6

Experimental Details and Results

6.1 Introduction

An experiment is generally used to help solve practical problems and to support or negate theoretical assumptions. To support the algorithms proposed in Chapter 3 and Chapter 4 of the thesis, several experiments were performed taking different scenarios into consideration. Chapter 5 gives the implementation details of the grid test bed and GridSim based grid simulator used for experimentation. This chapter provides the details of all experimental scenarios, input parameters and comparison with the existing algorithms. Experimental test bed uses the simulation model as used by Braun et. al.[21].

6.2 Simulation Model

To evaluate the proposed algorithms, a simulation model similar to one used in the [21] is used. To simulate the heterogeneous environment, completion time for every job on every resource is contained in a completion time matrix, *CT*, which is an $N \times M$ matrix. Since, in a heterogeneous environment any job executed on a machine/ resource can have different completion time, hence, this matrix contains the completion time of every job on every machine / resource. In the actual grid systems state estimation is generally done based on the task profiling and analytical benchmarking. In this experimental test bed, a pre - computed completion time matrix is used, instead of actual task profiling.

A row in the *CT* matrix contains the completion time for a given job on each resource, where as a column consists of the completion time of every job on a given resource. Hence, for a job J_i and a resource R_j, an entry of the *CT* matrix contains the completion time of job J_i on resource R_j.

6.2.1 Constructing CT matrix

Constructing the *CT* matrix is a three step process as given below in the pseudo code. In the *Step 1*, all necessary initialization are performed. Completion time matrix, *CT*, is a two dimensional array of size $N \times M$ where, N is the total number of jobs and M is the total number of machines / resources. To construct the *CT* matrix, first a baseline vector, B, of floating point number is generated. Baseline vector, B, will contain N elements.

Step 2 explains the creation of Baseline vector, B. To generate the baseline vector, B, a uniform random number is X_b is generated such that $X_b \in [1, U_b)$ (U_b is the upper bound of the range of possible values of B). X_b is generated repeatedly for N times and baseline vector, B, is constructed such that $B[i] = X_b^i$ where $1 \leq i \leq N$.

In the *Step 3*, rows of the *CT* matrix are generated from the baseline vector, B. A uniform random number X_r, known as row multiplier, is generated such that $X_r \in [1, U_r)$ (U_r is the upper bound of the range of possible values of the row multiplier, X_r). M different row multipliers are required for a row. $X_r^{i,j}$ is generated for every element of the *CT* matrix in a row and final value of *CT* matrix element is generated as $CT[i,j] = X_r^{i,j} * B[i]$, where $1 \leq i \leq N$ and $1 \leq j \leq M$.

6.2.2 Pseudo Code

```
// Step 1. Do Initializations
Initialize CT[N][M]
Initialize B[N]
Initialize Ub
Initialize Ur

// Where CT[N][M] is the Completion time matrix
// B[N] is the base line vector
// N = Total number of jobs
// M = Total Number of Machines / Resources
// Ub = upper bound of range of possible values of
baseline vector B.
// Ur = upper bound of Range of possible values for Row
multiplier

// Step 2. Generate a base line vector
For i = 1 to N
{
     // Get a Random Number Xi in the range (1 - Ub)
     Xb = getRandom(1, Ub);
     B[i] = Xb
}

// Step 3. Generate Rows and Fill the CT[N][M]
For i = 1 to N
{
     For j = 1 to M
```

```
        {
                // Generate Row Multiplier Xr,
                // where Xr is a uniform Random number in the
                // range (1-Ur)
                Xr = getRandom(1, Ur);
                CT[i][j] = B[i] * Xr
        }
}
```

6.2.3 Achieving Heterogeneity

The characteristics of the *CT* matrix are varied to achieve the heterogeneity. The variation among the execution times of jobs for a given machine is defined as *task heterogeneity*. Task heterogeneity is achieved by changing the upper bound of the random numbers with in the base line column vector, *B*. High Task heterogeneity is achieved taking the U_b = 3000 and low task heterogeneity is achieved by taking U_b = 100. Similarly high machine heterogeneity and low machine heterogeneity is achieved by varying the upper bound, U_r, of the random number used to multiply the base line column vector. High machine heterogeneity is represented by taking U_r = 1000 and low machine heterogeneity is achieved by taking U_r = 10.

Further the *CT* matrix can be categorized, based on the consistencies, as *consistent, partially consistent and inconsistent*. Consistent means, if a resource R_j executes the job J_i faster than resource R_k then, it will execute all the jobs faster than R_k. On the other hand, inconsistent means a resource R_j can be faster than resource R_k for some tasks and slower for others. Partially consistent matrix is inconsistent matrix that contains the consistent sub matrix of predefined size. Since grid is a truly heterogeneous environment, hence, only inconsistent and partially consistent matrices are considered here for experimentation. Sample 4×4 excerpt for inconsistent and partially consistent matrices are shown in the Table 6.1 and Table 6.2 respectively. These are taken from the actual matrix of 512×16, used in the experiments.

933803.1	2156144.2	2202018.1	2286210
479091.9	150324.5	386338.1	401682.9
1400308.1	2378363	2458087.2	351387.4
576144.9	1475908.2	424448.8	576238.7

Table 6.1: 4×4 excerpt from inconsistent, high task heterogeneity and high machine heterogeneity matrix

101202.5	16453.7	64152.5	29172.8
195067.8	79175.8	787263.3	173239.2
434445.7	135989.1	496326.8	221097.9
138449	32730.9	93025.9	90044.4

Table 6.2: 4×4 excerpt from partially consistent, high task heterogeneity and high machine heterogeneity matrix

6.2.4 Algorithm parameters used in experiments

This section describes the parameters and their values for different algorithms that are common for all the experiments. Other values that are specific to the experiments are described along with the experimental details. Table 6.3 below provides the parameters names and their values taken in the experiments.

Algorithm Name	Parameter Name	Value	Description
SGASchedule	X_c	0.75	Cross Over Probability
	X_m	0.01	Mutation Probability
	ThresholdSD	1.00E-06	Threshold Standard Deviation
ACOSchedule	α	0.6	Priority of *Pheromone trail* to decide the probability of next move
	β	0.4	Priority of *Heuristic information* to decide the probability of next move
HybridGSA	T	1	Initial Temperature
	Υ	0.9	Cooling Rate
	X_c	0.75	Cross Over Probability
	X_m	0.01	Mutation Probability
	ThresholdSD	1.00E-06	Threshold Standard Deviation

Table 6.3: Parameters and their values used in experiments.

Table 6.3 shows the parameters and their values with respect to different algorithms as taken in the experiments. X_c and X_m are the crossover and mutation probabilities used in GA based algorithms SGASchedule and HybridGSA. Similarly *ThresholdSD* is the threshold standard deviation. If the standard deviation of the population comes below this *ThresholdSD* the solution is assumed to be converged. *T* and Υ are the initial temperature and cooling rate for simulated annealing, used in HybridGSA, which are used to decide the selection probability of newly formed offspring.

6.3 Experimental Results for proposed scheduling algorithms using Grid Test Bed

This section provides the experimental details and results for the proposed scheduling algorithms SGASchedule, ACOSchedule and HybridGSA. As mentioned in the simulation model, Completion Time matrix can vary as per the heterogeneity. Since grid is highly dynamic and heterogeneous environment, so we have not considered consistent CT matrix for our experimentations. Inconsistent and partially consistent matrices are considered only. Also for our experimental purposes we are considering high task heterogeneity only, where as both high and low machine heterogeneity is considered. Since high task heterogeneity is considered for all experiments, hence $U_b = 3000$ is taken for the construction of all *CT* matrices. To compare the results with already existing results, we have considered the 512×16 matrices as given in [21], i.e. $N = 512$ (number of jobs) and $M = 16$ (number of machines). Table 6.4 shows results obtained from different experiments for minimizing the makespan and graphs in Figure 6.1 – 6.4 below shows the comparative analysis among different proposed scheduling algorithms and also with the existing GA and SA based algorithms discussed in [21].

6.3.1 Minimizing makespan

The existing results were available for minimizing the makespan only, so minimizing makespan is taken as the objective function for the experiments and comparison purposes. This is achieved by setting the $\theta = 0$ in the Equation 3.1. All these values are averaged over *10* simulations runs for each type of matrix and each type of algorithm.

	SGASchedule	HybridGSA	ACOSchedule
Inconsistent, High Machine Heterogeniety	341512	467784	389161
Inconsistent, Low Machine Heterogeniety	71019	90127	82194
Partially Consistent, High Machine Heterogeniety	391156	521016	434032
Partially Consistent, Low Machine Heterogeniety	82114	112147	100213

Table 6.4: Makespan results obtained from experimental results for proposed algorithms

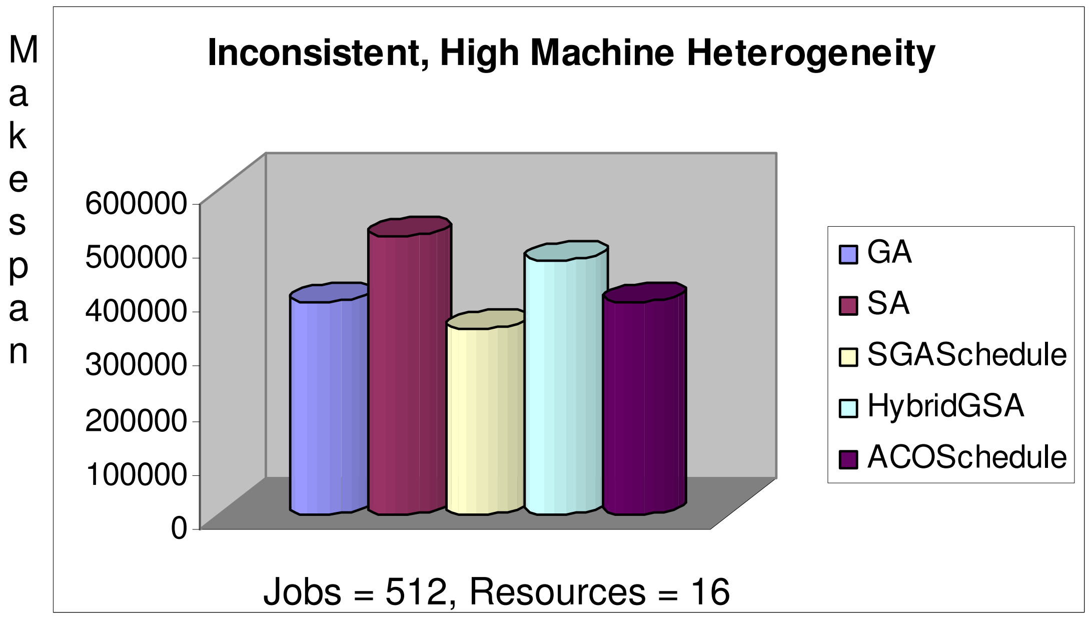

Figure 6.1: Comparison results for Inconsistent, High Machine heterogeneity CT matrix

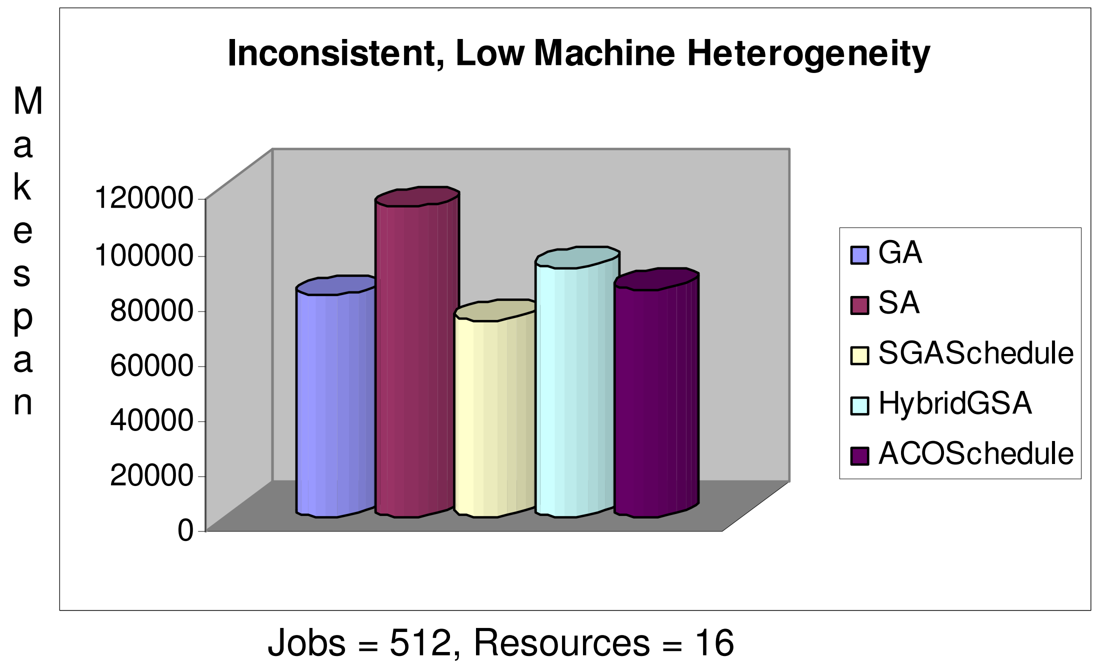

Figure 6.2: Comparison results for Inconsistent, Low Machine heterogeneity CT matrix

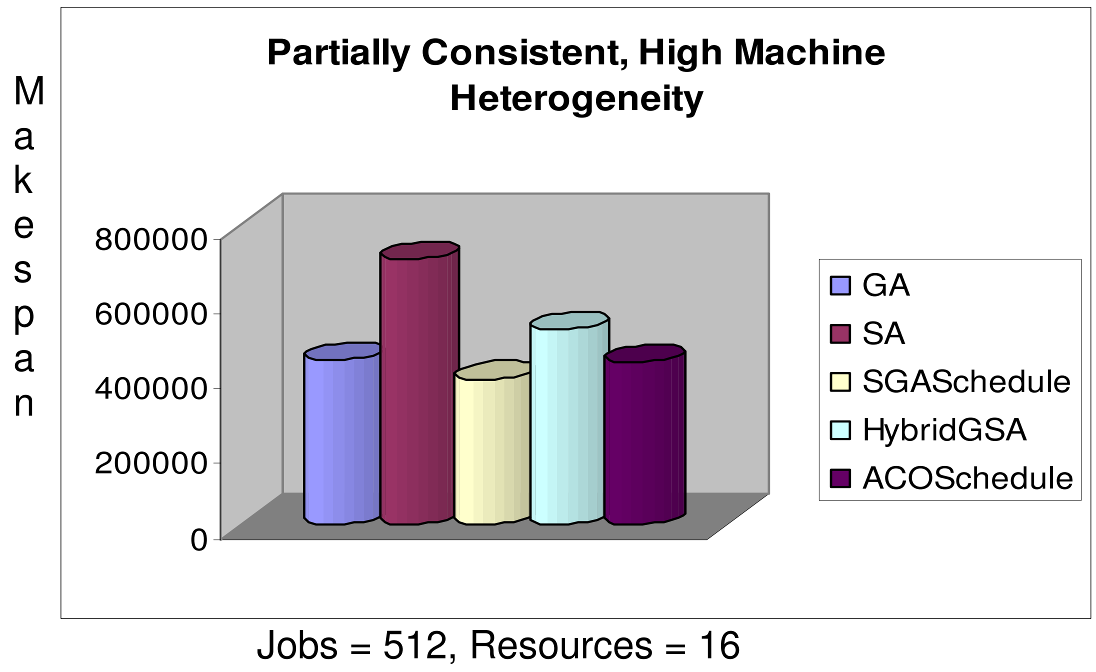

Figure 6.3: Comparison results for Partially Consistent, High Machine heterogeneity CT matrix

Graphs in Figure 6.1-6.4 show the comparison of proposed algorithms with the existing ones. GA and SA represent the results of existing genetic algorithms and simulated annealing based scheduling algorithms as given in [21], where as the

SGASchedule, ACOSchedule and HybridGSA represent the proposed grid scheduling algorithms.

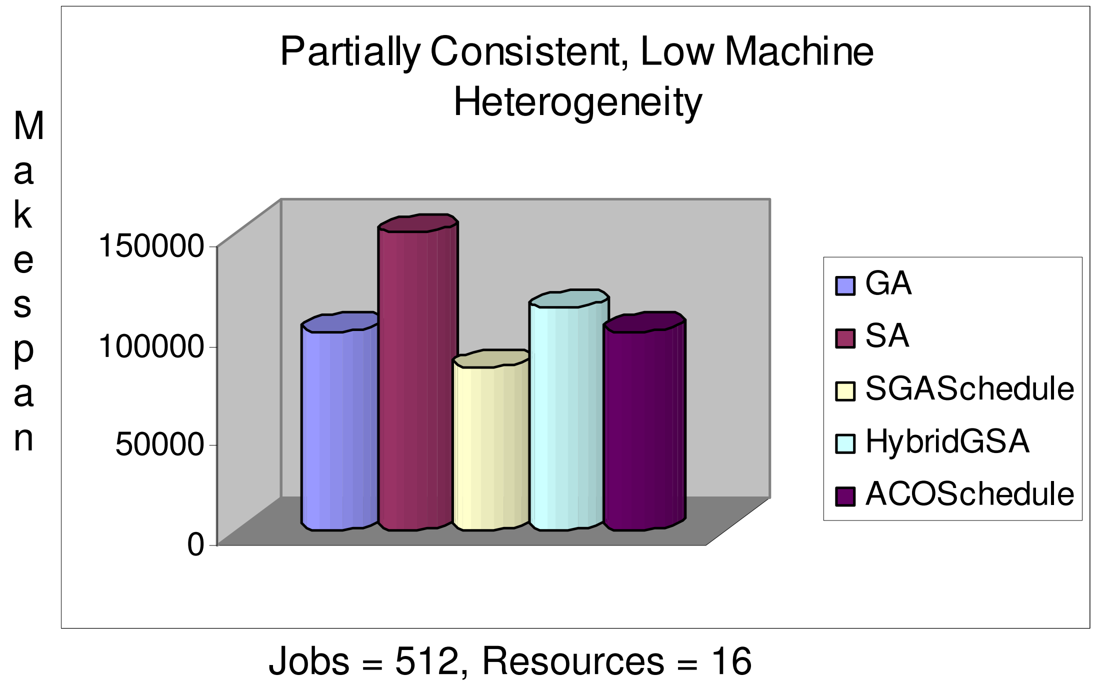

Figure 6.4: Comparison results for Partially Consistent, Low Machine heterogeneity CT matrix

6.3.2 Minimizing Cumulative Time delay (T_{delay})

Next experiments were carried on to compare proposed algorithms for cumulative time delay (T_{delay}), instead of makespan. This is achieved by setting $\omega = 0$ in the Equation 3.1.

	SGASchedule	HybridGSA	ACOSchedule
Inconsistent High Machine Heterogeneity	17213	23121	19213
Partially consistent High Machine Heterogeneity	19347	24921	20519

Table 6.5: Comparison results for minimizing T_{delay}

Since there were no results for minimizing the cumulative time delay in for existing algorithms. Hence the comparison is done only for the proposed algorithms. This is achieved with the experimental test bed by specifying deadline, while submitting the jobs. Experiments were done inconsistent *CT* matrix with high machine heterogeneity and partially consistent *CT* matrix with high machine heterogeneity. Also for both of these experiments task heterogeneity is considered as high, as in all other experiments. Results are given in Table 6.5. Comparison among proposed algorithms is depicted by the graphs in Figure 6.5 and Figure 6.6.

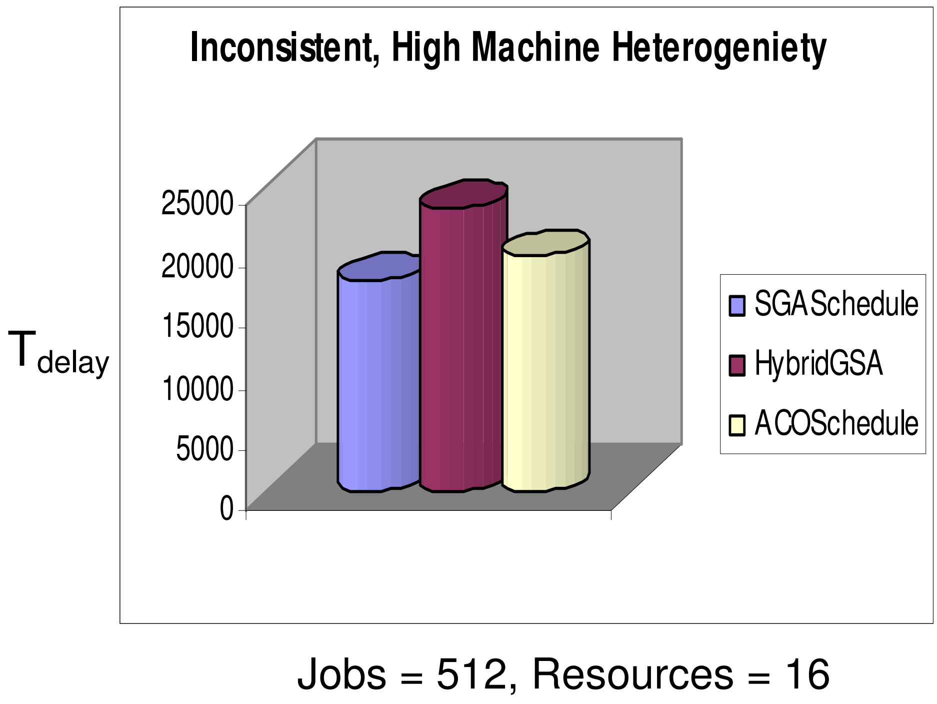

Figure 6.5: Comparison of cumulative time delay (T_{delay}) for proposed algorithms for inconsistent, high machine heterogeneity.

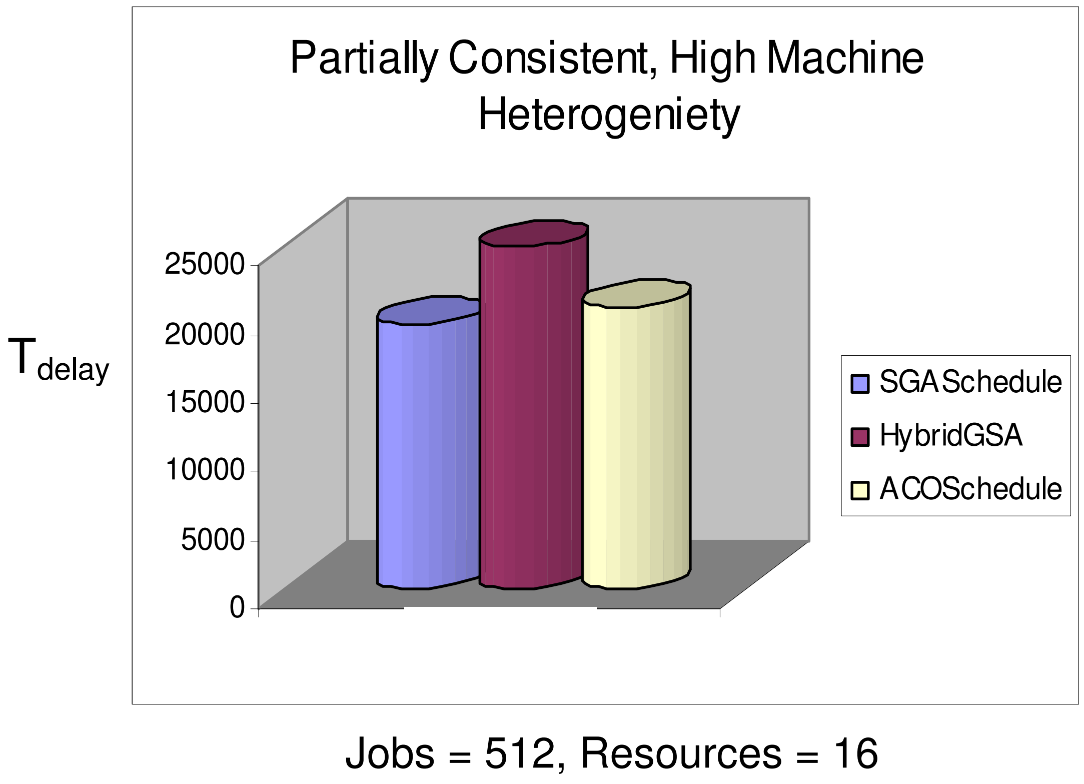

Figure 6.6: Comparison of cumulative time delay (T_{delay}) for proposed algorithms for partially consistent high machine heterogeniety.

It is observed from the above results that SGASchedule (SexualGA based scheduling algorithm) outperform all other proposed as well as existing algorithms. SGASchedule uses SexualGA based selection mechanism, where both parents are selected using different selection schemes. It uses roulette wheel selection to select one parent and n-tournament selection as the selection mechanism for selecting second parent. Hence it truly reflects the selection in the living beings by correctly simulating the male vigor and female choice. This SexualGA based selection mechanism reduces the selection pressure for the individuals and hence provides the better exploration of search space. SGASchedule performs best for minimizing both of the objectives: minimizing the makespan and minimizing the cumulative time delay. ACOSchedule is the second best in achieving the objectives. It uses pheromone deposited by ants as the learning mechanism to schedule the jobs on to the resources which is further improved by applying the genetic operators. Application of genetic operators (crossover and mutation) reduces the probability of ACO based algorithm to saturate without proper exploration of search space. It is clear from the results that ACOSchedule performs almost same (sometimes better) as existing GA based algorithm which was considered as best scheduling heuristic till now. HybridGSA (a hybrid approach based on Genetic Algorithms and simulated annealing) performed better than existing simulated annealing based scheduling algorithm. But still existing GA based algorithm is better than the hybrid approach for grid scheduling problem.

6.4 Experimental Results on GridSim based Grid Simulator

Grid simulator, as mentioned in the previous chapter, is a simple centralized broker that has been realized on the top of a well-known grid simulation toolkit 'GridSim' to validate the efficacy of proposed scheduling algorithms. Since grid is very dynamic and heterogeneous environment, hence it is very difficult to perform experiments with large size grids. Hence GridSim based grid simulator provides a controlled environment. Also comparison is done with the Nimrod/G implementation on GridSim.

To compare the results with Nimrod/G [103], experiments have been performed with grid simulator. Results have been obtained by scheduling independent sets of jobs generated randomly on Nimrod/G set on time sharing mode and same set is scheduled using the SGASchedule (proposed algorithm) on grid simulator. Three different experiments were performed with *30* resources / machines. Number of jobs / gridlets is taken as *500, 800* and *1000* for three experiments. (A Gridlet is a package that contains all the information related to the job and its execution management details such as job length expressed in MI (Millions Instruction), the size of input and output files, and the job owner id. Individual users model their application by creating

Gridlets for processing them on Grid resources.). In GridSim based simulations, a machine / resource can have more than one processing elements. But for the experiments, all the machines are assumed to have one processing element only. Hence the terms “machine”, “resource”, “processor” are same.

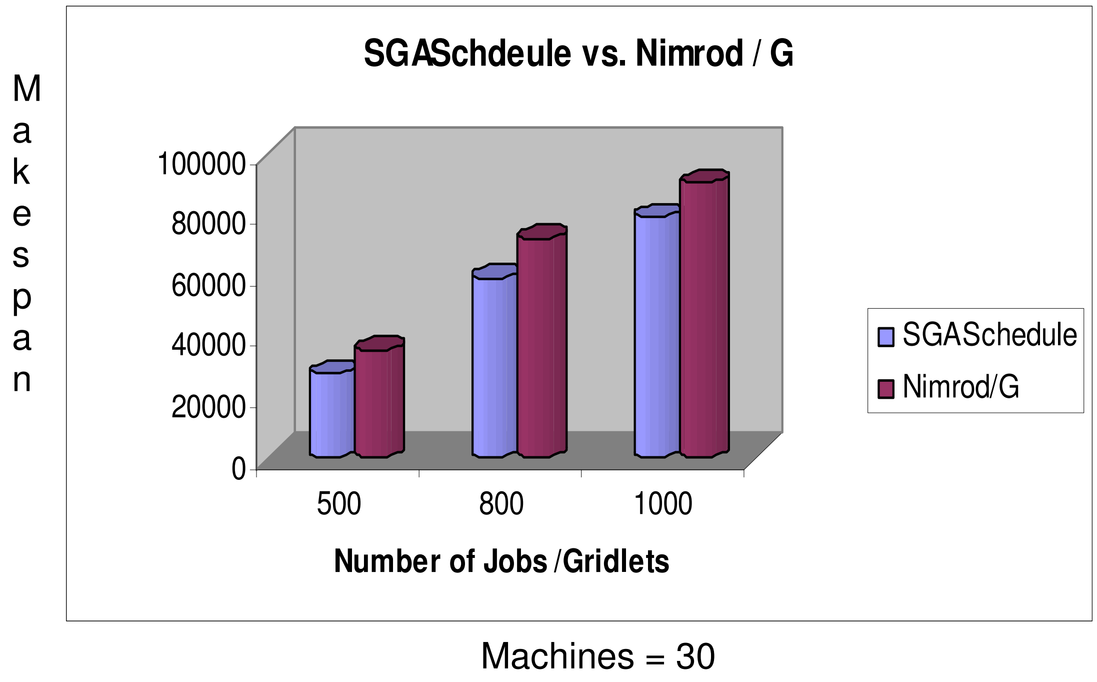

Figure 6.7: Comparison of SGASchedule Vs. Nimrod/G

It is clear from the Figure 6.7 that SGASchedule performs better than Nimrod/G implementation [103] for minimizing makespan. As the number of jobs / Gridlets increase, the SexualGA based proposed scheduling algorithm, SGASchedule, performs better than the Nimrod/G. Nimrod/G schedules the jobs using first come first served (FCFS) basis initially and switches to GA based scheduling at later stage. On the other hand the proposed algorithm uses GAs for scheduling from start to end. Also the use of two different selection schemes, as mentioned earlier, aids more exploration of search space. The combined effect of these two factors makes SGASchedule perform better as compared to Nimrod/G.

6.5 Experimental results for Proposed Resource Discovery Algorithm

Experimental details and results for proposed grid scheduling algorithms are discussed in previous section. This section discusses the experimental details and results for proposed ACO guided mobile agents based resource discovery algorithm (ACORD). Implementation details are provided in the Chapter 5. To perform the experiments and ACO based multi-agent environment is implemented by integrating proposed algorithm in the Grid Test bed.

Experiments are done by simulating the grids of different sizes varying from *200* nodes to *1000* nodes (node represents a machine that can host one or more resources for sharing). Each node is having varying degree of resources shared on grid. Resources were selected from the list of *100* resource IDs and these are assigned randomly to all the nodes. Nodes were added and removed during simulation runs to simulate the dynamic nature of the grid. User requests are generated using non-uniform (Gaussian) distribution. This distribution displays the patterns very similar to real world systems. A sample non-uniform distribution of user requests is shown in Figure 6.8.

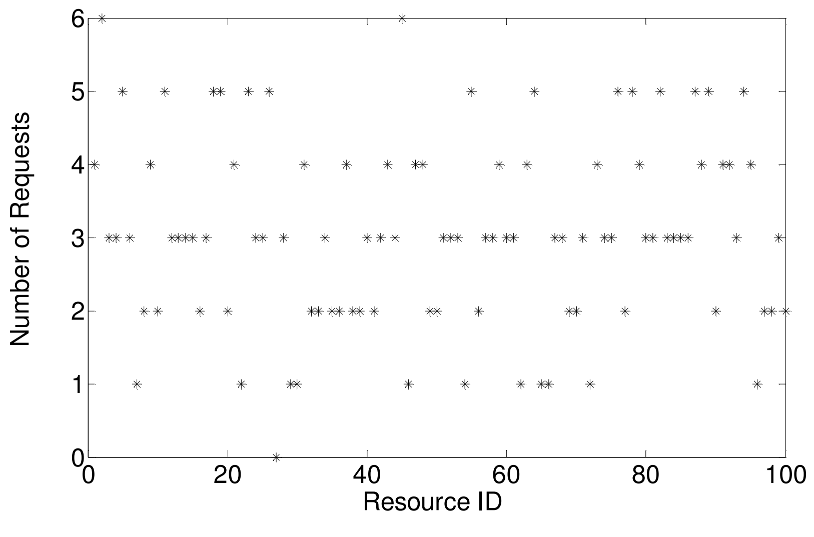

Figure 6.8: User Requests Profile (Gaussian distribution)

First experiment is carried out to find out the appropriate number of ants (BranchingFactor) to be created corresponding to the resource query. Same experiment is repeated by varying the number of ants from *1* to *50* and number of ants vs. percentage query hits is plotted. Values of α and β (Equation 4.4) are taken as *0.6* and *0.4* respectively. The plot in Figure 6.9 shows that as we increase the number of ants, query hits also increase. For lesser number of ants there is lesser pheromone deposited to guide the ants and hence less hits. Also for number of ants above *20,* there is not much increase in the number of query hits. So *15* seem to be the appropriate number of ants. For all other experiments, *15* ants are taken per query.

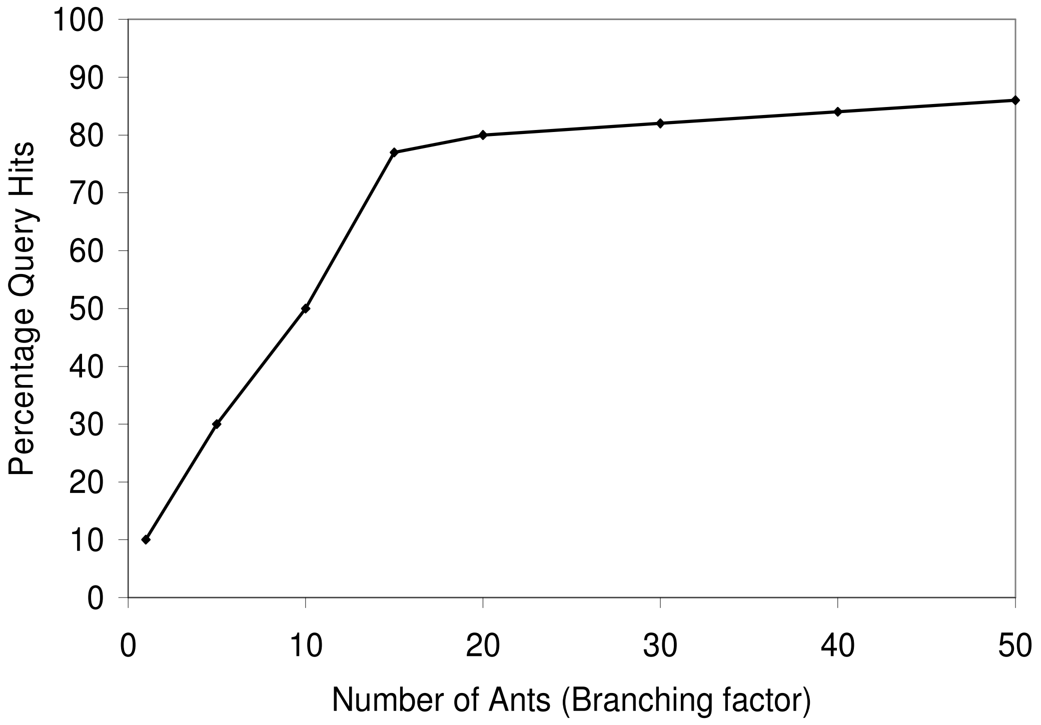

Figure 6.9: Number of Ants Vs Query Hits

Finally, the performance of ACORD is tested by varying the number of nodes and hence the network size. *MaxNodes* is taken as *10*, means an ant is allowed to visit at most *10* nodes for its search. Percentage of Query hits Vs. Network size is depicted in the plot given in Figure 6.10. This can be concluded from the Figure 6.10 that on an average there are *80%* of query hits. There is a small decrease in the query hits as the network size grows. As the network size grows *MaxNodes* needs to be adjusted accordingly. This will increase the search periphery of the ant before it actually declares the query miss. It is established from the results shown in plot (Figure 6.10) that as we increase the *MaxNodes* from *10* to *15*, the number of query hits also increase at increased network sizes. The average query hits percentage increases to approximately *85%*.

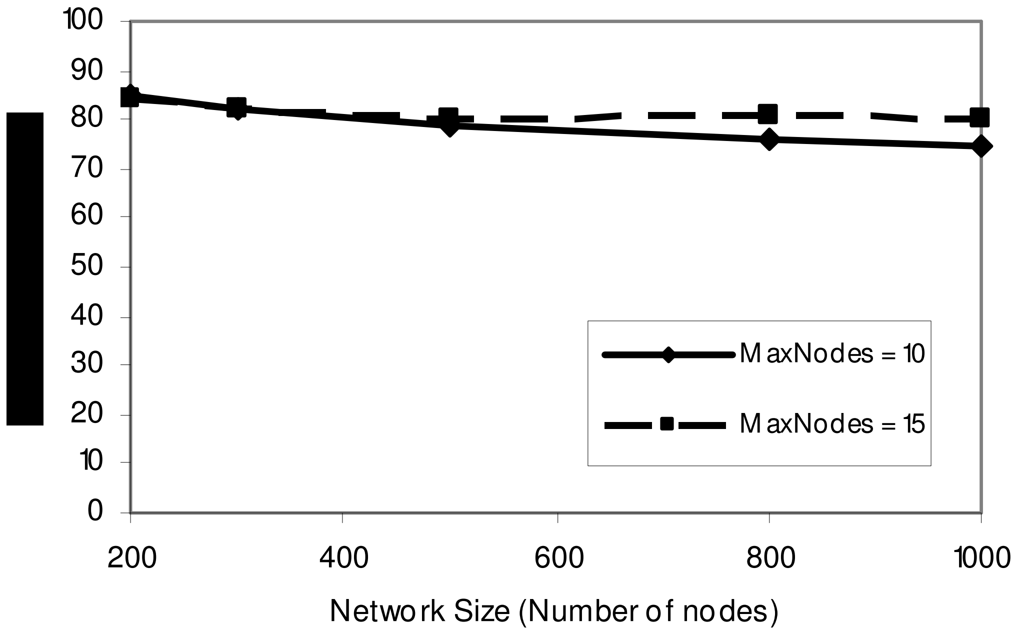

Figure 6.10: Network Size Vs. Query Hits

Experimental results show that the ACORD is scalable and can cope with the increasing network sizes with the adjustment of few parameters mentioned in the algorithm. Also the use of mobile agents shifts the actual processing to the different nodes rather than having the whole processing on a centralized machine.

6.6 Summary

This chapter provides the experimental details and results of the proposed algorithms in a simulated environment. Experiments are performed based on the inconsistent and partially consistent completion time matrix with high task and machine heterogeneity. Results show that proposed meta-heuristic based algorithms perform better than the existing algorithms for grid scheduling. Also the experimental details about ACO guided mobile agent based proposed resource discovery algorithm are given and results show that with the adjustment of few parameters this noble approach is able to cope with the growing network sizes in a grid environment.