Independent Job Scheduling by Fuzzy C-Mean Clustering and an Ant Optimization Algorithm in a Computation Grid

Tarek Helmy Zeehasham Rasheed

***Abstract*—Grid computing is gaining more significance in the high-performance computing world. This concept leads to the discovery of solutions for complicated problems regarding the diversity of available resources among different jobs in the Grid. However, the major problem is the optimal job scheduling for heterogeneous resources, in which each job needs to be allocated to a proper grid's node with the appropriate resources. An important challenge is to solve optimally the scheduling problem, because the capability and availability of resources vary dynamically and the complexity of scheduling increases with the size of the grid. This paper, therefore, presents a framework which combines the Fuzzy C-Mean clustering with an Ant Colony Optimization (ACO) algorithm to improve the scheduling decision when the grid is heterogeneous. In the proposed model, the Fuzzy C-Mean algorithm classifies the jobs into appropriate classes, and the ACO algorithm maps the jobs to the appropriate resources. The ACO is characterized by ant-like mobile agents that cooperate and stochastically explore a network, iteratively building solutions based on their own memory and on the traces (pheromone levels) left by other agents. The simulation is done by using historical information on jobs in a grid. The experimental results show that the proposed algorithm can allocate jobs more efficiently and more effectively than the traditional algorithms for scheduling policies.**

***Index Terms*—Fuzzy C-Mean, ACO, Job Scheduling.**

I. Introduction

In high-throughput computing, the grid is used to schedule the independent jobs with respect to the dynamically distributed resources [1]. Grid computing is the principle of sharing the computational resources like processors, storage, network & instruments in a secure way. Under this principle, grid computing has faced a lot of problems in acquiring flexible, secure, and coordinated sharing among dynamic collections of resources [1, 3]. The main objective of the scheduler is to maximize the resources utilization. The previous research on scheduling for distributed systems, such as clusters and supercomputers, focused on extracting the maximum throughput from the entire system [4, 5]. Grid scheduling is responsible for resources discovery, resources selection, and job assignment over distributed nodes of the gird. Grid scheduling concentrates on improving response times in a grid containing autonomous resources whose availability varies dynamically with time. The grid scheduler must interact with the local schedulers managing computational resources and must adapt its behavior to the changing resources loads. Thus the scheduling is conducted from the perspective of the application or the user rather than that of the system. Grid scheduling involves a series of challenging tasks. These include: searching for resources in the collection of geographically distributed nodes; and making scheduling decisions according to the required quality of service. A grid scheduler differs from a scheduler for conventional computing systems in several respects. One of the primary differences is that the grid scheduler does not have full control over the grid. More specifically, the local resources are generally controlled not by the grid's scheduler, but by the local scheduler. Another difference is that the grid scheduler cannot assume that it has a global view of the grid. The demand for scheduling is to achieve high-performance computing. It is very difficult to find an optimal resource allocation for specific jobs that minimizes the scheduled length of the jobs. The scheduling problem is a NP-hard problem [6] and it is not trivial.

There are basically two approaches to solve this problem. The first is based on job characteristics, and the second on a distributed resources discovery and allocation system. We have studied the feasibility and the usefulness of applying heuristics and machine learning techniques to this field. We provide a scheduling model [Figure 1] based on Fuzzy C-Mean (FCM) clustering and Ant Colony Optimization (ACO) algorithm for grid scheduling. In this paper, we compare our algorithm with the performance of various job-scheduling algorithms in grid computing environment.

This paper is organized as follows. Section II reviews relevant research. Section III gives a brief overview of FCM clustering and ACO algorithm. Section IV discusses problem description. Sections V and VI present some theoretical aspects of the proposed algorithm. Section VII and VIII discuss the experimental setup and results. Our conclusion and suggestions for future work are given in Section IX.

II. Related Work

In grid computing, there are a lot of important issues, including job scheduling, information service, information security, resource management, routing, and fault tolerance. The job scheduling is a major problem, since it is a fundamental and crucial step in achieving high performance. Job scheduling has been described as a combinatorial

Manuscript received March 9, 2009. This work was supported in part by King Fahd University of Petroleum and Minerals.

Tarek Helmy. Author is with College of Computer Science and Engineering, King Fahd University of Petroleum and Minerals, Kingdom of Saudi Arabia (Tel & Fax +96638601967, helmy@kfupm.edu.sa). On leave from the College of Engineering, Department of Computers Engineering and Automatic Control, Tanta University, Egypt.

Zeehasham Rasheed. Author is with Computer and Communication Information Technology Research (CCITR) at Research Institute, KFUPM, Kingdom of Saudi Arabia, hasham@kfupm.edu.sa.

optimization problem. Scheduling in a grid can be seen as an extension to the scheduling problem on local parallel systems. In general, job scheduling predictions in a grid are dependent on the job's execution time and the job's running time. For example, the prediction engine module in [7] is a part of the scheduler and offers a history-based approach for estimating the run time of job submission.

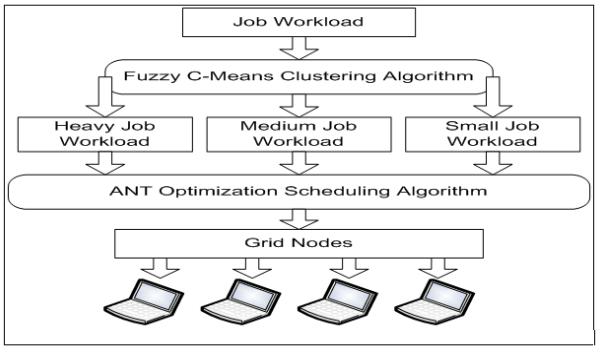

Figure 1: Model for Job Scheduling in Grid Environment

The authors in [8] proposed two modules for predicting the completion time of jobs in a service grid and for applying evolutionary techniques to job scheduling. The problem of estimating a job's run time from historical data has been studied in [9, 10, 11]. All of them adopt the method of making predictions for future jobs by applying different job characterizations to classify similar previous jobs and then using them to make predictions. After detailed analysis, we noticed that those methods have unclear definitions of jobs, i.e. what are the important features used in measuring the similarity of jobs which can be included in the prediction module. Another approach to predict application performance and to detect of unexpected execution behavior has been proposed in [12]. These authors found that the unexpected execution behavior is typically caused by an unanticipated load on the shared grid resources. Similar predicting application's performance on a given parallel system has been the most widely studied in [13, 14]. More recently those studies have been extended to distributed systems [15, 16, 38, 39, 40]. Traditional performance prediction techniques often focus on performance models that are specific to a single architecture or a static set of resources. However, computational grid environments consist of a collection of dynamic, heterogeneous resources and a collection of different jobs. Our approach especially examines the implications of the fact that the characterization of jobs is expected to affect the mentioned resource utilization. Even more interestingly for researchers on performance quality. We use information about static workload data from the standard workload archive [17] and from experiments reported in several publications [18, 19, 20, 21]. Moreover, these workload traces were used for the evaluation of different scheduling strategies for parallel systems [22, 23, 24, 25] and for grid research [26, 27, 28, 29, 30]. These workload traces consist of information about all job submissions on a node for a certain period of time which usually ranges over several months and several thousands of jobs. Therefore, it is reasonable to start with the available workload traces information from the computing centers to evaluate the impact of jobs characterization in grid. Our approach separated the workload data into three classes based on job run-time historical data [Figure 1]. Other algorithms such as Min-Min, Max-Min, Fast greedy Tabu search and Ant system are some of the heuristic algorithms which create a static environment. They must predict the execution time and workload in advance. In [31], the authors have proposed a simple grid simulation architecture using ACO. They used the response time and the average utilization of resources as the evaluation index. In [32] and [33], the authors proposed ACO algorithms, such as job finishing ratio which could improve the performance.

III. OVER VIEW OF FCM AND ACO ALGORITHMS

A. Fuzzy C-Mean Clustering

This paper aims to cluster jobs according to their similarities into groups. Fuzzy C-Mean (FCM) is a famous clustering algorithm for building Fuzzy partitions. FCM will be used in this approach as the basic tool for building job characterizations in grid. The FCM algorithm was introduced by Bezdek [15] as an extension to Dunn's algorithm [16] to generate Fuzzy sets for every observed feature. Fuzzy clustering methods allow for uncertainty in the cluster assignments. Rather than partitioning the data into a collection of distinct sets (where each data point is assigned to exactly one set), Fuzzy clustering creates a Fuzzy pseudo partition, which consists of a collection of Fuzzy sets. Fuzzy sets differ from traditional sets in that membership in the set is allowed to be uncertain. A Fuzzy set is formalized by the following definitions. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of given data, where $x_i \in R_n$ is a set of feature data. The minimization objective function of the FCM algorithm is frequently used in pattern recognition as follows:

$$\min_{(\mathbf{U},\mathbf{V})} \left\{ J_m(\mathbf{U},\mathbf{V}) = \sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij}^{m} D_{ij}^{2} \right\} \quad (1)$$

$$\sum_{j=1}^{c} u_{ij} = 1 \ , \forall j \ , \ i = 1, 2, \ldots n, \ j = 1, 2, \ldots c \quad (2)$$

$$D_{ij}^{2} = \left\| \mathbf{x}_i - \mathbf{v}_j \right\|_{\mathbf{A}}^{2} \quad (3)$$

Where m is any real number > 1, $V = \{v_1, v_2, \ldots, v_c\}$ are the cluster centers. $U = (\mu ij)$ nxc is the degree of membership of vector x_i in cluster k. The value of U must satisfy the condition in equation (3). D_{ik} is the norm Euclidean distance expressing the similarity between any measured data and the center. The cluster centers V can be calculated according to the following equations:

$$\mathbf{v}_j = \left(\sum_{j=1}^{n} u_{ij}^{m} \mathbf{x}_j \Big/ \sum_{k=1}^{n} u_{ij}^{m} \right), \forall j \ , j=1, 2,\ldots c \quad (4)$$

$$u_{ij} = \left[\sum_{k=1}^{c} \left(\frac{D_{ij}}{D_{ik}} \right)^{\frac{2}{m-1}} \right]^{-1} , \forall \ i, \ j \quad (5)$$

The algorithm will stop if $E = \left\| \mathbf{V}_k - \mathbf{V}_{k-1} \right\| < T$, where T is the termination threshold and k is the iteration number.

B. Ant Colony Optimization Algorithm

The ACO algorithm is based upon a heuristic approach and on the behavior of real ants. Each ant deposes the chemical pheromone on its path when it searches for food from its nest. When each ant moves in a particular direction,

the strength of the pheromone increases. With this guidance, other ants can also trail along. This idea inspired the discovery of the ACO algorithm. This algorithm uses a colony of artificial ants that behave as cooperative agents in a mathematical space, where they are allowed to search and reinforce pathways (solutions) in order to find the optimal ones. This approach, which is population-based, has been successfully applied to many NP-hard optimization problems. The ACO is characterized by ant-like mobile agents that cooperate and stochastically explore a network, iteratively building solutions based on their own memory and on the traces (pheromone levels) left by other agents. At regular intervals, a forward ant is launched from a random source node to another random destination node. In its trip, the forward ant will select the next processor using a random scheme that accounts the path selection probabilities, given by the pheromone levels in each neighbor link, and a heuristics value, calculated from the congestion of each neighbor links.

IV. PROBLEM DISCRIPTION

Grid computing is dynamic that it allocates the jobs to the resources effectively. The main aim of the scheduler in the grid is to allocate the jobs to the available nodes with the best available resources. The best match must be allocated from the list of available jobs and from the list of available resources. The selection is based on the prediction of the computing power of the resource [34]. The grid users expect to run their jobs efficiently. The efficiency depends upon two criteria: makespan and flow time. These two criteria are very important in the grid system. Makespan measures the throughput of the system, and flow time measures its QoS [35]. The expected Execution Time (ET) is the expected time to complete the job. This also includes the submit time of each job. The element ET_{ij} of the ET matrix is defined as the amount of time taken to complete the i^{th} job in the j^{th} resource. The jobs are owned by different users, and all jobs are interdependent. All the resources may be dynamically added or removed from the grid. They use the expected time to compute ET in [36]. 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 currently available. In our experiment, processors are taken as resources. The Ready time ($Ready_m$) indicates the time in which the resource 'm' would have finished the previously assigned jobs. The completion time of the i^{th} job on the j^{th} processor/resource is:

$$CT_{ij} = Ready_j + ET_{ij} \quad (6)$$

Max (CT_{ij}) is the makespan of the complete schedule. Makespan is used to measure the throughput of the grid system. The main objective of this algorithm is to minimize the makespan. In general the existing heuristic mapping can be divided into two categories: on-line mode and batch mode. In the on-line mode, the scheduler is always ready. 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 a prescheduled time. The batch mode produces better decisions because the scheduler knows the full details of the available jobs and resources. The proposed algorithm is also a heuristic algorithm for the batch mode. The result of the algorithm will have four values (task, node, starting time, expected completion time). The number of jobs available for scheduling is always greater than the number of nodes available in the grid. The node M_j free time will be known by using the function free (j). The starting time of job t_i on resource M_j is:

$$B_i = \text{free (j)} + 1 \quad (7)$$

Then the new value of free (j) is the starting time + ET_{ij}. In the algorithm, the minimization function is used in order to find out the best resource:

$$F = \max (\text{free (j)}) \quad (8)$$

And use the following heuristic information is used:

$$\eta_{ij} = 1 / \text{Free(j)} \quad (9)$$

Formula # 9 is used to find out the highest priority node which is free earlier. All the ants are maintaining a separate list. Whenever they select the next task and resource, they are added into the list. The ants calculate the minimize function 'F_k (k_{th} ant)' and the pheromone trail updates the value:

$$\triangle T_{ij} = 1\text{-}\rho / F_k \quad (10)$$

In this algorithm, two set of tasks are maintained: the scheduled tasks and newly arrived & unscheduled tasks. The algorithm starts automatically, whenever the set of scheduled jobs become empty. According to [37], the first task to be performed, and the machine in which it is performed are chosen randomly. Next, the task to be run and the node in which it is to be run are computed by the following formula:

$$P_{ij} = T_{ij} \cdot \eta_{ij} / \Sigma T_{ij} \cdot \eta_{ij} \quad (11)$$

- η_{ij} is the attractiveness of the move as computed by some heuristic information indicating a prior desirability of that move;
- T_{ij} is the pheromone trail level of the move, indicating how profitable it has been in the past to make that particular move;

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

V. THORITICAL ANALYSIS

The goal of the proposed scheduling algorithm is to minimize the total execution time of jobs. As the scheduling is performed statically, all necessary information about the jobs in the grid and the processors in the system is assumed to be available a priori. For the heterogeneous environment, we used a hybrid approach in which grouping of jobs is done using FCM clustering algorithm and affinity between jobs and processors is calculated using ACO algorithm. Essentially, the expected running time of each job on each processor must be known, and this information can be stored in an expected ET matrix. A row in an ET matrix contains the ET for a single job on each of the available processors, and so any ET matrix will have n x m entries, where n is the number of jobs and m is the number of resources or processors. In order to simulate various possible heterogeneous scheduling problems as realistically as possible, we define different types of ET matrix for our experiments: execution time, submission time, load balancing and fault tolerance. The task heterogeneity in our experiment is defined as the processors are not identical, and each processor can take a differing amount of time to process any given job with respect to available resources

A. Finding Job Clusters

In our approach FCM clustering algorithm is used as the basic tool for building jobs characterization in grid environment. Initially, we have given unlabeled dataset of jobs represented by X = {x1, x2, x3…, xn} where n is the number of jobs in X. Also each xk ε Rp where p is the number of features in each input vector. For our approach we used completion time, submission time and resources required as features. As far as clusters are concerned, we used three clusters of jobs as heavy, medium and small workload. They are represented as V= {v1, v2, v3} ε Rp where vi is the ith cluster center. After the initialization, next step is the calculation and optimization of objective function represented in equations (1, 2, 3 and 5). The objective function represents the degree of membership value of each job against each cluster and is calculated by using Euclidean distance. The cluster centers are calculated by using aggregated mean represented in equation (4). The cluster centers are updated in every iteration, and the algorithm continues until $E = \|\mathbf{V}_k - \mathbf{V}_{k-1}\| < T$, where T is the termination threshold, and k is the iteration number. After that, these workloads are sent to the ACO algorithm for optimization.

B. Defining the Pheromone Trial

The fact that jobs will run at different speeds on different processors means that this problem cannot be approached as the same sort of grouping problem as for the homogeneous case. However, we can exploit this fact to use what is perhaps the more intuitive pheromone trail definition that certain jobs may have certain affinities with certain processors, and so it would be useful to store information about good processors for each job. The pheromone value T_{ij} is selected to represent how profitable is to schedule a particular job i onto a particular processor j. For the first time, all processors have value 0.5, means every job has the same benefit or 50 percent profit running on any processor. After the heuristic and the fitness functions are calculated to produce the final pheromone matrix. The pheromone matrix will thus have a single entry for each job-processor pair in the problem.

C. The Heuristic and Fitness Functions

The min-min heuristic is a very effective algorithm for this problem. It suggests that the heuristic value of a particular job should be proportional to the minimum completion time of the job, that is the time a job i can be expected to finish on a processor j. Contrary to conventional ACO algorithm which was using only one ant for heuristic and fitness function, we are calculating probability matrix (Pk_{ij}) by using k ants. The minimum completion time of a job i on a processor j is used for the heuristic function. The resulting η_{ij} function use by the ants is defined in equation (9). If the job i has minimum completion time on processor j, then by using η_{ij} for calculating probability in equation (11), we can find the best processor j for that job i which can complete job i in minimum time span. The same procedure is done by each ant having their own heuristic and fitness function values.

D. Updating the Pheromone Value

For updating the pheromone value, ants should be allowed to share information about good solutions for a policy. Allowing only the best ant to leave pheromone after iterating makes the search much more aggressive, and significantly improves the performance of ACO algorithms. Using equations (10, 11), each ant follows the same pheromone update policy for each pair of job i and processor j in their own pheromone matrix.

$$T_{ij} = \rho\, T_{ij} + \triangle T_{ij}$$

Where ρ. is a parameter which defines the pheromone evaporation rate and $\triangle T_{ij}$ is the pheromone trial value.

E. Building a Solution

A simple strategy, following the minimum execution time approach, would be to allocate each job i, in arbitrary order, to a processor j picked probabilistically with respect to the pheromone value between i and j, and the execution time of completion of i on j (a lower value is preferable). The solution building technique used for this ACO approach is an attempt to follow the concept of the best heuristic method. First, the processor j which completes a job i earliest is established. A job i then picked to be scheduled next based on the pheromone value between job i and processor j. The probability of selecting job j to be scheduled next is given by equation (11). At this stage, we have k probability matrices one for each ant. A job is then selected based on the highest probability among all the matrices, and the chosen job i is then allocated to processor j. This process is repeated until all jobs have been scheduled and a complete solution has been built.

In [37], the algorithm uses only one ant. To overcome this disadvantage, a new algorithm is proposed. In this method, the probability matrix (Pkij) is modified by using several ants (k ants) and the number of ants used is less than or equal to the number of tasks. From all the possible scheduling lists, the one having the minimum makespan is found and that ant's scheduling list is used. So, at the time of execution, the scheduler finds the list of available resources (processors) in the grid, forms the ET matrix, and starts scheduling. The steps for the proposed algorithm are as follows:

1. Find the classification of workload by FCM clustering.
2. Collect all necessary information of jobs (n) and resources (m) of the system in the ET matrix and the submit Time matrix (size should be m x n).
3. Set all the initial values, ρ = 0.05 (pheromone evaporation value), T = 0.5 (initial pheromone deposit value), Free = 0 (one dimensional matrix of size m), k = m (k is the number of ants, and m is the number of resources).
4. For each ant (to prepare the scheduling list) do the steps # 5 and 6.
5. Select the task (i) and resource (j) randomly.
6. Repeat the following until all jobs are executed.
   a. η_{ij} =1 / Free(j)

b. Calculate the pheromone trail value $\triangle T_{ij} = 1-\rho / F_k$, where $F_k = \max(\text{free}(j))$

c. Update the pheromone trail matrix $T_{ij} = \rho T_{ij} + \triangle T_{ij}$

d. Calculate the Probability Matrix $P_{ij} = T_{ij} . \eta_{ij} / \Sigma T_{ij} . \eta_{ij}$

e. Select the highest probability i and j, (the next task i to be executed on resource j)

7. Find the best feasible solution by using the scheduling list of all the ants. Best feasible solution leads to the minimum makespan time of nodes and to the minimum average waiting time of jobs.

Load balancing and fault tolerance are checked during runtime. If any load is overloaded, jobs are swapped from one node to another. Also, if any node fails, all the remaining jobs are distributed among other nodes.

VI. Experimental Setup and Results

In the experiments, we used a workload data from a standard workload archive [17]. These data consists of 1,000 jobs, from which 500 are randomly selected for the experiment. Each job's record has 18 attributes. However, we focused on the execution time and submission time of each job. In the experiments we assumed that each job is allowed to run in each node by using a space-sharing mechanism. In the space-sharing mechanism, each processor can serve only one job at a time. We simulated 10 different performance nodes in the grid. The experiments were conducted in five parts. The first part focused only on the execution time of jobs and all the remaining parts treat both execution and submission time. The second part took care of the execution time as well as the submission time of jobs. The third part supports load balancing between the nodes, the fourth part of the experiment supports fault tolerance and the fifth part focused on the affect of the number of users. The experiments showed the classification of jobs workload into three groups: heavy, medium and light workload [Figure 2]. Figure 3 shows the job membership functions given by using FCM clustering algorithm.

In the experiments, the jobs in workload data are allocated to three classes, each with number of jobs shown in Table 1. After classifying of the work load, the workload is given to the proposed ACO algorithm for grid scheduling. In our experimental testing, we used 10 heterogeneous nodes and 500 tasks. For the performance measure, we evaluated the completion time of each node along with the waiting time of each individual job. We compared the results of the ACO algorithm with the three traditional job scheduling algorithms: First-Come-First-Served (FCFS), Largest Job First (LJF) and Shortest-Job-First (SJF).

TABLE 1: WORKLOAD CLASSES

No	Class	Total
1	Heavy Job Workload	72
2	Medium Job Workload	66
3	Small Job Workload	362

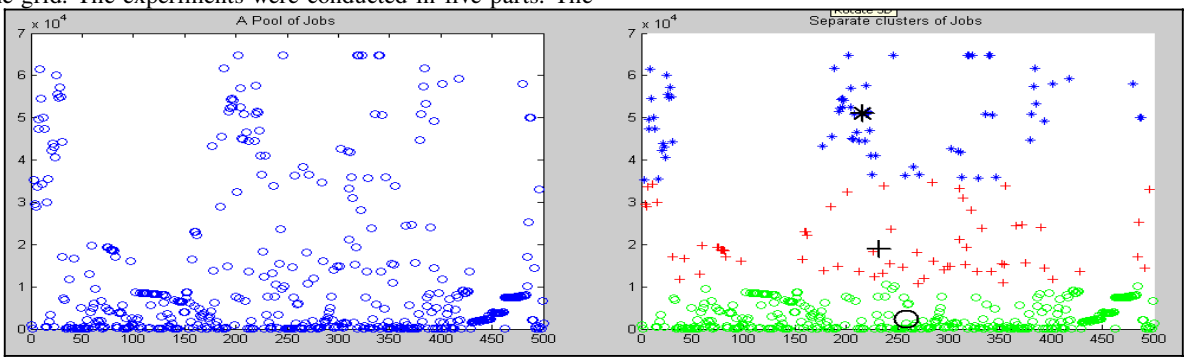

Figure 2: (a) Pool of Jobs (b) Separate Clusters of Job

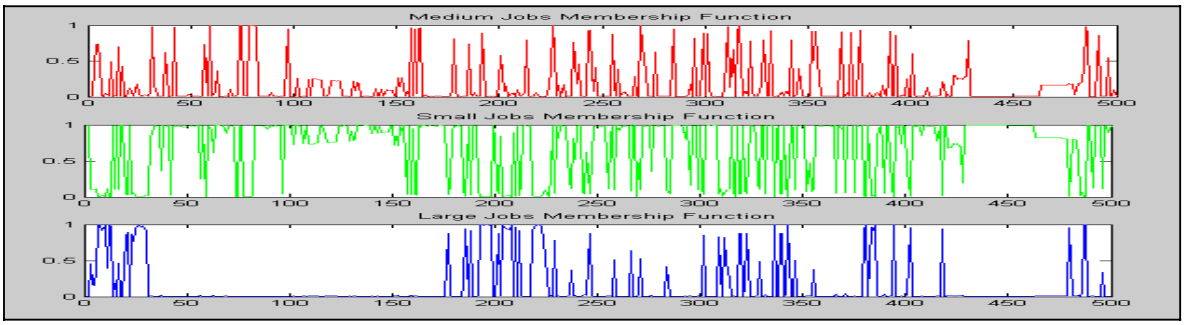

Figure 3: Membership Function of Workload

A. *Jobs with Execution Time only*

The results for the first time, where we consider only the execution time of job are shown in Figures 4 and 5. In Figure 4, we can see that all the nodes have minimum completion time when using the ACO algorithm as compared to others. This completion time includes the execution time and the waiting time of all jobs at their nodes. Similarly, Figure 5 shows that nearly all the jobs have less waiting time than SJF, LJF and FCFS.

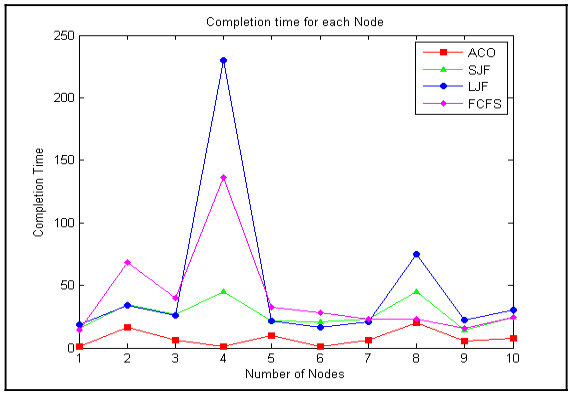

Figure 4: Completion Time of each Node when only the Execution Time is considered

B. *Jobs with Execution and Submission Time*

For this case, we include the submit time of each job when they arrived. So the total waiting time of each job contains the execution time of all previous jobs plus the time from its submission to the time when it gets a processor. The results are shown in Figures 6 and 7. We can conclude that SJF and the proposed ACO algorithms have similar completion times as compared to other algorithms when the submission time is considered.

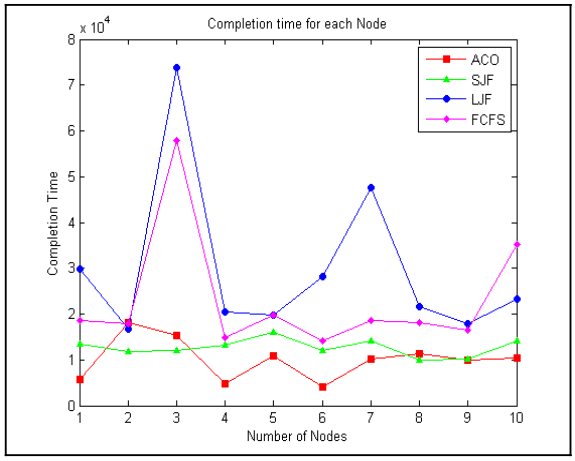

Figure 6: Completion time of each Node when the Execution and the Submission Time are considered

C. *Load Balancing between Nodes*

We implemented a load-balancing mechanism between all nodes. The proposed algorithm continuously monitors each node, and, if any node becomes overloaded, the jobs are migrated from one node to another. The results are shown in Figures 8 and 9. We can see that all the algorithms are trying to balance the load at each node, and here our proposed algorithm ACO has the minimum completion time for each node.

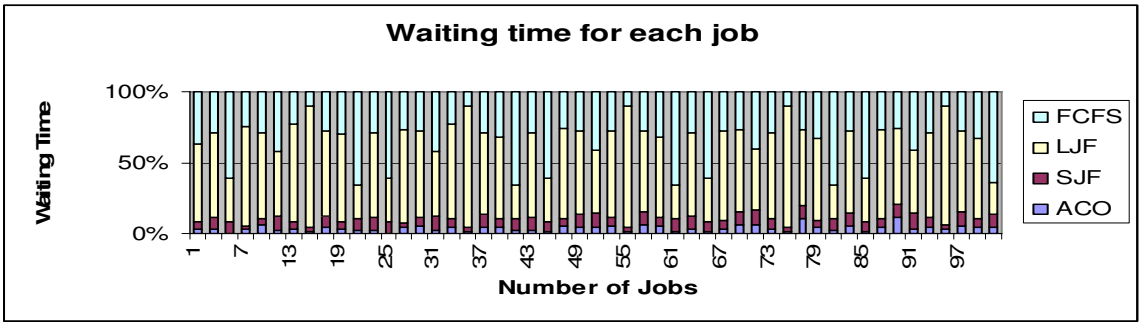

Figure 5: Waiting Time for each job when only the Execution Time is considered

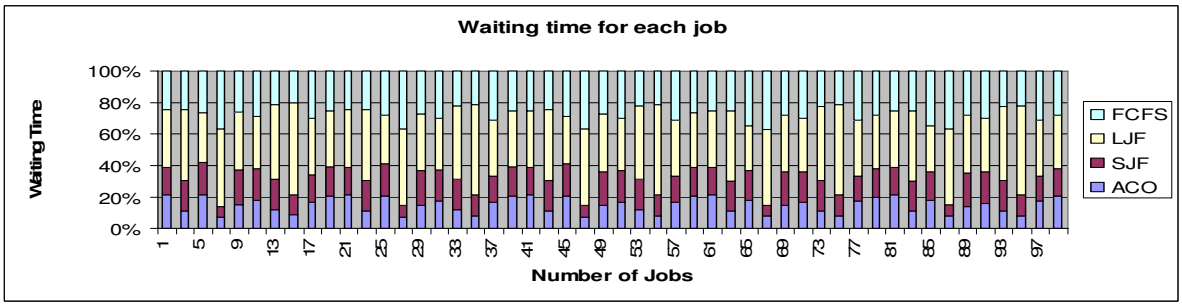

Figure 7: Waiting Time for each job when the Execution and the Submission times are considered

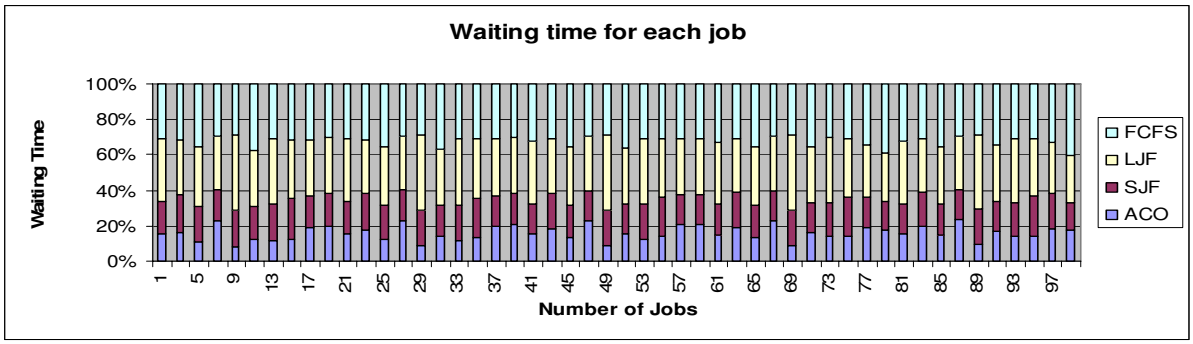

Figure 8: Waiting Time of each job by using Load Balancing Mechanism

D. Fault Tolerance

We considered a scenario in which node 4 failed after some time. In order to support fault tolerance, jobs on node 4 are distributed between different nodes taking care of the completion time as well as the load balancing using checkpoint-restart during runtime. The results are shown in Figures 10 and 11. We can see that the proposed ACO algorithm outperforms all other algorithms despite the failure of nodes. The overall results are stated in Table 2, which shows that the proposed ACO algorithm performs best among different algorithms under different circumstances.

TABLE 2: COMPLETION TIME FOR EACH NODE IN DIFFERENT SCENARIOS

	FCM+ACO Algorithm	SJF	LJF	FCFS
	Time Units			
Execution time only	75.1533	272.137	495.316	405.863
Execution + Submission times	988.7	1274.5	2990.1	2313.8
With Load Balancing Mechanism	838.4	1319.9	1659.2	1640.9
With Fault Tolerance Mechanism	1012.8	1663.4	3488.7	2697.2

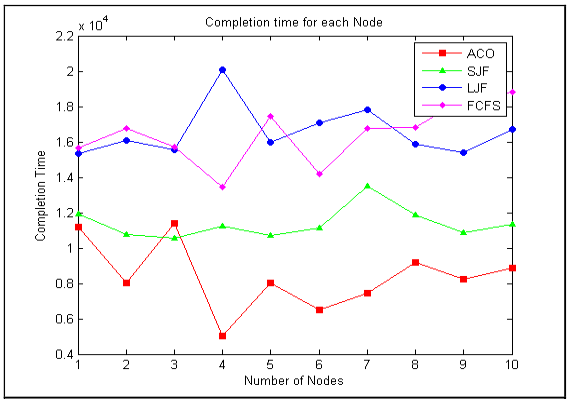

Figure 9: Waiting Time for each Node using Load Balancing Mechanism

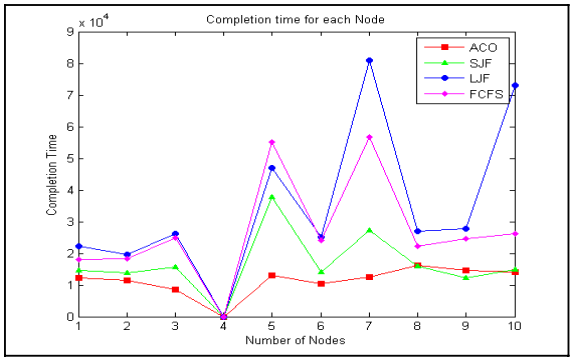

Figure 10: Waiting Time for each Node when one Node Fails

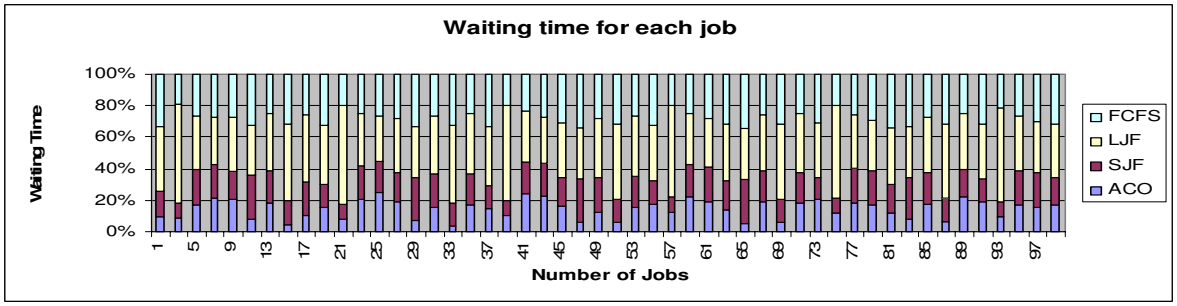

Figure 11: Waiting Time of each job when one Node Fails

VII. Experiment with the Number of Users

We carried out another experiment based on the number of users with four different cases. The number of users is defined by three different configurations as shown in Table 3. For 10 users, the completion time for each scheduler with respect to four different scenarios is stated in the Table 4. For 25 users, the completion time for each scheduler with respect to four different scenarios is stated in the Table5. For 50 users, the completion time for each scheduler with respect to four different scenarios is stated in the Table 6. From Figures 12, 13, and 14 we can infer that the proposed algorithm performs better than all the traditional scheduling algorithms. With the proposed algorithm, the completion time for all jobs is less for all user configurations as well as for all different scheduling algorithms. Also as the number of users and number of jobs increases, the proposed algorithm is performing much better than the traditional scheduling algorithms. For 10 users' configuration, the difference between the completion time of FCM-ACO algorithm and SJF is small, but for 50 users' configuration, the difference between the two algorithms increased. This shows that in real time environment where we have a huge number of users, the proposed algorithm will definitely perform better as compared to others scheduling algorithms.

TABLE 3: NUMBER OF USERS & JOBS IN DIFFERENT SCENARIOS

Number of Users	Jobs per User	Total Number of Jobs
10	10	100
25	10	250
50	10	500

TABLE 4: COMPLETION TIME FOR 10 USERS

	FCM+ ACO	SJF	LJF	FCFS
	Time Units			
Execution time only	17.04	46.76	279.17	144.96
Execution time + Submission time	52.753	95.69	423.04	398.88
With Load Balancing Mechanism	21.319	40.292	860.14	749.38
With Fault Tolerance Mechanism	206.73	250.67	1143.3	1258.4

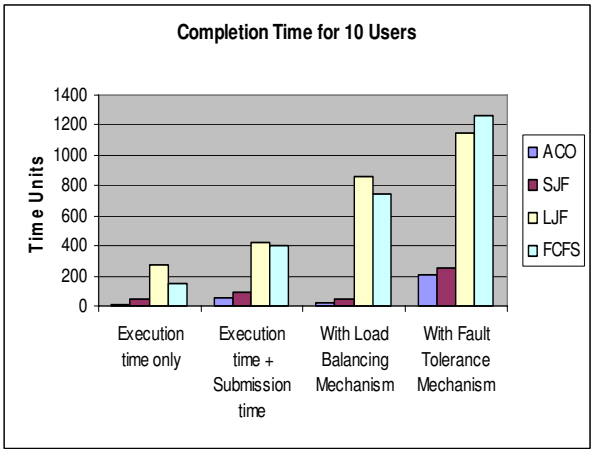

Figure 12: Completion Time for 10 Users

TABLE 5: COMPLETION TIME FOR 25 USERS

	FCM+ ACO	SJF	LJF	FCFS
	Time Units			
Execution time only	43.23	244.80	366.12	300.75
Execution time + Submission time	32.48	104.96	512.10	280.12
With Load Balancing Mechanism	129.05	230.62	552.06	381.26
With Fault Tolerance Mechanism	122.60	208.52	444.82	385.27

TABLE 6: COMPLETION TIME FOR 50 USERS

	FCM+ ACO	SJF	LJF	FCFS
	Time Units			
Execution time only	75.15	272.13	495.31	405.86
Execution time + Submission time	988.7	1274.5	2990.1	2313.8
With Load Balancing Mechanism	838.4	1319.9	1659.2	1640.9
With Fault Tolerance Mechanism	1012.8	1663.4	3488.7	2697.2

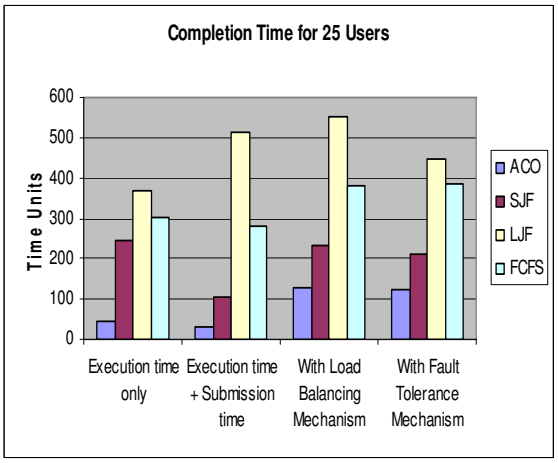

Figure 13: Completion Time for 25 Users

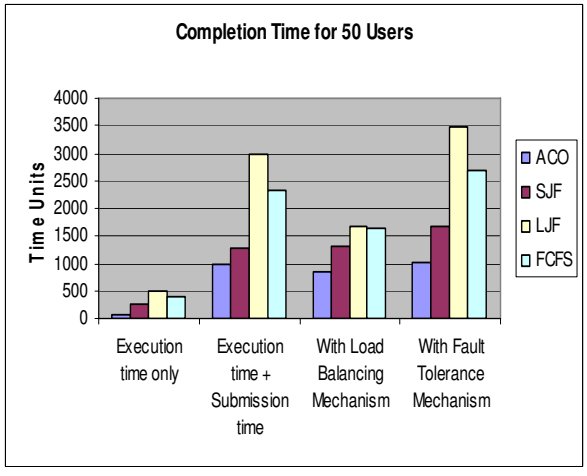

Figure 14: Completion Time for 50 Users

VIII. CONCLUSION AND FUTURE WORKS

We have studied the job scheduling for a grid environment as a combinatorial prediction and optimization problem. We have proposed an intelligent scheduling algorithm in a grid which uses the FCM clustering technique for predicting three classifications of job workload and the ACO algorithm for allocating them to different grid nodes. The proposed hybrid scheduling is very efficient in terms of calculation, because classifying the workload in the first step made the calculation very simple for the ACO algorithm, and it can efficiently schedule each job with respect to its workload class. The experimental results, for each part on the completion time of nodes and the waiting time of each job have shown that the scheduling system using the proposed algorithm outperforms all other algorithms and gives optimal results. Also, from all the traditional scheduling algorithms, only SJF is comparable to the ACO algorithm. Also as the number of users and number of jobs increases, the proposed algorithm is performing much better than the traditional scheduling algorithms. For future work, our simulation environment will include more complex characterization of the constraints for grid scheduling in real time systems.

ACKNOWLEDGMENT

We would like to thank King Fahd University of Petroleum and Minerals for providing the computing facilities. Special thanks to anonymous reviewers for their valuable comments on this paper. Thanks extended to Mr. David Birkett for his help in proofreading the paper.

REFERENCES

[1] L. Faster C. Kesselman S. Tueske "The anatomy of Grid" Internal Journal of Super Computer App. 15(3) 200.

[2] L. Faster C. Kesselman "The Grid blue print for a future computing infrastructure", chapter2 Morgan Kaufmann Publication 1999.

[3] I. Foster and C. Kesselman, Eds., "The Grid 2 Blueprint for a New Computing Infrastructure", San Francisco, CA: Morgan Kaufmann, 2004.

[4] T. L. Casavant, J. G. Kuhl, "Taxonomy of scheduling in general purpose distributed computing", IEEE Transactions on Software Engineering 1988; 14(2).

[5] D. G. Feitelson, L. Rudolph, "Job Scheduling Strategies for Parallel Processing", Lecture Notes in Computer Science, vol. 1659). Springer: Heidelberg, 1999.

[6] D. Fernandez-Baca (1989) "Allocation Modules to processors in a Distributed System", IEEE Transactions on Software Engineering. Vol.15 (11): Pages 1427-1436

[7] A. Arshad, A. Ashiq, B. Julian, R. Cavanaugh, Frank van Lingen, R. McClatchey, Muhammad A. Mehmood, H. Newman, C. Steenberg, M. Thomas, I. Willers, "Predicting the Resource Requirements of a job submission", Proceedings of the Conference on Computing in High Energy and Nuclear Physics 2004.

[8] Y. Gao, H. R. Joshua, Z. Huang., "Adaptive grid job scheduling with genetic algorithms", Future Generation Computer Systems, Volume 21, Issue 1, January 2005, Pp. 151-161.

[9] A. Downey, "Predicting Queue Times on Space-Sharing Parallel Computers", in International Parallel Processing Symposium, 1997.

[10] R. Gibbons, "A Historical Profiler for Use by Parallel Schedulers", Master's thesis, University of Toronto, 1997.

[11] W. Smith, I. Foster, and V. Taylor, "Predicting Application Run Times Using Historical Information", Proceedings of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[12] F. Vraalsen, R.A. Aydt, C.L. Mendes, and D.A. Reed, "Performance Contracts: Predicting and Monitoring Grid Application Behavior", Proceedings of the Second International Workshop on Grid Computing, 2001.

[13] P. Mehra, C. H. Schulbach and Yan, "A Comparison of Two Model-Based Performance-Prediction Techniques for Message-Passing Parallel Programs", In Proceedings of the ACM Conference on Measurement & Modeling of Computer Systems, pp. 181–190, 1994.

[14] R. Saavedra-Barrera, A. J. Smith, and E. Miya, "Performance prediction by benchmark and machine characterization", IEEE Transactions on Computers 38, 12, pp.1659–1679, December 1989.

[15] N. Kapadia, J. Fortes and C. Brodley, "Predictive Application-Performance Modeling in a Computational Grid Environment", In Proceedings of the 8th IEEE Symposium on High-Performance Distributed Computing, pp. 47–54, August 1999.

[16] A. Petitet, S. Blackford, J.Dongarra, B. Ellis, G. Fagg, K. Roche, and S. Vadhiyar, "Numerical Libraries and the Grid: The GrADS Experiments with ScaLAPACK", Tech. Rep. UT CS-01-460, April 2001.

[17] http://www.cs.huji.ac.il/labs/parallel/workload/ (Archive of information regarding the workloads on parallel machines)

[18] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riordan, "Modeling of Workload in MPPs", pp. 94–116. Springer–Verlag, Lecture Notes in Computer Science LNCS 1291, 1997.

[19] C. Ernemann, B. Song, and R. Yahyapour, "Scaling of Workload Traces", Vol. 2862 of Lecture Notes in Computer Science, pp. 166–183, June 24, 2003.

[20] D. G. Feitelson and A. M. Weil, "Utilization and Predictability in Scheduling the IBM SP2 with Backfilling", In Proc. 12th Int'l Parallel and Distributed Processing conference, pages 542–547, CA, 1998.

[21] D. G. Feitelson and B. Nitzberg, "Job Characteristics of a Production Parallel Scientific Workload on the NASA Ames iPSC/860", pp. 337–360. Springer, Berlin, Lecture Notes in Computer Science LNCS 949, 1995.

[22] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour, "On the Design and Evaluation of Job Scheduling Systems", Springer, Lecture Notes in Computer Science, LNCS 1659, 1999.

[23] U. Schwiegelshohn and R. Yahyapour, "Analysis of First-Come-First-Serve Parallel Job Scheduling, "In Proceedings of the 9th SIAM Symposium on Discrete Algorithms, pages 629-638, January 1998.

[24] U. Schwiegelshohn and R. Yahyapour, "Improving First-Come-First-Serve Job Scheduling by Gang Scheduling", pp. 180–198. Springer, LNCS 1459, 1998.

[25] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, and K. C. Sevcik, "Theory and Practice in Parallel Job Scheduling", LNCS, 1291:1–34, 1997.

[26] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "Evaluation of Job-Scheduling Strategies for Grid Computing", pp. 191–202, Springer, Berlin, Lecture Notes in Computer Science LNCS 1971, 2000.

[27] C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "On Advantages of Grid Computing for Parallel Job Scheduling", In Proc. 2nd IEEE/ACM Int'l Symposium on Cluster Computing and the Grid, May 2002.

[28] G. Sumathi and N. P. Gopalan, "Computational Power and Level of Parallelism Based Scheduling for Heterogeneous Grid Environments", International Journal of Computer Sciences and Engineering Systems, Vol.2, No.2, April 2008.

[29] L. Keqin, "Job scheduling and processor allocation for grid computing on meta-computers", Journal of Parallel and Distributed Computing, Volume 65 , Issue 11 pp. 1406 – 1418, November 2005.

[30] Y. Pan, Y. Lee, F. Wu, "Job Scheduling of Savant for Grid Computing on RFID EPC Network," IEEE International Conference on Services Computing (SCC'05) Vol-2, pp. 75-84, 2005.

[31] Z. Xu, X. Hou and J. Sun, "Ant Algorithm-Based Task Scheduling in Grid Computing", IEEE CCECE, 2003.

[32] E. Lu, Z. Xu and J. Sun, "An Extendable Grid Simulation Environment Based on GridSim", volume LNCS 3032, pages 205–208, 2004.

[33] H. Yan, X. Shen, X. Li and M. Wu, "An Improved Ant Algorithm for Job Scheduling in Grid Computing", In Proceedings of the 4th International Conference on Machine Learning and Cybernetics, 18-21 August 2005.

[34] L. Gong., X. H. Sun, E. Waston, "Performance Modeling and Prediction of Non-Dedicated Network Computing", IEEE Transaction on Computer, Vol. 51 9, pp. 1041–1055, 2002.

[35] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, R. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems", 8th IEEE Heterogeneous Computing Workshop, pp. 30–44, 1999.

[36] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. P. Robertson, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", Journal of Parallel and Distr. Comp., Vol. 61, No. 6, pp.810–837, 2001.

[37] S. Fidanova and M. Durchova,"Ant Algorithm for Grid Scheduling Problem", Large Scale Computing", LNCS No. 3743, Springer, pp.405-412, 2006.

[38] L. Liu; Y. Yang; L. Lian; S. Wanbin "Using Ant Colony Optimization for Super Scheduling in Computational Grid" 2006. APSCC 06. IEEE Asia-Pacific Conference on Services Computing, Volume, Issue, Dec. 2006 Page(s):539 – 545.
[39] L. Yaohang "A Bio-inspired Adaptive Job Scheduling Mechanism on a Computational Grid" IJCSNS International Journal of Computer Science and Network Security, VOL.6 No.3B, March 2006.
[40] A. Abraham, H. Liu, W. Zhang, and TG. Chang. Scheduling jobs on computational grids using fuzzy particle swarm algorithm. In Proceedings of 10th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, pages 500-507, 2006.

Tarek Helmy is currently with the department of Information and Computer Science, College of Computer Science and Engineering at King Fahd University of Petroleum and Minerals (KFUPM). On leave from the College of Engineering, Department of Computers Engineering and Automatic Control, Tanta University, Egypt. He received his Ph.D. in Intelligent Systems from Kyushu University, Japan, in 2002. His research interests include Operating Systems, Multi-Agent Systems, Personalized Web services, and Cooperative Intelligent Systems. He has published more than 50 papers in major international journals and conferences in the field of cooperative intelligent agents, artificial intelligence and operating systems. Dr. Helmy is on the program/organizing committee for various international journals/conferences in the field of artificial intelligence, multi-agents, Operating Systems, intelligent and distributed systems.

Zeehasham Rasheed is working as a Research Engineer in Computer and Communication Information Technology Research (CCITR) at Research Institute KFUPM. He completed his M.S in Computer Science from Information and Computer Science Department, KFUPM in 2009. His research interests include Artificial Intelligence, Pattern Recognition, Data mining, Computer Vision, Bioinformatics and Mobile Programming. After completing B.E in Computer and Information System Engineering from N.E.D University of Engineering and Technology, Pakistan, he worked as a Software Engineer for Plexus Private Limited for one year where he was assigned for Software Testing and Software Development.