Chapter 7

Two level Scheduling Decision Algorithm

7.1 Introduction

Evolution in computing techniques emerged to grid computing paradigm to address need large scale applications and use of idle resources. Computational grid (CG) is the collection of computational resources (computing capacity) designed to satisfy computing demands of grid applications. The Grid consists of Grid Resources (GR), Grid Information Service (GIS) and the Grid Scheduler (GS). GR are capable of executing jobs, GIS provides the status of resources realizing grid. The GS produces a job queue for the GR using desired matching criteria. GR properties for instance CPU idle time, free memory, cost of execution etc. are matched with job demand parameters such as estimated execution time, required memory and Quality of service (QoS) etc. GS uses GSA for mapping jobs on GR. Research in grid scheduling shows the use of classical strategies for example First in First Out (FIFO), Round Robin (RR) etc. while designing initial versions of grid. These are static scheduling algorithms since they lack the consideration of changes in GR properties at runtime. Intelligent static algorithms are the algorithms with learning capability. These algorithms use various learning methods to infer about the behavior of GR. The heuristic methods for instance Min-Min, Min-Max use heuristic functions to produce next optimized schedule. Comparative study shows the considerable improvement in the performance of learning algorithms than classical static methods. Now days, demands of applications are not only execution but with satisfaction of QoS [26] parameters such as security, reliability, availability, cost etc. are added job requirements. Design of security based algorithms includes security parameters which are also matched with basic computation needs. Further research in GSA shows the use of nature inspired methods in the design of scheduling
algorithms. For instance GA, ACO, Simulated Annealing (SA) and Neural Network (NN)[19] prove to be the milestone in the optimization of scheduling algorithms. GS can predict more accurate schedule which in turn improves success rate of job execution. Performance improvement in terms of Turn Around Time (TAT) is one of the major goals while experimenting new techniques.

7.2 Motivation

Comparative study shows the concrete improvement in performance when nature inspired algorithms are used for job scheduling. These are the learning algorithms and each of them has unique reasoning method. The proposal is to combine the benefits of two algorithms so as to predict more accurate schedule for execution. This implementation will reduce the execution time for grid application since a) Success rate of execution is maximized b) It reduces rescheduling time c) Applied intelligence is maximized. We perform experimentation which studies the improvements and overheads in scheduling. Figure 7.1 shows levels of decision.

![Figure 7.1: Decision Steps for TLDA](image)

7.3 Proposed Approach

In this proposed TLDA approach the scheduler consists of two levels of decision modules. We obtain the schedule by applying ACO. The second stage refines the decisions taking into
consideration the results and past results executions. We propose to use GA for the second state. Final schedule considers the advantages of both these methods. Application of ACO to scheduling problem and GA proves to be beneficial for intelligent scheduling. Figure shows Two stage decision engine.

![Figure 7.2: Flowchart of GA used for Scheduling](image)

7.4 Design of Algorithms

In this section we present the design of algorithms applied to scheduling of jobs on grid.

7.4.1 Ant Colony Optimization ACO

ACO algorithm is the learning algorithm based on natural ants. Algorithm formulates the analogy of ants working to find their food and optimize path to their food to be followed by successors. The ACO algorithm uses a colony of artificial ants that behave as co-operative agents in a mathematical space were they are allowed to search and reinforce pathways (solutions) in order to find the optimal ones. Solution that satisfies the constraints is feasible. After initialization of the pheromone trails, ants construct feasible solutions, starting from random nodes, then the pheromone trails are updated. At each step ants compute a set of feasible moves and select the
best one (according to some probabilistic rules) to carry out the rest of the tour. The transition probability is based on the heuristic information and pheromone trail level of the move. The higher value of the pheromone and the heuristic information, the more profitable it is to select this move and resume the search. In the beginning, the initial pheromone level is set to a small positive constant value and then ants update this value after completing the construction stage.

Steps of ACO applied to scheduling are procedure

**ACO Algorithm**

begin Initialize the pheromone
while stopping criterion not satisfied
do Position each ant in a starting node
repeat for each ant
do Chose next node by applying the state transition rate
end for
until every ant has build a solution
Update the pheromone
end while
end
7.4.2 Genetic Algorithm (GA)

GA is refines the schedule prepared by ACO. GA is analogically applied to formulate the decision from set of chromosomes. Mutation and crossover of these chromosomes generates the optimized final state. Genetic algorithms combine exploitation of past results with the exploration of new areas of the search space by using survival of the fittest techniques combined with a structure randomized information exchange. The generalized procedure of GA for scheduling is

**GA Algorithm**

Generate initial population of jobs
Select random two parents from initial population
Perform crossover to produce child
Perform Mutation for child
Find the fitness of child
Scheduling is best chromosome

The proposed approach uses results of ACO as initial population for mutation and crossover stage. The fitness function for new schedule is optimized TAT for the given application.

7.4.3 Two level Decision Algorithm

In TLDA approach We combine the randomness of GA to obtain the fitness function. GA initializes two random schedules S1 and S2. We produce results for GA using steps crossover and mutation as in Figure 7.3.

In TLDA we decide schedule S2 from application of ACO. GA fine tunes this schedule by generation of randomized schedule S1. The crossover and mutation takes place on these S1 and S2 schedules. In last step we verify fitness value of thus obtained schedule for selection. Figure 3 shows the design of proposed approach of TLDA.

7.5 Scheduler Implementation

Our implementation of scheduler provides the interface for application submission and selection of algorithms for schedule preparation. Scheduler accepts an application and decomposes this
into set of jobs. These jobs are then queued for the suggested GR for execution. Figure 7.4 shows the steps executed by scheduler while scheduling jobs. In TLDA, the failed jobs get rescheduled applying the algorithm initially selected for scheduling. The grid observes for the successful execution of the jobs and reports the failures. Figure 7.5 shows the snapshot of user interface which provides status of jobs executing on the grid. We vary the number of applications submitted to grid. For a single application grid performance is shown in Figure 6. We independently apply ACO, Fuzzy, GA and ACO and GA together.

Results show the improvement in performance by using TLDA. The minimum execution time is obtained for our approach as compared to other counterparts. We apply the increasing number of applications so as to obtain the performance for ACO, GA and TLDA. Comparison shows improvement in the execution time over the individual performance of the ACO and GA.
Table 7.1 Shows the execution time obtained with respect to algorithm. For instance ACO takes 14.5 Sec. whereas TLDA executes an application in 12.5 sec. The bar graph Figure 7.6 Display performance with increasing applications for each algorithm. Our approach shows improvement over ACO and GA.

7.6 Observations

Our Implementation of TLDA shows improvement over nature based algorithms applied independently. We observe the decrease in execution time for given number of applications. The overhead of decision making time can be neglected as compared to improvement in execution time. The approach is more dynamic and considers changes at run time in resource properties.
Table 7.1: Table shows the execution time for applications submitted to TLDA

<table>
<thead>
<tr>
<th>Number of Applications</th>
<th>ACO (sec)</th>
<th>GA (sec)</th>
<th>TLDA (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3.5</td>
</tr>
<tr>
<td>3</td>
<td>4.5</td>
<td>4.5</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>6.5</td>
<td>5.5</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>14.5</td>
<td>15.5</td>
<td>12.5</td>
</tr>
</tbody>
</table>

Figure 7.6: Comparative performance of TLDA