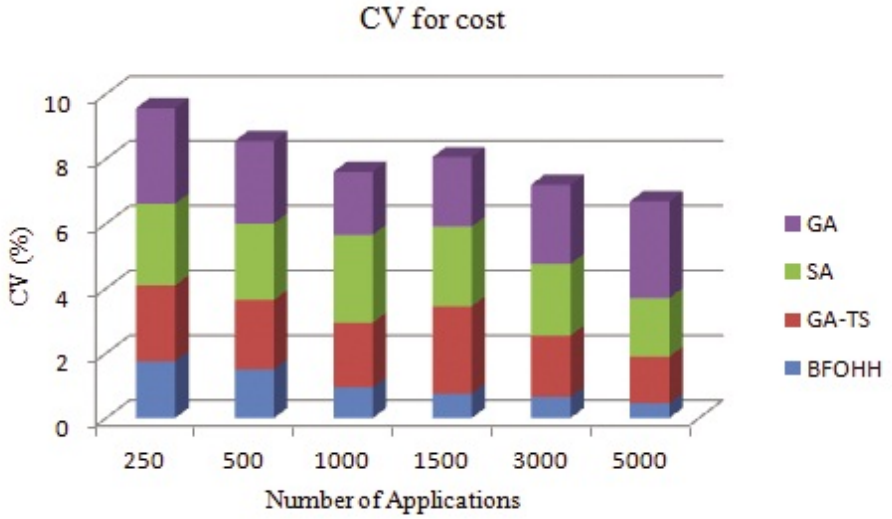

Figure 5.22: Coefficient of variation for the cost with each algorithm

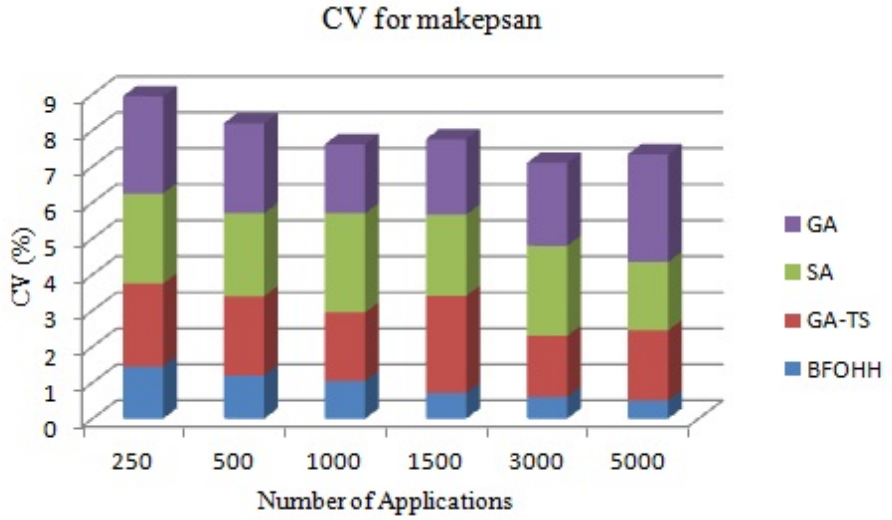

Figure 5.23: Coefficient of variation for the makespan with each algorithm

with small coefficient of variation value is more balanced, so the proposed algorithm achieved the best results in the Grid with regard to both makespan and cost as QoS parameters.

5.5 Framework Validation

In the proposed framework, resource scheduling has been done on the basis of the QoS parameter(s) based resource provisioning policy which was not considered traditionally. Figures (in experimental results section) show that QoS based provisioned approach generates better results and resource provisioning based scheduling algorithm is able to schedule the job on the ingredient resources more efficiently. Resource provisioning and scheduling framework has been validated against the features of the existing resource provisioning and scheduling frameworks and its result has been depicted in Table 5.3.

Table 5.3: Comparison of Resource Provisioning and Scheduling Framework with existing frameworks

Framework /Features	**RPSF**	**Dragon**	**Glare**	**GARA**	**QoS-GRAF**	**RAA/Falkon**	**Intra Grid**	**Ali's Framework**
Usage	Resource provisioning and Scheduling 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 enable the deployment of applications across multiple Grids.	It worked before resource provisioning to discover resources.
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 & Deallocation 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

5.6 Conclusion

In this chapter verification details, experimental setup and testing results of the proposed work have been demonstrated. Formal specifications of resource provisioning policies have ensured that the customer gets resources as per the desire and the scheduling process is more clear. The experimental results have been illustrated for resource provisioning and resource scheduling. The performance of the BFO based hyper-heuristic algorithm has been compared with well-known scheduling algorithms such as GA, SA and GA-TS. The performance of the proposed algorithm with variation in both the number of jobs and the number of resources have been analyzed, which are expected to vary in the real Grid environment. The algorithm's performance has been evaluated with respect to both 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 it also helps to satisfy the user's requirements. The testing results demonstrate that the proposed solutions are working and can be effectively used to address resource provisioning and scheduling challenges to establish an efficient Grid.

The next chapter concludes the thesis and also discusses the future directions.