Int. J. Open Problems Compt. Math., Vol. 1, No. 3, December 2008

Ant Algorithm for Grid Scheduling Powered by Local Search

Kousalya.K and Balasubramanie.P

Department of Computer Science and Engineering , Kongu Engineering College, Tamilnadu,India,

E-mail: keerthi.kous@gmail.com, pbalu_20032001@yahoo.co.in

Abstract

Grid computing is a form of distributed computing that coordinates and shows computing power, applications, data storage and network resources across dynamic and geographically dispersed organizations. Resource management and application scheduling are the two major problems in grid computing. The resources are heterogeneous in terms of architecture, power, configuration and availability. This complicates the task scheduling problem. The major objective of grid scheduling is to reduce the makespan. Hence the scheduling must consider some specific characteristics of the job and decide the metrics to be used accordingly. Ant algorithm, which is one of the heuristic algorithm suits well for the grid scheduling environment. This paper proposes an modified ant algorithm for Grid scheduling problem that is combined with local search. The proposed ant algorithm takes into consideration the free time of the resources and the execution time of the jobs to achieve better resource utilization and better scheduling. In the evaluation study, a number of intensive experiments are conducted using the standard bench mark problem. The result shows that the proposed ant algorithm is capable of producing high quality scheduling of jobs to grid resources. Thus the algorithm can be used to design efficient dynamic schedulers for real time grid environments.

Keywords: *computational grid, scheduling algorithm, Heuristic approach, Ant algorithm, simulation*

1 Introduction

The computational scientist has started to adapt to the grid computing techniques for the past seven years. This has increased the computing power and capability of inter-organization, national and international grid computing infrastructures such

as the e-minerals grid in the U.K, the tera grid in the U.S and the Guhas EGEE [1]. The main aim of grid computing is to integrate clusters into global infrastructures. The grid users need not be aware of the computational resources that are used for executing their jobs and storing their data [1].

Grid computing is emerging as the next generation of parallely distributed computing platform for solving large scale computational and data intensive problems in science, engineering and commerce [4]. It enables the sharing, selection and aggregation of a wide variety of geographically distributed resources including supercomputers, databases, data sources and specialized devices owned by different organizations. Authentication and authorization, secure and reliable file transfer, distributed storage management and resource scheduling across organizational boundaries are the list of problems need to be solved in grid computing area. The Grid users need not be aware of the computational resources that are used for executing their applications and storing their data. The resource allocation to a large number of jobs is hard and much more difficult than the LAN computational environments [2]. The load balancing of the available resources in the computational grid is another important factor. However, different applications have different characteristics, and their demands on the resources may differ greatly. Efficient and application adaptive resource management and scheduling are technical challenges in the Grid.

Grid computing is now being used in many applications that are beyond distribution and sharing of resources. The distributed resources are useful only if the Grid resources are scheduled. Using optimal schedulers results in high performance grid computing, where as poor schedulers produce contrast results. Now, the grid scheduling is a big topic in grid environment for new algorithm model. The scheduling in Grid environment has to satisfy a number of constraints on different problems.

The existing approaches for scheduling in grid applications uses queuing systems or adhoc schedulers that use specific knowledge of the underlying grid infrastructure to achieve an efficient resource allocation. However, there approaches cannot deal with the complexity of the problem due to dynamic nature of the grid. In fact, job scheduling in computational grids is multi objective in its general formulation and therefore optimization approaches that could tackle many conflicting objectives are imperative.

A grid scheduler, often called resource broker, acts as an interface between the user and distributed resources. It hides the complexities of the computational grid from the user. The scheduler does not have full control over the grid and it cannot assume that it has a global view of the grid. The single most challenging issue of the grid scheduler encounters is the dynamicity of resources. Although a resource may be participating in a grid, its main purpose is used by local users of the organization that it belongs to. Therefore, the load on the resource imposes a great strain on grid scheduling.

The grid scheduling consists of three stages [5]. Resource discovery and filtering in the first phase, the resource selection and scheduling according to the certain objective is the second phase and the job submission is the third phase. The third stage includes the file staging and cleanup. High performance computing and high throughput computing are the two different goals of grid scheduling algorithm. The main aim of the high performance computing is to minimize the execution time of the application. To increase the processing capacity of systems over a long period of time is the aim of the high throughput computing. The proposed approach is the high throughput computing.

Grid scheduling is a NP-Complete problem. Heuristic optimization techniques are the best approach to solve NP-complete problem. The four basic heuristic methods for grid scheduling are namely Genetic Algorithm (GA) [7], Simulated Annealing (SA) [10], Ant Colony Optimization (ACO) and Tabu search (TS) [6]. The main focus of this paper is to develop a high throughput scheduling algorithm based on ACO. This technique does the repeat sampling experiments on the model of the system. It uses the stochastic component in the state sampling and/or transition rules. The statistical knowledge about the problem and the estimate of the variables are updated using the above experimental results. The variances in the estimation of the described variables are reduced using the above knowledge repeatedly. In ACO algorithms, the ants try to build the feasible solution to apply the stochastic decision policy repeatedly. This paper implements a modified ant algorithm which results in minimum makespan and maximum resource utilization for the grid scheduling problems. The paper is further organized as follows. In section 2, the related works are discussed. Section 3 contains the problem description. Section 4 contains the result of the proposed method and the advantages of the proposed method, while Section 5 concludes with future direction.

This document can be used as a template for Microsoft Word versions 6.0 or later. You may open this document then type over sections of the document or cut and paste to other document and then use adequate styles. The style will adjust your fonts and line spacing. Please set the template for A4 paper (21 x 29.7 cm). For emphasizing please use italics and do not use underline or bold. Please do not change the font sizes or line spacing to squeeze more text into a limited number of pages.

2 Literature Review

This section reviews a set of heuristic algorithms which has been designed to schedule the meta-tasks in the computational grids. The collection of independent tasks with no data dependencies is called as meta-task. Meta-tasks are mapped on to the available machines statically; each machine in the computational grid executes a single task at a time. For this mapping, it is assumed that the number of machines, ‘m’ and the number of tasks‘t’, are known a priori. A large number of

heuristic algorithms have been designed to schedule tasks to machines on grid computing systems. The eleven commonly used algorithms are listed as follows.

Opportunistic Load Balancing' (OLB) is one of the easiest techniques. For each job, it finds out the next available machine and simply schedules that job to that machine. The machine selection is in an arbitrary manner. In this algorithm, the expected execution time is not taken into the account, so it produces the poor result [5]. But it uses the resources in the balanced way.

The next simplest schedule is User Directed Assignment (UDA). It schedules each job on the resource, which resource will have the best expected execution time for that task. The load will not be balanced across all the available resources. If all jobs will be best expected execution time in one resource than the other resources (consistent) then all the jobs are allocated to that machine only [5]. The next effective approach is to assign each job, in arbitrary order, to the resource on which it is expected to finish earliest. The algorithm calculates the current job's completion time against the list of available machines. This approach is Fast Greedy method. The benefits of OLB and UDA are combined and form the approach Fast greedy [5].

One of the best and simple heuristics method is called Min-min. In this method, compute the minimum completion time of each task with respect to all machines. The task with the overall minimum completion time is selected and assigned to the corresponding node. The currently assigned job is removed from the unscheduled task list and the above process is repeated until all the tasks are scheduled. Here, all jobs have a good chance to select a suitable resource. So this method automatically minimizes the makespan and balances the load to an extent. It is more complex than the UDA. But it produces better solution when compared to UDA [6]. Another heuristics approach is Max-min. Like Min-min, Max-min also calculates the minimum completion time of each job. It selects a job with the overall maximum of minimum completion time [6].

The Genetic Algorithm (GA) is one of the best methods to search the large solution space. This method operates on a population of chromosomes for a given problem. First it generates the initial population randomly. The initial population may be generated by any other heuristic algorithm; if the population is generated by Min-Min then it is called "seeding" the population with Min-Min [11]. Simulation Annealing (SA) is an iterative technique. It finds out only one possible solution at a time for each Meta task. This method probabilistically allows solution to obtain a better search of the solution space based on a system temperature [10]. The Genetic simulated annealing (GSA) is a combination of genetic algorithm and simulated annealing techniques [12]. Tabu search is also a solution space search. It keeps track of the regions which have already been searched. The new search need not repeat a search near this area[6].

A* is a tree search. Initially, the root node has null solutions.. When the tree grows, the intermediate nodes have a partial solution and the leaf nodes have final

solutions. Each and every node has its own cost function. The node with minimum cost function is replaced by its children. When a new node is added into the tree, the tree is pruned by deleting the node with the highest cost function. The above process is repeated until a leaf node is reached [13].

2.1 Comparison of above stated algorithms

The algorithms, OLB, UDA, Max-min, SA, GAS and Tabu, do not produce good results [6]. The remaining algorithm Min-min and A* produce good results of makespan. The makespan of the above mentioned algorithm difference is within 10%. GA is little better than Min-min. The A* produces best and worst results as compared with GA and Min-min in a different situations. Min-min algorithm is faster than GA and A*. Min-min also has some pitfalls. In the Min-min method, too many jobs are assigned to a single grid node and this will lead to system overloading and the response time of the job is not assured. This is the main disadvantage of Min-min method. Load balancing is not considered in the OLB method.

In this paper [8], the scheduler is available in all the resources. The scheduler looks for the best job to be executed in the resource. Here the jobs travel from one location to another to identify the correct resource which it requires. So the traffic in the grid system will be automatically increased. In this paper [9] communication cost is considered to be an important factor.

The grid scheduling problem is a complex one. So, lots of researchers do their research in this area. The main aim of the researches is to find out the optimal solution and to improve the overall system performance. Min-Min, Max-min, fast greedy, tabu search and ant system are some of the heuristic algorithms which create a static environment. Here, they must know the execution time and the workload in advance. In this paper [9], grid simulation architecture using ACO is proposed. The response time and average utilization of resources are used as the evaluation index. In the paper [14], they could improve the performance like job finishing ratio using the ACO algorithm.

2.2 Ant Algorithm

Dorigo M. introduced the Ant algorithm in 1996, which is new heuristics, predictive scheduling algorithm. It is based on the real ants. When an ant looks for food, ant deposits some amount of pheromone on the path, thus making, it is followed by a trail of this substance. If an ant tries to move from one place to another then it encounters a previously laid trail. The ant can detect the pheromone trail and decide with high probability to follow it. This ant also reinforces the trail with its own pheromone. When more ants are following the trail, then the pheromone on shorter path will be increased quickly. The quantity of pheromone on every path will affect the possibility of other ants to select path. At last all the ants will choose the shortest path. In paper [9], the experiment

results show that ant algorithm has produced an optimum solution. The ACO algorithm has been used to solve many NP problems, such as TSP, assignment problem, job-shop scheduling and graph coloring successfully. So the ant algorithm is suitable to be used in Grid computing task scheduling. In the grid environment, the algorithm can carry out a new task scheduling by experience, depending on the result in the previous task scheduling. In the grid computing environment, this type of scheduling is very much helpful. So ant algorithm for task scheduling in Grid Computing, is proposed in this paper.

2.3 Local Search

The heuristic algorithms can often be improved by combining the local search techniques to take the solution to its local optimum in the search space [3]. The local search technique is to define the neighborhood of a solution. In general a solution will have one or more 'problem' resources (those with schedule lengths equal to the makespan of the whole solution). Try to reduce the 'problem' resource makespan as this will immediately reduce the overall makespan of the solution. The neighborhood is a solution of single transfer of a job from the problem resource to any other resources. The local search technique analysis, the neighborhood and the transfer which reduces the maximum schedule length of the two resources is involved the most. The above process is repeated until no further improvement is possible

3 Problem Description

In this study, the grid is composed of number of hosts. Each host has several computational resources. The resources may be homogeneous or heterogeneous. The grid scheduler finds out the better resource of a particular job and submits that job to the selected host. The grid scheduler does not have control over the resources and also on the submitted jobs. Any machine in grid can execute any job, but the execution time differs. The resources are dynamic in nature. As compared with the expected execution time, the actual time may be varied at the time of running the jobs to the allocated resource.

The grid scheduler's aim is to allocate the jobs to the available nodes. The best match must be found from the list of available jobs to the list of available resources. The selection is based on the prediction of the computing power of the resource. So, lots of problems are needed to be solved in this area. The grid scheduler must allocate the jobs to the resources efficiently. The efficiency depends upon two criteria; one is makespan and the other is flow time. These two criteria are very much important in the grid system. The makespan measures the throughput of the system and flow time measures its QoS. The following assumptions are made before discussing the algorithm. The collection of independent tasks with no data dependencies is called as a meta-task. Each

machine executes a single task at a time. The meta-task size is one and the numbers of machines are 'm'.

The ant based algorithm is evaluated using the simulated execution times for a grid environment. Before starting the grid scheduling, the expected execution time for each task on each machine must be estimated and represented by an ET matrix. Each row of ET matrix consists of the estimated execution time for a job on each resource and every column of the ET matrix is the estimated execution time for a particular resource of list of all jobs in the job pool. ETij is the expected execution time of task t_i and the machine m_j. The time to move the executables and data associates with the task ti includes the expected execution matrix ET_{ij}. For this algorithm, it is assumed that there are no inter-task communications, each task can execute on each machine, and the estimated expected execution times of each task on each machine is known.

The ET matrix will have N x M entries, where N is the number of independent jobs to be scheduled and M is the number of resources which is currently available. Each job's workload is measured by million of instructions and the capacity of each resource is measured by MIPS.

Definition 3.1 *The Ready time (Ready $_m$) indicates the time resource 'm' would have finished the previously assigned jobs. The completion time of i^{th} job on the j^{th} machine is*

$$CT_{ij} = Ready_j + ET_{ij}$$

$Max(CT_{ij})$ is the makespan of the complete schedule. Makespan is used to measure the throughput of the grid system. In general the existing heuristic mapping can be divided into two categories. One is on line mode and the other one is batch mode. In the on line mode, the scheduler is always in ready mode. Whenever a new job arrives to the scheduler, it is immediately allocated to one of the existing resources required by that job. Each job is considered only once for matching and scheduling.

In the batch mode, the jobs and resources are collected and mapped at prescheduled time. In this mode, it takes better decision because the scheduler knows details of the available jobs and resources. The proposed algorithm is also a heuristic algorithm for batch mode.

The result of the algorithm will have four values (task, machine, starting time, expected completion time). The number of jobs available for scheduling is always greater than the available number of machines in the grid. The machine M_j's free time will be known using the function free(j).

Definition 3.2 *The starting time of job t_i on resource M_j is*

$$B_i = free(j) + 1$$

Then the new value of free(j) is the starting time plus ET_{ij}.

Definition 3.3 *A minimization function F and the heuristic information ηi is used to find out the best resource*

$$F = \max(\text{free}(i))$$

$$\eta_i = \frac{1}{\text{free}(j)}$$

Using the definition 3.3 the highest priority machine is found which is free earlier. Here three to four ants are used. Each ant starts from random resource and task (they select ET_{ij} randomly j^{th} resource and i^{th} job). All the ants maintain a separate list. Whenever they select next task and resource, they are added into the list. At each iteration the ants calculate the minimized function 'F_k (kth ant)' and the pheromone level of the elements of the solutions is changed by applying following updating rule

Definition 3.4 *The pheromone level update (*τ_{ij}) is

$$\tau_{ii} = \rho\, \tau_{ii} + \Delta\tau_{ii}$$

where,

$$\Delta\tau_{ij} = \frac{1-\rho}{F_k}$$

The rule $0 < \rho < 1$ models evaporation and $\Delta\tau_{ij}$ is an additional pheromone and it is different for different ACO algorithms. In this algorithm two set of tasks are maintained. One is set of scheduled tasks and the other is set of arrived and unscheduled tasks. The algorithm starts automatically, whenever the set of scheduled jobs become empty.

According to the paper [5], the first task to be performed and the machine in which it is performed is chosen randomly. Next, the task to be run and the machine in which it is to be run is computed by the definition 3.5

Definition 3.5 *To select the next, the task to be run and the machine in which it is to be run is computed by*

$$P_{ij} = \frac{\tau_{ij} \cdot \eta_{ij}}{\Sigma\, \tau_{ii} \cdot \eta_{ii}}$$

where

η_{ij} is the attractiveness of the move as computed by some heuristic information indicating a prior desirability of that move.

τ_{ij} is the pheromone trail level of the move, indicating how profitable it has been in the past to make that particular move(it represents therefore a posterior indication of the desirability of that move).

P_{ij} is the probability to move from a state i to a state j is depending on the combination of above two values

The above definition 3.5 has the disadvantage, that all the columns in the probability matrix have the same probability value. This decides the best resource, but the task is chosen to be the first non zero value of the column. In paper [5], they use one ant. To overcome this disadvantage a new algorithm is proposed. In this method, the probability matrix (P_{ij}) is modified and several ants are used. The number of ants used is less than or equal to the number of tasks. From all the possible scheduling lists find the one having minimum makespan and uses the corresponding scheduling list.

Here two kinds of ET matrices are formed, first one consists of currently scheduled jobs and the next consists of jobs which have arrived but not scheduled. The scheduling algorithm is executed periodically. At the time of execution, it finds out the list of available resources (processors) in the grid environment, form the ET matrix and start scheduling. When all the scheduled jobs are dispatched to the corresponding resources, the scheduler starts scheduling over the unscheduled task matrix ET. This is guaranteed that the machines will be fully loaded at maximum time.

3.1 Scheduling Algorithm

In paper [15,16],

Definition 3.6 *The P_{ij} 's value has been modified to include the ET_{ij} in the paper [15,16]*

$$P_{ij} = \frac{\tau_{ij} \cdot \eta_{ij(1/ET_{ij})}}{\Sigma\ \tau_{ij}\ \eta_{ij}\ (1/ET_{ij})}$$

The inclusion of ET_{ij} execution time of the i^{th} job by the jth machine(predicted) in the calculation of probability, that the j^{th} machine will be free, has shown a positive result in performance improvement. This improvement is in terms of the decrease in makespan time. The result produced by the algorithm is little better than the algorithm in the paper [5]. The paper [15,16] used the same formula (8) but the only difference was, in paper [16], they used the simulation bench mark problem as the data. Further more, instead of adding ET_{ij}, execution time of the i^{th} job by the j^{th} machine (predicted), in the calculation of probability, adding CT_{ij}, like formula (8) still produces better results.

Definition 3.7 *The P_{ij} 's value has been modified to include the CT_{ij}*

$$P_{ij} = \frac{\tau_{ij} \cdot \eta_{ij}\,(1/CT_{ij})}{\Sigma\ \tau_{ij} \cdot \eta_{ij}\,(1/CT_{ij})}$$

where

$$CT_{ij} = free[j] + ET_{ij}$$

The proposed algorithm starts only if the unscheduled tasks (n) set does not empty. The initialization part of the algorithm is as follows. The algorithm collects the details about available resources (m) from the Resource Information center. Next it finds out the value of expected execution matrix ET_{nxm}. The initial value of pheromone evaporation value ρ is 0.05. The Pheromone deposit is (τ_0) initial value is 0.01. The number of ants(k) used in the proposed algorithm is 2. The variable free is a one dimensional matrix of size m and the value is zero because the proposed algorithm assumed that all the resources are available only for the gird scheduling.

After the initialization, the proposed algorithm runs as follows.

Algorithm 1 Algorithmic frame for a Ant Algorithm

For each Ant do
  Randomly select **$Task_i$** and **$resource_j$**
  Add (**$Task_i$** , **$resource_j$** , **free[j], free[j]+ET_{ij}**) to the output list.
  Remove the **$Task_i$** from the unscheduled list to scheduled list
  For each **$Task_i$** in the unscheduled list do
    Calculate the heuristic information (**η_{ij}**)
    Find out the current pheromone trail value (τ_{ij})
    Update the pheromone trail matrix where,

$$\tau_{ij} = \rho\, T_{ij} + \Delta\tau_{ij}$$

    Calculate the Probability matrix where,

$$P_{ij} = \frac{T_{ij} \cdot \eta_{ij} (1/CT_{ij})}{\Sigma\, T_{ij} \cdot \eta_{ij} (1/CT_{ij})}$$

    Find out the highest value of **P_{ij}** and add (**$Task_i$** , **$resource_j$** , **free[j], free[j]+ET_{ij}**) to the output list.
    Remove the **$Task_i$** from the unscheduled list
    Modify the resource free time

$$\textbf{free[j]} = \textbf{free[j]} + \textbf{ET}_{ij}$$

  done
  Find out the best feasible solution by analyzing of all the ants scheduling list
done

The output of the above algorithm 1 is passed to the Algorithm 2. The Algorithm 2 uses the local search technique and reduces the over all makespan further.

Algorithm 2 Algorithmic frame for a local search algorithm

S = current solution
$\bar{s}$ = NULL
Repeat until $\bar{s}$ <> S
  Find out the problem resource's and problem resource's problem job

Create neighbor of S ($\bar{s}$) to transfer the problem job to some other resource
If $\bar{s}$ is better quality than S then
S = $\bar{s}$
End repeat
The output is in S

Table 1. Makespan values for benchmark [4] (in arbitrary time units)

	Max-Min	Min-Min	AWOEC	AE	AE+LS	AC	AC+LS
u-c-hihi	12385672.00	**8460675.50**	26886302.05	13329540.99	9024099.15	11970785.32	8656648.16
u-c-hilo	204054.59	164022.44	198633.09	**103470.66**	**88226.95**	**107360.95**	**84491.79**
u-c-lohi	392566.69	**275837.34**	715846.45	362400.30	295571.47	353956.54	281904.20
u-c-lolo	6945.36	5546.26	5563.69	**2786.58**	**3348.26**	**3698.49**	**2798.02**
u-i-hihi	8018378.10	3513919.25	25597881.49	**3748883.61**	**3731727.82**	**3549854.64**	**3535831.94**
u-i-hilo	151923.83	80755.68	254829.33	**37657.48**	**37488.95**	**35639.59**	**35490.82**
u-i-lohi	251528.85	120517.17	846602.99	122833.39	122027.96	**117714.01**	**117244.76**
u-i-lolo	5177.70	2779.09	8492.14	**1251.86**	**1250.01**	**1177.98**	**1172.98**
u-s-hihi	9208811.50	**5160343.00**	27063804.32	7941120.75	5331396.36	7678387.39	5306503.15
u-s-hilo	172822.70	104540.73	220030.62	**60377.62**	**52668.41**	**60107.27**	**50765.09**
u-s-lohi	282085.73	**140284.48**	768171.59	168550.24	214605.62	209563.92	172168.15
u-s-lolo	6232.24	3867.49	7101.91	**1967.03**	**1742.51**	**1893.95**	**1714.36**

4 Experimental Results

Here the results are compared with the Max-Min[17], Min-Min, existing Ant algorithm [5] without execution time and completion time(AWOEC), the ant algorithm with execution time(AE)[15,16] the ant algorithm with completion time (AC) the proposed ant algorithm AE with local search (AE+LS) and the ant algorithm AC with local search. To simulate the various heterogeneous problems, different types of ET matrix using benchmark simulation model [6] are defined.

The ET matrix considers three factors: task heterogeneity, machine heterogeneity and consistency. The task heterogeneity depends upon the various execution times of the jobs. The two possible values are defined high and low. Similarly the

machine heterogeneity depends on the running time of a particular job across all the processors and again has two values: high and low. In the real scheduling, three different ET consistencies are possible. They are consistent, inconsistent, and semi consistent.

The instances of bench mark problems are classified into twelve different types of ET matrices. Each consists of 100 instances. The instances depend upon the above three factors as task heterogeneity, machine heterogeneity and consistency. Instances are labeled as u_x_yyzz.k where

u - is a uniform distribution, used to generate the matrix.
x – is a type of consistency
c- consistent
s-semi consistent
i-inconsistent

An ET matrix is said to be consistent if a resource Ri execute a task Ti faster than the resource Rk, and Ri executes all other jobs faster than Rk. An ET matrix is said to be in-consistent if a resource Ri executes some jobs faster than Rj and some slower. A semi consistent ETC matrix is an inconsistent matrix which has a sub matrix of a predefined size.

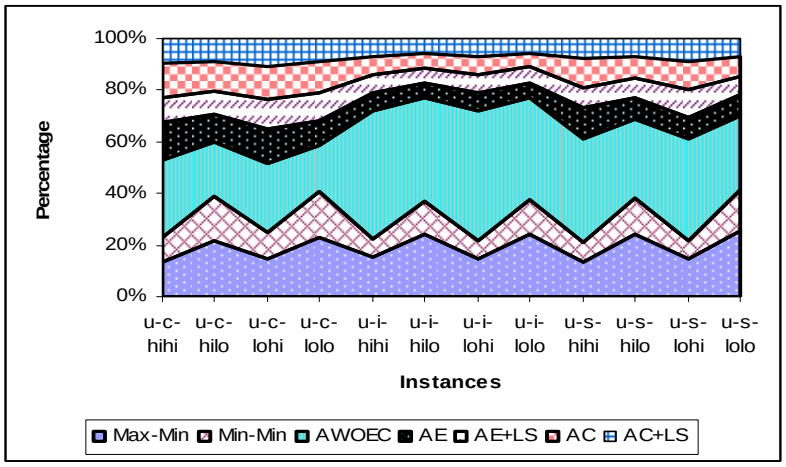
yy- is used to indicate the heterogeneity of the jobs(hi – high, lo-low)
zz-is used to indicate the heterogeneity of the resources (hi-high, lo-low)

All the instances consist of 512 jobs and 16 machines. For each method the makespan is computed. It allows a fair comparison of the presented methods.

5 Performance Evaluation

The computation results of makespan obtained from the Max-Min, Min-Min, AWOEC, AE, AC, AE+LS and AC+LS, is given in table 1. The stacked area chart of the makespan value is shown in Fig 1.

Figure 1. Graphical representation of makespan values (arbitrary time units)



The Min-min, Max-min, AWOEC, AE, AE+LS, AC, AC+LS values are stacked one on the top of the other to sum up to 100%. This above Figure shows the relative size of each algorithm representing its contribution to the total. Max-Min and AWOEC's makespan values are higher than the other algorithms.

Table 1 show that AE, AE+LS, AC, AC+LS produce better results for High Task Low Machine and Low Task High Machine instances. The proposed algorithms AE+LS and AC+LS yield better results for seven out of 12 and eight out of 12 considered instances. As compared with AE and AE+LS, the AE+LS reduces the makespan further. Like this Comparing AC with AC+LS, AC+LS also reduces the makespan further

Table 2. Percentage of makespan values by AE, AE+LS, AC and AC+LS in comparison with AWEOC (values in %)

	AE	**AE+LS**	**AC**	**AC+LS**
u-c-hihi	-57.55	-6.66	-41.49	-2.32
u-c-hilo	**36.92**	**46.21**	**34.54**	**48.49**
u-c-lohi	-31.38	-7.15	-28.32	-2.20
u-c-lolo	**49.76**	**39.63**	**33.32**	**49.55**
u-i-hihi	-6.69	-6.20	-1.02	-0.62
u-i-hilo	**53.37**	**53.58**	**55.87**	**56.05**
u-i-lohi	-1.92	-1.25	**2.33**	**2.72**
u-i-lolo	**54.95**	**55.02**	**57.61**	**57.79**
u-s-hihi	-53.89	-3.31	-48.80	-2.83
u-s-hilo	**42.24**	**49.62**	**42.50**	**51.44**
u-s-lohi	-20.15	-52.98	-49.38	-22.73
u-s-lolo	**49.14**	**54.94**	**51.03**	**55.67**

The percentage of the makespan values by the AE, AC, AE+LS and AC+LS algorithms when compared to Min-Min is listed in the Table 2. Fig. 2 shows that the AE, AE+LS, AC and AC+LS algorithms do better performance than the Min-Min algorithm for many of the instances.

Figure 2. Graphical representation of makespan values (%) by AE, AE+LS, AC and AC+LS in comparison with AWEOC (values in %)

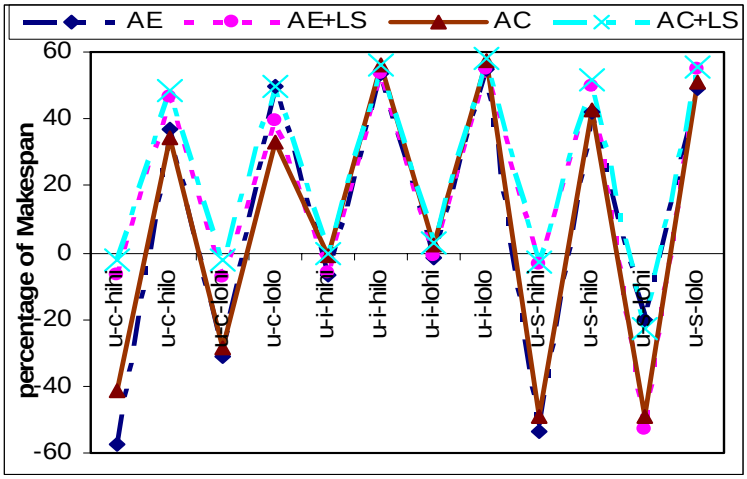
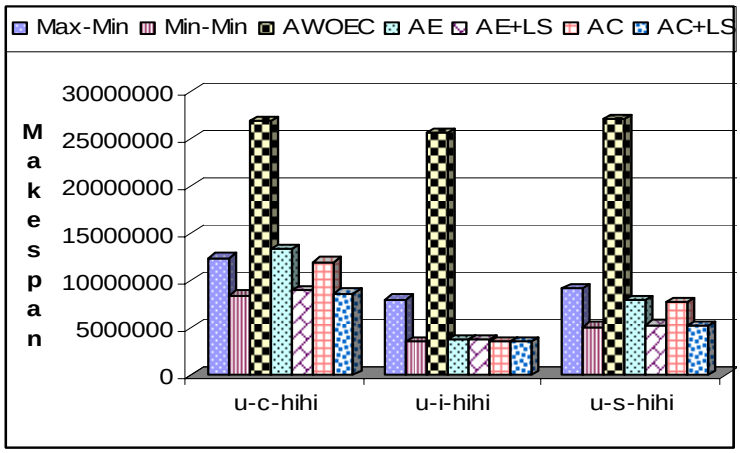


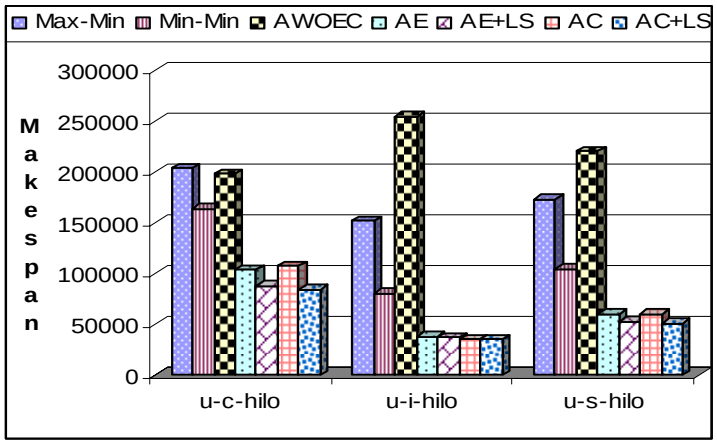
Fig. 3, Fig. 4, Fig. 5, Fig. 6 shows the comparison of Min-min, Max-min, AWOEC, AE, AE + LS, AC and AC+LS algorithms in High Task High Machine, Low Task High Machine, High Task Low Machine, and Low Task Low Machine respectively. The hardware/software configuration used is irrelevant because the execution times are given in their time complexity.

Figure 3. Graphical representation of makespan values of High Task High Machine



As can be seen from Fig. 3, Fig. 4, Fig. 5, Fig. 6 AE, AE+LS, AC and AC+LS obtain the best makespan values as compared to AWOEC. AE, AE+LS, AC and AC+LS are appropriate for consistent, inconsistent and semi-consistent matrices.

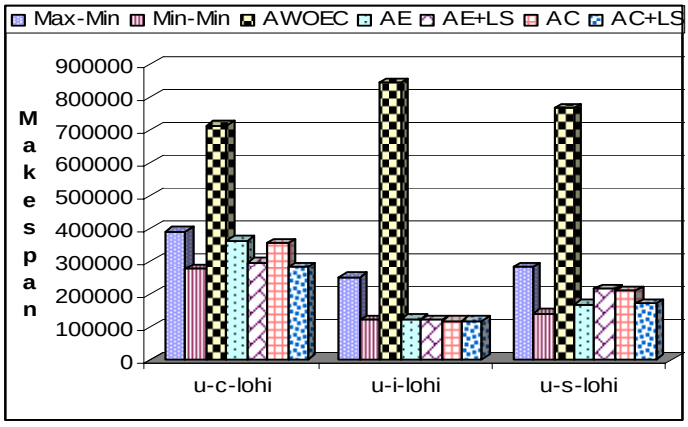
Figure 4. Graphical representation of makespan values of High Task Low Machine



As shown in Fig. 4 and 6, AE, AE+LS, AC and AC+LS show good results of makespan than the Min-Min method. Therefore the AE, AE+LS, AC and AC+LS yields better results for High Task Low Machine and Low Task Low Machine.

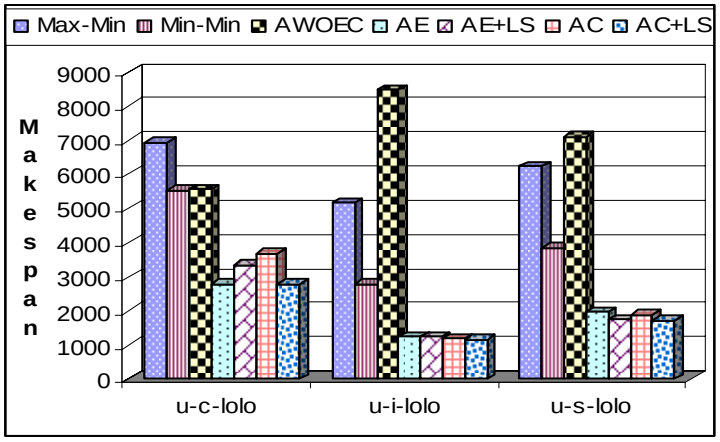
AC and AC+LS produced the best result for inconsistent matrix. AE yielded better result for low task high machine. The percentage of makespan value by the AE, AE+LS, AC, AC+LS algorithms when compared to Min-min decreases for seven out of twelve considered instances approximately.

Figure 5. Graphical representation of makespan values of Low Task High Machine



From the heuristic techniques seen above, the AC, AC+LS Ant algorithm performs sixty eight percentage better than the AWOEC ant algorithm 2.5% better than the algorithm AE, AE+LS in all possible cases on an average. Thus, addition of Local search to AE and (AE+LS) and AC (AC+LS) have shown a positive result in performance improvement. This improvement is in terms of decrease in makespan time.

Figure 6. Graphical representation of makespan values of Low Task Low Machine



6 Conclusion and Future Work

Selecting the appropriate resources for the particular task is one of major challenging work in the computational grid. In this paper, to solve the Grid scheduling problem using Ant algorithm with Local Search is explained. The grid provides a real distributed real time system with no global control for schedulers.

The AC+LS and AE+LS methods take decision depending upon the current environment and are aware of the contexts. The algorithm can adopt the system environment freely at runtime. It allocates the resource optimally and adaptively in the scalable, dynamic and distributes-controlled environment. This allocation is done using the previous information.

In the study, the algorithm is designed and compared to different grid environments. Using ACO good workload balancing results can be obtained. The AC+LS and AE+LS algorithms improve the solution produced by AC and AE by combining them with local search techniques. They consistently find better schedule for several benchmark problems as compared with other techniques in the literature.

In the Grid environment the AC+LS and AE+LS ant algorithm will achieve high throughput as compared with previous ant systems AE[15,16] AWOEC[5] and AC. In this algorithm, the job completion time is one of the major input

parameter. In future, research would consider factors like CPU workload, Communication delay, QoS and so on.

The next research direction is to create Qos based heuristic algorithm for grid computing. The research could also focus on automatically changing the amount of pheromone evaporation and deposit depending upon the performance of the grid system. The techniques used may have to diverge somewhat from those described here, but the results presented here suggest that there is considerable scope for future research in this area.

References

[1] Chapman, C.; Musolesi, M.; Emmerich, W.; Mascolo, C.”Predictive Resource Scheduling in Computational Grids” in Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International Volume , Issue , 26-30 March 2007 Page(s):1 – 10

[2] F. Xhafa, J. Carretero, L. Barolli and A. Durresi. Immediate Mode Scheduling in Grid Systems. International Journal of Web and Grid Services, Vol.3 No.2, 219-236, 2007

[3] G. Ritchie and J. Levine. A fast, effective local search for scheduling independent jobs in heterogeneous computing environments. Technical report, Centre for Intelligent Systems and their Applications, School of Informatics, University of Edinburgh, 2003.

[4] Rajkumar Buyya1, Manzur Murshed, David Abramson and Srikumar Venugopal, “Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm “ Journal of Software—Practice & Experience, Volume 35, No. 5 pp: 491-512,2005 .

[5] Stefka Fidanova and Mariya Durchova, ”Ant Algorithm for Grid Scheduling Problem”, Large Scale Computing, Lecture Notes in Computer Science No. 3743, Springer, germany, pp. 405-412, 2006.

[6] R. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson,M. Theys, B. Yao, D. Hensgen and R. Freund, 2001, “A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems”, Journal of Parallel and Distributed Computing, Vol.61, No.6, pp. 810- 837, 2001.

[7] Aggarwal, M.; Kent, R.D.; Ngom, A , “ Genetic algorithm based scheduler for computational grids” International Symposium on High Performance Computing Systems and Applications, 2005.. Volume15 , No.18 pp: 209 – 215

[8] Li Liu, Yi Yang, Lian Li and Wanbin Shi, "Using Ant Optimization for Super Scheduling in Computational Grid", In Proceedings of the IEEE Asia-pasific Conference on Services Computing, 2006

[9] Zhihong XU, Xiangdan HOU, Jizhou SUN, " Ant- Algorithm-Based Task scheduling in Grid Computing", Montreal, In Proceeding of the IEEE Conference on Electrical and Computer Engineering, pp. 1107-1110, 2003.

[10] Fidanova.S, " Simulated Annealing for Grid Scheduling Problem", International IEEE Symposium on Modern Computing, 2006.

[11] " Lee Wang; Siegel H.J.; Roychowdhury V. P.; Maciejewski A.A., "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach", Journal of parallel and distributed computing 47(1)8-2, nov 1997.

[12] Chen, H. Flann, N.S. Watson, D.W., "Parallel genetic simulated annealing: a massively parallel SIMD algorithm "IEEE Transactions on Parallel and Distributed Systems, Vol.9, No.2 pp.126-136,1998

[13] K. Chow and B. Liu. " On mapping signal processing algorithms to a heterogeneous multiprocessor system". International IEEE Conference of Acoustics, Speech, and Signal Processing pages 1585–1588, May 1991.

[14] H. Yan, X. Shen, X. Li and M. Wu, "An Improved Ant Algorithm for Job Scheduling in Grid Computing", In Proceedings of the IEEE International Conference on Machine Learning and Cybernetics, pp. 2957-2961, 2005.

[15] Kousalya.K and Balasubramanie.P, "Resource Scheduling in Computational Grid using ANT algorithm", In Proceedings of the International Conference on Computer Control and Communications, Pakistan,2007.

[16] Kousalya.K and Balasubramanie.P, "An Enhanced ant algorithm for grid scheduling problem", International Journal of Computer Science and Network Security,Vol. 8, No.4, pp.262-271, 2008.

Kousalya .K received the B.E. and M.E. degrees in Computer Science and Engineering from Bharathiar University, Coimabatore, India, in 1993 and 2001, respectively. She is currently doing her PhD degree in Anna University, Chennai, India. Currently she is an Assistant Professor in the department of Computer Science and Engineering, Perundurai, Tamilnadu. Her areas of interest are Grid Computing, Compiler Design and Theory of Computation. She has published papers in National, International Conferences and in International Journal.

Dr.P.Balasubramanie has obtained his PhD degree in theoretical computer science in the year 1996 from Anna University, Chennai. He was awarded junior research fellow by CSIR in the year 1990. Currently he is a professor in the Department of Computer Science and Engineering, Kongu Engineering College, Perundurai, Tamilnadu. He has published more than 50 research articles in International/National Journals. He has also authored six books. He has guided 3

PhD scholars and guiding 15 research scholars. His areas of interest include theoretical computer science, data mining, image processing and optimization Techniques.