Chapter 4

Resource Scheduling Algorithm

The previous chapter discussed in detail the design of Resource Provisioning and Scheduling Framework and QoS parameter(s) based Resource Provisioning Policies.

In this chapter, a resource scheduling algorithm has been designed considering the QoS expectations of both, the resource providers and resource consumers. As hyper-heuristic is more pliable for Grid environment therefore a novel Bacterial Foraging Optimization (BFO) based hyper-heuristic resource scheduling algorithm has been proposed. This algorithm effectively schedules the jobs on available resources in a Grid environment and simultaneously minimizes the cost and the makespan. QoS parameter(s) based resource provisioning policies that have been discussed in the previous chapter are the basis of the proposed algorithm.

This chapter initially presents the Grid resource scheduling model. Then, the resource scheduling problem as a combinatorial optimization problem has been formulated that minimizes the cost and makespan. It then moves to the design of BFO based hyper-heuristic for resource scheduling algorithm. Finally, the last section provides the detailed description of the proposed algorithm.

4.1 Resource Scheduling Model

Grid scheduling is a two-step process. In step one, the required set of resources is identified as per the user's requests and in the second step, the jobs are mapped on to the actual set of resources thus further ensuring near optimal satisfaction of QoS parameters. For example, suppose that a person has to buy something from a shop. The shopkeeper would ask the buyer about his budget and then he will show the items accordingly. The buyer will now select the most appropriate items amongst all the items shown that would be an exact match for his cost and other specifications.

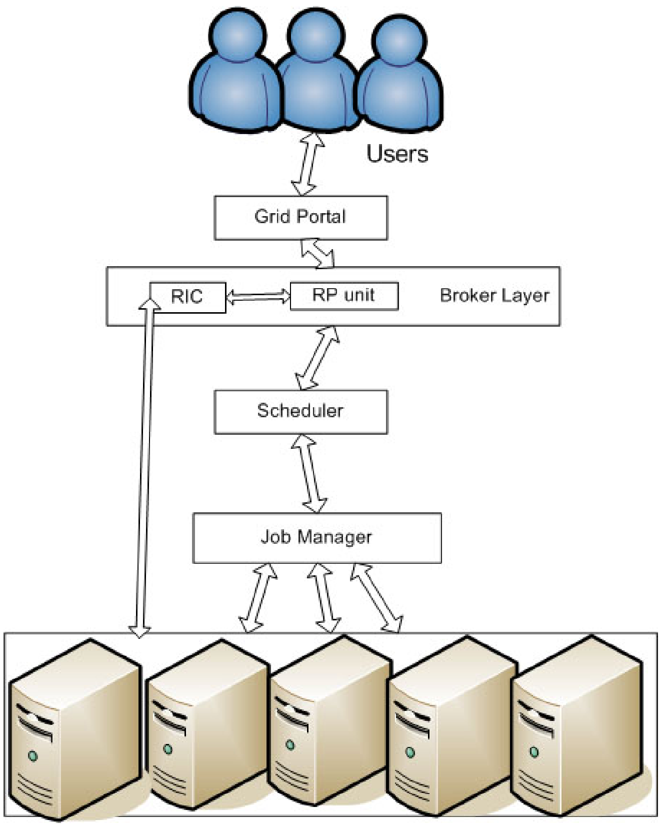

Figure 4.1: Resource Scheduling Model

Figure 4.1 shows the Grid resource scheduling model (elaborated from the proposed framework that has been discussed in the previous chapter).

- Each user tries to access the resources for application's execution through Grid portal. After this, the task of authorization and authentication is performed at the portal side.
- Resource Provisioning (RP) unit takes the information about the available resources from RIC. RIC collects all the information about the resource

provider's resources. Then, the RP unit performs preliminary provisioning for user's requests as shown in Figure 4.1. On the completion of resource provisioning, the task of mapping of jobs to exact resources is performed.

- Grid scheduler further takes all the information from RP unit who has a list of provisioned resources which are available in the Grid environment. According to the status of the resource, scheduler will consult with the Job Manger (JM) for job execution.

- Job Manager is a protocol engine for communicating with a range of different local resource schedulers using a standard message format [195]. Grid computing resources are typically operated under the control of a scheduler which implements allocation and prioritization policies while optimizing the execution of all submitted jobs for high efficiency and performance. Then the mapping of jobs to resources is performed. Resources can be clusters of computers, memory space, storage capacity, files and attached peripheral devices etc.

This section described the resource scheduling model. The next section would discuss the problem formulation.

4.2 Resource Scheduling: Problem Formulation

To map 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 requirements is a combinatorial optimization problem. The main goal of the Grid scheduler is to schedule the resources effectively and efficiently. The resources and applications/jobs can leave and join the Grid dynamically. Grid resources are heterogenous and dynamic in nature which makes the scheduling problem an NP-complete problem [98]. In this model, independent jobs have been considered to handle the realistic scenarios as there are many scenarios in which the need of scheduling of independent jobs arises. Firstly, this problem is suitable to Grid systems because of the nature of Grid users, who submit jobs or monolithic applications in an independent manner to the system. Secondly, Grid systems are most useful for massive parallel processing, in which large amounts of data are processed independently [196].

In this model, scheduling problem has been considered from both the user's and resource provider's point of view. User wants to minimize the cost whereas the resource provider wants to minimize the makespan. Makespan is used to indicate the general productivity of the Grid systems. Smaller values of makespan indicates that the scheduler is planning the jobs in an efficient manner. Cost is an another optimization criteria, which refers to the total cost of the job execution on a particle resource. In order to formalize the problem instance, the Ali et.al computational model has been used [197]. The problem has been mathematically formalized to get an optimal solution.

To consider this problem, a set of independent jobs $\{j_1, j_2, j_3, ...j_m\}$ to map on a set of heterogenous and dynamic resources $\{r_1, r_2, r_3.........r_n\}$ has been taken.

$R = \{r_k \mid 1 \leq k \leq n\}$ is the collection of resources and n is the total number of resources. $J = \{j_i \mid 1 \leq i \leq m\}$ is the collection of jobs and m is the total number of jobs.

The Expected Time To Compute (ETC) value of each application/ job on each resource is assumed to be given by the user-supplied 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 [198] and historical information[199] [200]. Under the ETC simulation model for problem formulation, the following constraints have been considered:

i. Each job to be scheduled for application's execution has a unique id.

ii. Jobs are independent.

iii. Arrival of jobs for execution of application is random and jobs are placed in a queue of unscheduled jobs.

iv. The computing capacity/speed of the resources is measured in Multiple Instruction Per Second (MIPS) as per Standard Performance Evaluation Corporation (SPEC) benchmark.

v. The processing requirement of job is measured in Million Instructions (MI).

vi. Execution time for every job on resource is obtained from the ETC matrix. 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.

A simple weighted sum function of makespan and cost has been used to deal with their simultaneous optimization.

4.2.1 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:

$$FitnessFunction = \theta cost + \delta makespan \tag{4.1}$$

$$cost = min(c(r_k, j_i)) \quad for \quad 1 \leq k \leq n, 1 \leq i \leq m \tag{4.2}$$

$$makespan = min(F_{j_i}) \quad for \quad j_i \in J \tag{4.3}$$

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 Equations 4.2 and 4.3 have been used for the definition of an objective function for the independent parallel job scheduling as defined in Equation 4.1.

$$c(r_k, j_i) = c_e(r_k, j_i) \tag{4.4}$$

$$c_e(r_k, j_i) = \sum_{j_i \in J} completion(j_i, r_k) \Big/ completion_{m(j_i)} \times J \tag{4.5}$$

Where as :

$$completion_{m(j_i)} = max_{j_i \in J, r_k \in R} completion(j_i, r_k) \tag{4.6}$$

Cost c(r_k, j_i) is the cost of job j_i which executes on resource r_k. Equation 4.5 denotes $c_e(r_k, j_i)$ user's application execution cost. In Equation 4.6, the $completion_{m(j_i)}$ denotes the maximal completion time of the users 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. Avail $\_time_{r_k}$ is the time in which the machine can complete the execution of all the previous assigned jobs. Completion time $completion(j_i, r_k)$ 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.

$$completion(j_i, r_k) = avail\_time_{r_k} \pm ETC(j_i, r_k) \tag{4.7}$$

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.

To provide the security and reliability as QoS parameters, makespan and cost have been calculated using the below mentioned equations. The objective function in this case is same as in Equation 4.1. but it includes the values of the $P_f(j_i, r_k)$ and $P_{rb}(r_k)$ probabilities. Makespan has been updated by using Equation 4.9.

$$\widetilde{ETC}(j_i, r_k) = (1 + P_f(j_i, r_k) + P_{rb}(r_k))ETC(j_i, r_k) \tag{4.8}$$

$$completion(j_i, r_k) = avail\_time_{r_k} \pm \widetilde{ETC}(j_i, r_k) \tag{4.9}$$

Cost $c(r_k, j_i)$ is the cost of job j_i which executes on resource r_k in addition to security-assurance cost that is defined in Equation 4.11.

$$cost = Min(c(r_k, j_i)) for 1 \leq k \leq n, 1 \leq i \leq m \tag{4.10}$$

$$c(r_k, j_i) = c_e(r_k, j_i) + c_s(r_k, j_i) \tag{4.11}$$

Equation 4.5 denotes $c_e(r_k, j_i)$ user's application execution cost and Equation 4.12 indicates $c_s(r_k, j_i)$ the cost of security-assured mapping of the user applications.

$$c_s(r_k, j_i) = \sum_{j_i \in J} (P_f(j_i, r_k) \times ETC(j_i, r_k)) \Big/ ETC_{m(j_i)} \times J \tag{4.12}$$

where as :

$$ETC_{m(j_i)} = max_{j_i \in J, r_k \in R} ETC(j_i, r_k) \tag{4.13}$$

In Equation 4.13, $ETC_{m(j_i)}$ is the (expected) maximal computation time of the tasks of the user.

The next section describes the hyper-heuristic based scheduling algorithm.

4.3 Hyper-heuristic based Scheduling Algorithm

Meta-heuristics are heuristics which control the search in a space of solutions performed by a single low-level heuristic. Optimality is not guaranteed in meta-heuristics. It requires multiple search parameters as there is a lack of theoretical basis. Meta-heuristic algorithms like tabu search, ant colony, etc., have different search methods but sometimes different searches may yield different solutions to the same problem. Thus, meta-heuristic approaches that perform well on a particular real-world problem may not work on all the problems. They may produce very poor solutions for other problems or even for other instances of the same problem. Thus, an extensive knowledge in both problem domain and appropriate heuristic techniques is required. A simplistic way of solving problem using meta-heuristic has been shown below:

```
if(problem type(p)==p1) then
apply(heuristic1, P);
elseif(problem(p)==p2) then
apply(heuristic2,P);
elseif(problem(p)==p3) then
apply(heuristic3,P);
elseif.....................
```

One logical extreme of such an approach would be an algorithm containing an infinite switch statements enumerating all finite problems and then applying the best known heuristic for each problem, which is not a good approach. A better approach is to find the best heuristic under the problem condition and then apply different heuristics to different parts or phases of the solution process. A hyper-heuristic 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 [177]. 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 [178]. Figure 4.2 shows the basic hyper-heuristic framework.

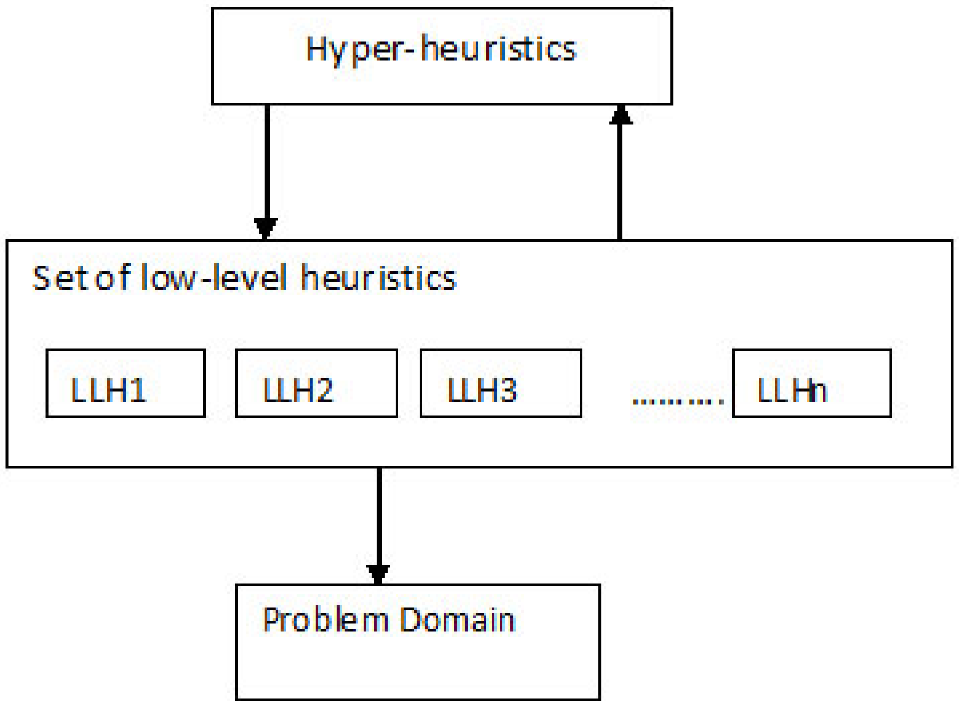

Figure 4.2: Hyper-heuristic Framework[201]

4.3.1 Bacterial Foraging Optimization

Bacterial Foraging Optimization (BFO) algorithm was proposed by Passino et al. [173][202]. It is population based numerical optimization algorithm based on foraging behavior of Escherichia coli bacteria. E.coli bacteria has a control system, which directs its behavior in food foraging. In the foraging theory, the objective of the animal is to search and obtain nutrients so that energy intake per unit time (E/T) is maximized. Foraging is a process in which a group of bacteria move 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 four main mechanisms: Chemotaxis, Swarming, Reproduction and Elimination-dispersal event.

Chemotaxis: It is the process of simulating the movement of E.coli bacteria, which is carried in a flagella, through swimming and tumbling. Two different ways of movement of E.coli bacteria are : Swim in the same direction and tumble in a random direction.

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\phi(j) \tag{4.14}$$

Where $\theta^i(j,k,l)$ stands for the current position of the i^{th} individual; j,k and l indicate the numbers of chemotaxis, reproduction and elimination-dispersal events respectively. $C(i)$ is the size of the step taken in a random direction and $\phi(j)$ is the direction angle of the j^{th} step.

If the cost at $\theta^i(j+1;\ k;\ l)$ is better than the cost at $\theta^i(j;\ k;\ l)$ then the bacterium takes another step of size $C(i)$ in that direction otherwise it is allowed to tumble in a random direction $\phi(j)$. This process is repeated until the number of steps taken is greater than the number of iterations in chemotaxis loop, N_c.

Swarming: 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. Each bacterium also releases repellants to signal the others to be at a minimum distance from it. Thus all of them will have a cell to cell attraction via attractant and cell to cell repulsion via repellant. The cell to cell signaling in E.coli swarm may be represented by the following function:

$$\begin{aligned} J_{cc}(\theta,p(j,k,l)) &= \textstyle\sum_{i=1}^{S} J_{cc}(\theta,\theta^i(j,k,l) \\ &= \textstyle\sum_{i=1}^{S}[-d_{attractant}\,exp(-w_{attractant}\sum_{m=1}^{p}(\theta_m-\theta_m^i)^2)] \qquad (4.15) \\ &\quad + \textstyle\sum_{i=1}^{S}[h_{repeelant}\,exp(-w_{repellant}\sum_{m=1}^{p}(\theta_m-\theta_m^i)^2)] \end{aligned}$$

Here $J_{cc}(\theta,p(j,k,l))$ represents objective function value to be added to the actual objective function, S is the total number of bacteria, p is the number of parameters to be optimized which are present in each bacterium. It denotes the combined cell-to-cell attraction and repelling effects, where $\theta=[\theta_1,\theta_2,.....\theta p]^T$ is a point on the optimization domain and θ_m^i is the m^{th} component of the i^{th} bacterium position θ^i. $d_{attractant}$ is the depth of the attractant released by the cell (a quantification of how much attractant is released) and $w_{attractant}$ is a measure of the width of the attractant signal. $h_{repellant}=d_{attractant}$ is the height of the repellant effect (magnitude of its effect) and $w_{repellant}$ is a measure of the width of the repellant.

Reproduction: After N_c chemotaxis steps, a reproduction step is taken. Fitness value of the 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.

Elimination and Dispersal: Elimination event may occur due to some sudden changes like a significant local rise of temperature or a part of them may move to some other regions in the environment that will effect the overall behavior of the bacteria heavily. The elimination and dispersal event destroys the performance of chemotaxis event but dispersal may place bacteria near good sources of food [174].

4.3.2 Scheduling Algorithm

In this section, the pseudo code of bacterial foraging based hyper-heuristic for resource scheduling in the Grid environment has been discussed. Each bacterium in genome is a partial solution and is represented as a heuristic (eg: select, move, swap, drop) or a sequence of heuristics. A low-level heuristic is operated upon by the hyper-heuristic. Low-level heuristics can be simple or complex and can be implemented as follows :

i. 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.

ii. Move job j_i from its current resource to some other resource.

iii. Randomly select a job and swap it with some other job.

iv. Finally, remove a randomly selected job from the job pool already scheduled. This is the only heuristic which will move the search into an infeasible region because any job may be unscheduled. But, the search will surely move back into its feasible region by un-scheduling the job that has other valid resources so that it can move into the next iteration.

BFO can be used as a top-level heuristic to manage the overall performance and quality of all the mechanisms. Objective of the BFO is to find the best low-level heuristic that generates the best solution for the resource scheduling problem. The low-level heuristics are then applied in a specific order to find an optimal solution of the problem instance. The selection process of a low-level heuristic in hyper-heuristic stops after a pre-defined number of iterations. The number of iterations have been fixed so as to keep the computation time low. Region of the BFO search space is the possible combination of low-level heuristics. A swarm of bacteria has been endowed with pre-constructed solutions and let them collectively learn the heuristic space, so as to guide their selection of an appropriate low-level heuristic to improve the solution. The pseudo code of bacterial foraging based hyper-heuristic is given in Algorithm 1. The detailed description of the proposed algorithm is as follows:-

Algorithm 1: Bacterial foraging based hyper-heuristic

```
Data: Number of jobs and number of available resources.
Result: Mapping of the each job to the resources.
begin
    initialize resourcelist[Number of Resources]
    initialize joblist[Number of Jobs]
    Input $r$ = number of iterations
    Input $n$= number of heuristics
    Initialize a random feasible solution
    $S$= The number of bacteria in the population
    $P$ = Dimensions of the search space
    $C(i)$ = The size of the step taken in the random direction specified by
    the tumble
    $N_c$ = Chemotaxis steps
    $N_s$ = Swimming length
    $N_{re}$ = Reproduction steps
    $N_{ed}$ = Elimination and dispersal events
    $P_{ed}$ = Elimination and dispersal probability
    joblist= get job to schedule(), resourcelist= get available resources()
    Elimination and dispersal loop $l = l + 1$
    Reproduction loop $k = k + 1$
    Chemotaxis loop $j = j + 1$
    For $i = 1, 2, 3, ....., S$ take a chemotaxis step for bacterium $i$ as follows.
    choose the heuristic
    Compute $Fitness(i, j, k, l)$
    $Fitness_{last} = Fitness(i, j, k, l)$
    Tumble: Generate a random vector $\Delta(i) \epsilon R^n$ with each element $\Delta_m(i)$,
    $m = 1, 2, 3....p$; a random number on[-1,1].
    Move: $\theta(i, j + 1, k, l) = \theta(i, j, k, l) + c(i) * \Delta(n, i)$
    $ComputeFitness(i, j + 1, k, l)$
    Swim, let $m = 0$
    while $(m < N_s)$ do
        let $m = m + 1$
        if $(Fitness(i, j + 1, k, l) < Fitness_{last})$ then
            $Fitness_{last} = Fitness(i, j + 1, k, l)$
            update
            $\theta(i, j + 1, k, l) = \theta(i, j + 1, k, l) + c(i) * \Delta(n, i)$
            $Fitness(i, j + 1, k, l) = \theta(i, j + 1, k, l))$
        else
            let $m = N_s$, This is the end of while statement.
    go to next bacterium $(i + 1)$ if $i \neq S$ (i.e., go to step Compute
    $Fitness(i, j, k, l)$) to process next bacterium
    if $(j < N_c)$ then
        go to step chemotaxis loop
    for $(i = 1 to S)$ do
        $Fitness^i_{health} = \sum_{j=1}^{N_c} Fitness(i, j, k, l)$
```

```
The $S_r$ bacteria with the highest $Fitness_{health}$ values die and the other $S_r$
bacteria with the best values split (and the copies that are made are placed
at the same location as their parent) ;
if ($k < N_{re}$) then
    next $k$ ;
Elimination-dispersal: For $i = 1, 2, ........S$ with probability $P_{ed}$, eliminate
and disperse each bacterium ;
if ($l < N_{ed}$) then
    next $l$ ;
Apply the heuristic ;
while there are unscheduled jobs in the queue do
    for every resource is in resource list do
        get the next job from queue ;
        schedule the job on the resource on the basis of fitness;
Repeat each and every step till all the jobs are allocated;
```

- A resource list is obtained from the resource provisioning unit after provisioning of user's requests. Once the resource list has been obtained, a job list and a random feasible solution are initialized.

- The task to choose the best heuristic from low-level heuristics is started. There are a number of bacterium, each of which represents a hyper-heuristic agent supplied with an initial solution in the solution space and an access to the evaluation function. The bacterium constructs a sequence of heuristics through chemotaxis steps.

- It will then select a low-level heuristic at each decision point and compute fitness function $Fitness(i, j, k, l)$.

- $Fitness(i, j + 1, k, l)$ will be computed and then the process of swim will be started till the bacteria have not climbed too long. If at $\theta^i(j + 1;\ k;\ l)$ the cost $(Fitness(i, j + 1, k, l)$ is better than at $\theta^i(j, k, l)$, $Fitness_{last}$ then the bacterium takes another step of size $C(i)$ in that direction. This swim is continued as long as it continues to reduce the cost, but only up to a maximum number of steps, N_s.

- If $j < N_c$, go to chemotaxis loop. In this case, continue chemotaxis, since the life of bacteria is not over.

- For the given k, l and for each $i = 1, 2, 3, ..., S$, let $Fitness^i_{health} = \sum_{j=1}^{N_c}$ $Fitness(i, j, k, l)$ be the health of the bacterium i (a measure of how many nutrients it got over its lifetime and how successful it was at avoiding noxious substances). Sort bacteria in order of ascending cost $Fitness_{health}$ (higher cost means lower health). (After each bacterium has visited a certain number of heuristics, the fitness value of each bacterium is sorted in an ascending order to encode a good solution).

- During the reproduction step, a bacterium may choose to reject a new solution if it discovers that it is poorer than the current solution. If $k < N_{re}$ then go to reproduction loop: $k = k + 1$.

- For $i = 1, 2, ...S$ with probability P_{ed}, eliminate and disperse each bacterium which results in keeping the number of bacteria in the population constant. To do this, if a bacterium is eliminated, simply disperse one more bacterium to a random location on the optimization domain. If $l < N_{ed}$, then go to the elimination and dispersal loop; otherwise end.

- After selection of a low-level heuristic, it is then applied to the problem. Resource scheduling is performed till there are no unscheduled jobs in the queue.

4.3.3 Algorithm Complexity

For each generation, a given population of bacteria representing a sequence of heuristics is evaluated for fitness. Bacterial Foraging Optimization based Hyper-heuristic (BFOHH) performance is usually measured by the number of fitness function evaluations done during the course of a run. The main operations performed during BFOHH are as follows:

- For fixed population size which is the usual case in BFOHH implementations, the number of fitness function evaluations is given by the product of population size and the number of generations.

- To allocate any resource to an application, a number of iterations is to be done over each user application/job and resource i,e O(mn).

- All the operations are done for each application,i.e., n times. Thus, complexity of the bacterial foraging based hyper-heuristic resource scheduling algorithm is given by $O(mn^2 \times generations \times populationSize)$.

4.4 Conclusion

This chapter presented the Grid resource scheduling model and then formulated the objective function based on QoS parameters. This objective function tries to take care of interests of both providers and users by minimizing the makespan and cost simultaneously. Further, a BFO based hyper-heuristic resource scheduling algorithm has been proposed for the mapping of applications on the ingredient resources. The next chapter discusses the verification and validation of QoS parameter(s) based resource provisioning policies and resource scheduling algorithm.