Enhanced Genetic Algorithm Based Load Balancing in Grid

Sandip Kumar Goyal and Manpreet Singh

**Department of Computer Science & Engineering, M. M. Engg. College,
M. M. University, Mullana, Ambala, Haryana 133203, India**

Abstract

Load Balancing (LB) has been an increasingly important issue for handling computational intensive task in a grid system. By developing strategies that can schedule such tasks to resources in a way that balance out the load, the total processing time will be reduced with improved resource utilization. In this paper, an Enhanced Genetic Algorithm (EGA) is proposed for achieving task scheduling with load balancing. The simulation results show that proposed algorithm yields better performance when compared with other traditional heuristic approaches.

Keywords: *Load Balancing, Task Scheduling, Genetic Algorithm, Grid.*

1. Introduction

Grid computing environment [15] has become a cost effective and popular choice to achieve high performance and to solve large scale computation problems. Grid computing involves coupled and coordinated use of geographically distributed resources for purposes such as large scale computation and distributed data analysis. Task scheduling [4] and load balancing [16] are key grid services, where issues of load balancing represent a common concern for most grid infrastructure developers. In fact, it would be inaccurate to say that the computing power of any system increases proportionally with the number of resources involved. Care should be taken so that resources do not become overloaded and some other stays idle. In general, load balancing algorithms can be roughly classified as centralized or decentralized in terms of location where the load balancing decisions are made.

A load balancing scheme usually consists of three phases: information collection, decision making and data migration. During the information collection phase, load balancer gathers the information of workload distribution, state of computing environment and detects whether there is load imbalance. The decision making phase focuses on calculating an optimal data distribution, while the data migration phase transfers the excess amount of workload from overloaded resource to under loaded ones. In the past decades, a lot of research has focused on the development of effective load balancing algorithms for grid computing environment [1]. To make effective use of tremendous capabilities of the computational resources distributed within the grid environments and maximize the resource utilization, efficient task scheduling algorithms are required [11] [13]. Task scheduling algorithms are commonly applied by the grid manager to optimally dispatch the task to the grid resources [9] [14].

Decision about the assigning of tasks to the resources and finding the best match between the tasks and resources is NP-complete problem [2] [3]. This paper proposes a new task scheduling algorithm to maximize the utilization of grid resources. The algorithm uses genetic heuristic and searches the possible couples of the tasks and resources to find the best matching between them.

The rest of the paper is organized as follows: Section 2 presents related work and our motivation. Section 3 presents the system model. Section 4 describes in detail the design of the proposed algorithm. In Section 5, the performance of proposed algorithm is compared with other traditional heuristic approaches in a series of simulations. Finally, this paper is concluded in Section 6.

2. Related Work

A lot of research had already been done in the field of distributed environment related to load balancing. Due to some specific parameters of grid environments such as relatively high communication costs between resources, most of previously given scheduling and load balancing algorithms are not applicable to these systems [7] [8]. Therefore, there have been ongoing attempts to propose new scheduling algorithms, especially within heterogeneous distributed systems and grid environments [12] [13]. Some of these works are discussed below briefly.

[6] presented Min_min algorithm in which minimum completion time of each task with respect to all resources is computed. Then the task having overall minimum

completion time is selected and assigned to the corresponding resource. The mapped task is removed and process is repeated until the remaining tasks are mapped. [6] also presented Max_min algorithm in which minimum completion time of each task with respect to all resources is computed. Then the task having overall maximum completion time is selected and assigned to the corresponding resource. The mapped task is removed and process is repeated until the remaining tasks are mapped.

In [7], authors have presented an algorithm (QoS guided Min_min) which schedules tasks requiring high bandwidth before the others. Therefore, if the bandwidth required by different tasks varies highly, it provides better results than the conventional Min_min algorithm. Whenever the bandwidth requirement of all the tasks is almost same, the QoS guided Min_min algorithm act similar to the Min_min algorithm. [8] proposed a new algorithm called RASA. RASA uses the advantages of both Min_min, Max_min algorithm. To achieve this, RASA firstly estimate the completion time of the tasks on each of the available resources, and then applies the Max_min and Min_min algorithms alternatively. Experimental results show that RASA is better in comparison with both Min_min and Max_min algorithms within grid environments.

Wang et al. [10] have presented a genetic-algorithm based approach to dispatch and schedule subtasks within grid environments. Subtasks are produced from decomposition of tasks in grid manager and they should be scheduled appropriately. The genetic algorithm based approach separates the matching and scheduling representations and provides independence between the chromosome structure and the details of the communication subsystems. Furthermore, the algorithm considers the overlap existing among all computations and communications that obey subtask precedence constraints. The simulation task presented in [10] for small-sized problems shows that the genetic algorithm based approach can found the optimal solution for these types of problems. [5] [14] also presented genetic based approach to find the optimal schedule. In [16], authors proposed a solution based upon CPU queue length as the load optimization criteria.

3. System Model

We have proposed a model in which grid sites are clustered into regional grids around a set of meta-schedulers in terms of network transfer delay and meta-schedulers are organized in a fully decentralized fashion as shown in Fig. 1.

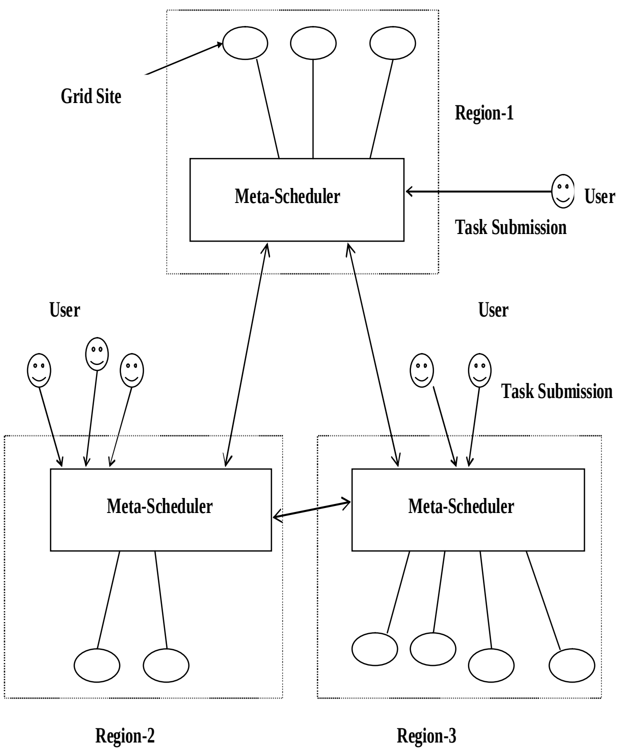

Fig. 1 Decentralized Grid Model.

The user will submit their tasks to the meta-scheduler which select feasible resources from its region for these tasks and finally generate task-to-resource mapping using Enhanced Genetic Algorithm.

4. Proposed Algorithm

The main objective of proposed algorithm is to achieve maximum resource utilization and a well-balanced load among all resources. To achieve this objective, it will consider Makespan value which represents the latest completion time when all tasks involved are considered together instead of looking for an earliest completion time for each task individually.

The EGA is designed based on the standard GAs. The method requires an encoding scheme which can represent all legal solutions to the optimization problem. Any particular solution is uniquely represented by a particular chromosome (or schedule). Chromosomes are manipulated in various ways by applying two genetic operators until the termination condition is met. In order for this manipulation to proceed in the right direction, a quality function called fitness function, is required. In this section we present an in-depth discussion on EGA by enumerating several major points involved. The notations used in the description of EGA are illustrated in Table 1.

Table 1: Notations Used

Notation	*Meaning*
m	No. of Tasks
n	No. of Resources
S	No. of schedules
I	No. of iterations
etc(i, j)	Estimated time of completion of Task i on Resource j
M_Span(i)	Make span of ith schedule
B_Time(i, j)	Busy time of ith resource during jth schedule
R(i,j,k,t)	ith Resource during jth schedule will run kth task for time t units
Ut(i, j)	Utilization value of ith resource during jth schedule
A_Ut(i)	Average resource utilization value of ith schedule
Load(i, S)	Determine the number of tasks allocated to resource i in schedule S
A_Task(i, j)	Allocate a new task as ith entry in resource j
T_Task(i, j)	Transfer task from resource i to resource j
Entry(i, j)	ith entry of jth schedule
Size(i)	Size of ith schedule

4.1 Encoding Mechanism (Generation of Population)

The population is generated consisting of S schedules in which 1st and 2nd schedules are generated using optimal strategies and remaining schedules on random basis. The generation of any schedule deploys a coding scheme satisfying following properties:

i) Size(i) = m for $1 \le i \le S$

ii) Entry (i, k) = $<T_i, R_j, etc\ (T_i, R_j)>$ for $1 \le i \le m,\ 1 \le k \le S,\ 1 \le j \le n$

Let these schedules are denoted as $S_1, S_2, \ldots, S_S$.

4.2 Fitness Function

The main objective is to get task assignments that will achieve well balanced load among all resources. The fitness function will measure the performance of schedules in relation to above said objective. To achieve maximum load balance, we first introduce the concept of average resource utilization. The average resource utilization is defined as the sum of all resources utilization divided by total number of resources. So, expected utilization of each resource based upon task assignment is calculated. This can be achieved by dividing the completion time of last task at each resource by the makespan. For each schedule S_j, calculate the busy time of all resources, makespan, utilization value of all resources and average utilization value as:

$$B\_Time\ (i, j) = \max_t\{R\ (i, j, k, t)\} \quad \text{for } 1 \le i \le n,\ 1 \le j \le S,\ 1 \le k \le m$$

$$M\_Span\ (j) = \max\ \{B\_Time\ (i, j)\}$$

$$Ut\ (i, j) = B\_Time\ (i, j)/M\_Span\ (j)$$

$$A\_Ut\ (j) = \sum_{i=1}^{n} Ut\ (i, j)/n$$

Now arrange the schedules according to decreasing value of fitness function (A_Ut) to obtain new population $S_{11}, S_{21}, \ldots, S_{S1}$ i.e. A_Ut (S_{11}) ≥A_Ut (S_{21}) ≥A_Ut (S_{31}) ≥ ……..≥A_Ut (S_{S1}).

4.3 Genetic Operators

Specialized crossover and mutation operators are developed for use with three-tuple coding scheme. The working of these operators is described below.

Crossover Operator: A single cross over operator is applied on existing population using following steps:

A) Generate a new population consisting of S schedules out of which second half schedules $OS_{S/2+1}$, $OS_{S/2+2}$, …, OS_S are created as given below.

$$OS_k = S_{(k-S/2)1} \quad \text{for } S/2+1 \le k \le S$$

B) Generate remaining schedules in the following manner:

a) Select randomly any two resources R_i and R_j which will act as base of crossover.

b) Apply following computations using the base values:

i) Start with the first schedule S_{11} containing entries of the form $<T_k, R_j, etc(T_k, R_j)>$, on interchanging resources R_i, R_j in all entries of schedule we get:

Tuple before crossover: $< T_1, R_i, etc\ (T_1, R_i)>$, $< T_2, R_j, etc\ (T_2, R_j)>$

Tuple after crossover: $< T_1, R_j, etc\ (T_1, R_j)>$, $< T_2, R_i, etc\ (T_2, R_i)>$

ii) The above step (i) is repeated for schedules S_{21},…., $S_{S/21}$ to obtain next S/2 schedules represented as COS_1 ,COS_2 ,…,$COS_{S/2}$ i.e.

Schedules before crossover: $S_{11}, S_{21}, \ldots, S_{S/21}$

Schedules after crossover: $COS_1, \ldots, COS_{S/2}$

Mutation Operator: This operator is modified to balance load among various resources in term of ensuring that busy time of each resource in a given schedule approaches to make span of schedule i.e.

$$B\_Time\ (i, j) \approx M\_Span\ (j) \quad i \in (1, 2, \ldots, n) \text{ and } j \in (COS_1, COS_2, \ldots, COS_{S/2})$$

Consider the population consisting of S/2 schedules COS_1, COS_2 …$COS_{S/2}$ generated after implementation of crossover operator.

i) Start with the first schedule COS_1 having entries of the form $<T_k, R_j, etc(T_k, R_j)>$ and perform following steps:

a) Calculate load of each resource and allocated tasks in a schedule in following manner:

Load (j, COS_1) = Count (<T_k, R_j , etc(T_k, R_j)> where $1 \le j \le n$, <T_k, R_j , etc(T_k, R_j)> ϵ COS_1

And Count (*) is a function which counts the tuples having second entry as R_j

A_Task (i, j) = k iff <T_k, R_j , etc(T_k, R_j)> ϵ COS_1 for $1 \le i \le$ Load (j, COS_1)

b) Find resources R_{min}, R_{max} having minimum and maximum load.

$$R_{min} = \min_{1 \le j \le n} \{Load\ (j, COS_1)\}$$

$$R_{max} = \max_{1 \le j \le n} \{Load\ (j, COS_1)\}$$

c) Transfer a task from R_{max} to R_{min} so that a portion of load gets balanced.

T_Task (R_{max} , R_{min}) = A_Task (Load (R_{max}, COS_1), R_{max})

d) New load values and task assignment at imbalance resources are:

At R_{min}: Load (R_{min}, COS_1) = Load (R_{min}, COS_1) + 1

A_Task (Load (R_{min}, COS_1), R_{min}) = A_Task (Load (R_{max}, COS_1), R_{max})

At R_{max}: Load (R_{max}, COS_1) = Load (R_{max}, COS_1) – 1

e) Repeat steps b) to d) until the load is balanced.

ii) The above step (i) is repeated for schedules COS_2,…,$COS_{S/2}$ to obtain new S/2 schedules.

Schedules before mutation: COS_1 , COS_2 …$COS_{S/2}$

Schedules after mutation: OS_1, OS_2 …$OS_{S/2}$

The schedules generated after implementation of mutation operator are combined with $OS_{S/2+1}$, $OS_{S/2+2}$ …OS_S to produce a new population of S schedule OS_1, OS_2 ,…,OS_S. Find fitness value of each schedule OS_j and then arrange these schedules of current population according to decreasing value of fitness function. After applying crossover and mutation operators I number of times, the final population comprising of S schedules is generated.

5. Experimental Results

In this section, we present some experiments that have been carried out to test the efficiency and effectiveness of proposed algorithm. The functional code is implemented using simulator built in C language on an Intel core 2 duo, 2 GHz window based laptop. The performance of EGA is tested on two datasets which differ from one another on the basis of expected completion time of tasks i.e. DS_1 (ETC varies from 100 to 200 units) and DS_2 (ETC varies from 100 to 500 units). The following assumptions are devised for simulation model:

i) Tasks are mutually independent i.e. there is no precedence constraint between tasks.

ii) Tasks are computationally intensive and communications overhead are negligible.

iii) Each resource has different computational capability i.e. heterogeneous environment.

In order to determine whether EGA can search a near optimal schedule for a large number of tasks or resources, the simulation was performed in three scenarios.

5.1 Scenario 1 (Effect of load in terms of tasks on average resource utilization)

The number of tasks is varied from 25 to 300 while keeping other simulation parameters as: n=30, S=40, I=500. The average results of execution of the algorithms on different data sets are demonstrated in Table 2, Fig. 2 and Fig. 3.

Table 2: Average Resource utilization on different datasets under scenario 1

No. of Tasks	*DS 1*			*DS 2*		
	EGA	***Max_ min***	***Min_ min***	***EGA***	***Max_ min***	***Min_ min***
25	.8091	.7818	.659	0.7785	0.6786	0.62
45	.9143	.7712	.7238	0.7927	0.6885	0.5656
60	.9655	.808	.8528	0.8906	0.7992	0.6979
90	.9823	.8333	.8771	0.9477	0.8861	0.7363
100	.9674	.8919	.8024	0.9272	0.887	0.7862
200	.9670	.9366	.8973	0.9383	0.9173	0.839
300	.9718	.9471	.9423	0.9634	0.9502	0.8934

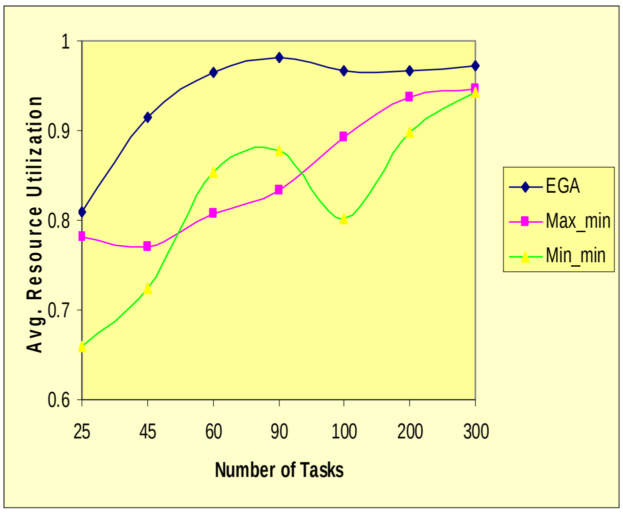

Fig. 2 Effect of load variation on DS1.

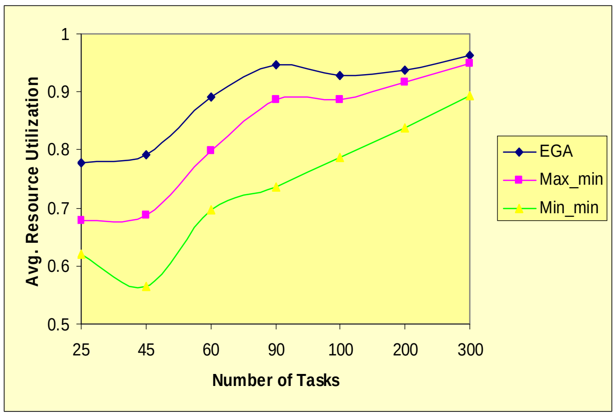

Fig. 3 Effect of load variation on DS 2.

5.2 Scenario 2 (Effect of scalability on average resource utilization)

The number of resources is varied from 10 to 40 while keeping other simulation parameters as: m=200, S=40, I=500. The average results of execution of the algorithms on different data sets are demonstrated in Table 3, Fig. 4 and Fig. 5.

Table 3: Average Resource utilization on different datasets under scenario 2

No. of Resources	*DS 1*			*DS 2*		
	EGA	*Max_ min*	*Min_ min*	*EGA*	*Max_ min*	*Min_ min*
10	.9885	.9767	.971	.99	.9855	.9423
15	.9844	.965	.938	.98	.9725	.9413
20	.9864	.9517	.9533	.961	.9546	.8811
25	.9858	.9557	.9255	.9719	.9491	.8715
30	.9670	.9366	.8973	.9383	.9173	.8391
35	.9734	.9062	.9277	.9368	.911	.8575
40	.9780	.9004	.8733	.9302	.9184	.843

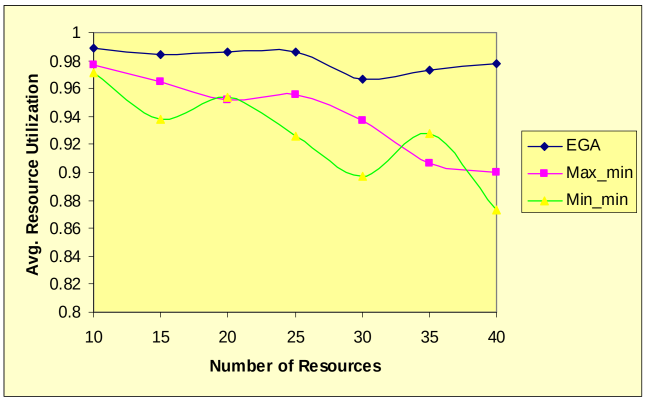

Fig.4 Effect of scalability on DS 1.

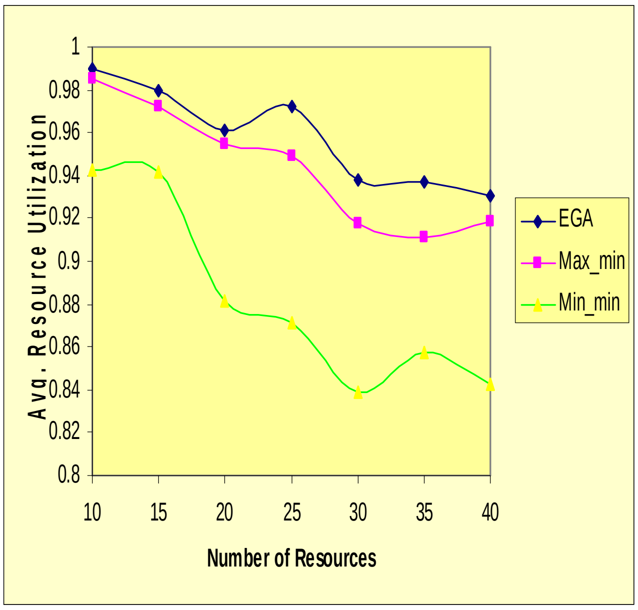

Fig. 5 Effect of scalability on DS 2.

5.3 Scenario 3 (Effect of load and scalability on average resource utilization)

The number of tasks is varied from 15 to 200 and resources are varied from 5 to 45 while keeping other simulation parameters as: S=40, I=500. The average results of execution of the algorithms on four different data sets are demonstrated in Table 4, Fig. 6 and Fig. 7.

Table 4: Average Resource utilization on different datasets under scenario 3

No. of Resources	*No. of Tasks*	*DS 1*			*DS 2*		
		EGA	*Max_ min*	*Min_ min*	*EGA*	*Max_ min*	*Min_ min*
5	15	0.9867	0.8917	0.8683	0.9823	0.8549	0.7522
5	25	0.97	0.9443	0.8738	0.9588	0.8932	0.8611
10	30	0.987	0.9067	0.9143	0.9375	0.8431	0.7528
10	50	0.9857	0.9169	0.9276	0.9667	0.9303	0.8183
15	45	0.9754	0.827	0.8133	0.9439	0.8486	0.7801
15	75	0.9738	0.9035	0.9138	0.9504	0.8873	0.8053
20	60	0.9756	0.8466	0.8722	0.9432	0.8642	0.7532
20	100	0.9895	0.919	0.8746	0.95	0.9089	0.8486
25	150	0.9814	0.928	0.9261	0.9557	0.9363	0.8874
45	200	0.9581	0.9115	0.8732	0.9133	0.8863	0.8324

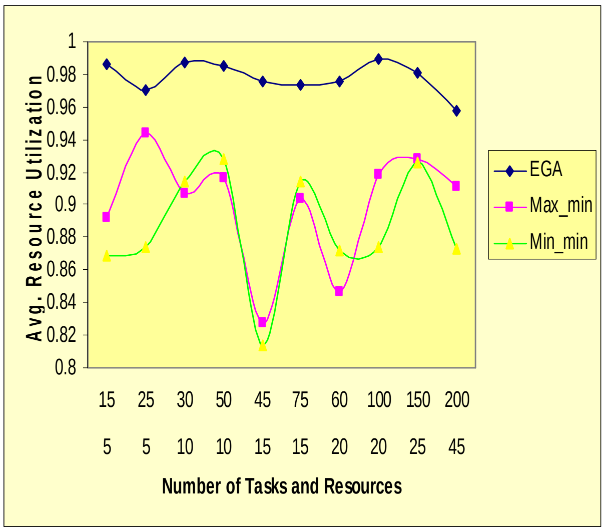

Fig. 6 Effect of load and scalability on DS 1.

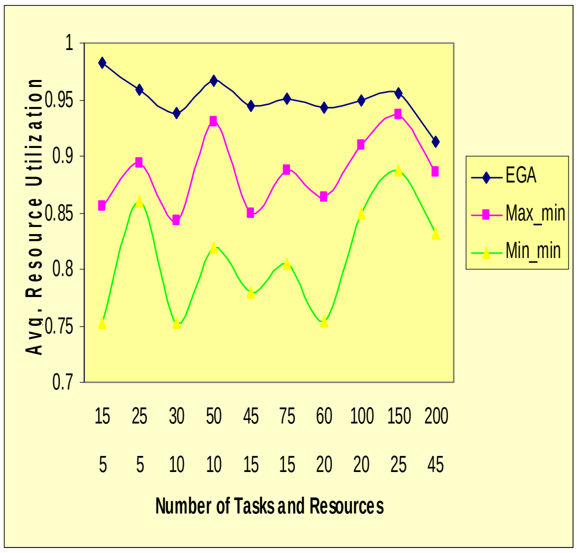

Fig. 7 Effect of load and scalability on DS 2.

6. Conclusions

Load balancing leads to achieve minimum waiting time, improves the response time and resource utilization rate. The problem of load balancing in grid environment is directly related to allocation of tasks among computational resources available in the system. In this paper we have proposed a genetic based algorithm for load balancing across resources for computational intensive tasks on grid environments. From the simulation results, it is concluded that the proposed algorithm has been effective under various load conditions and in terms of scalability.

References

[1] H. Shan, L. Oliker, and R. Biswas, "Job Super scheduler Architecture and Performance in Computational Grid Environments", Proceeding of ACM/IEEE Conference on Supercomputing, 2003, pp. 44-48.

[2] Yu-kwong, and Lap-sun., "A new fuzzy-decision based load balancing system for distributed object computing", Journal of Parallel and Distributed Computing, Vol. 64, No. 2, 2004, pp. 238-253.

[3] Xio Qin, and Hong Jeong, "Improving Effective Bandwidth of Networks on Clusters using Balancing for Communication-Intensive", Proceeding of 24th IEEE Intern Computing and Communications Conference (IPCCC 2005), 2005, pp.27-34.

[4] A.Y.Zomaya, and Y.H.The, "Observations on using genetic algorithms for dynamic load-balancing", IEEE Transactions on Parallel and Distributed Systems, Vol.12, No. 9, 2001, pp. 899-912.

[5] Reza Entezari-Maleki, and Alo Movaghar, "A Genetic Algorithm to Increase the Throughput of the Computational Grids", International Journal of Grid and Distributed Computing, Vol. 4, No. 2, 2011, pp.11-24.

[6] A. Armstrong, D.Hensgen, and T. Kidd, "The relative performance of various mapping algorithms in independent of sizable variances in runtime predictions", Proceeding of 7th IEEE Heterogeneous Computing workshop (HCW'98), 1998, pp. 79-87.

[7] X.He, X.H. Sun, and G.V. Laszewski, "QoS Guided Min_min Heuristic for Grid Task Scheduling", Journal of Computer Science and Technology, Vol. 18, 2003, pp. 442-451.

[8] S. Parsa, and R. Entezari-Maleki, "RASA: A New Grid Task Scheduling Algorithm", International Journal of Digital Content Technology and its Applications, Vol. 3, 2009, pp. 91-99.

[9] M. Maheswaran, S. Ali, and H.J. Siegel, and D. Hensgen, and R.F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems", Journal of Parallel and Distributed Computing, Vol. 59, 1999, pp. 107-131.

[10] L. Wang, H.J. Siegel, V.P. Roychowdhury, and A.A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Based Approach", Journal of Parallel and Distributed Computing, Vol. 47, 1997, pp. 1-15.

[11] E.U. Munir, J. Li, and S. Shi, "QoS Sufferage Heuristic for Independent Task Scheduling in Grid", Information Technology Journal, Vol. 6, 2007, pp. 1166-1170.

[12] L. Mohammad Khanli, and M. Analoui, "Resource Scheduling in Desktop Grid by Grid-JQA", Proceeding of 3rd International Conference on Grid and Pervasive Computing, 2008, pp.63-68.

[13] L. Mohammad Khanli, and M. Analoui, "Grid_JQA: A QoS Guided Scheduling Algorithm for Grid Computing", Proceeding of 6th International Symposium on Parallel and Distributed Computing, 2007, pp. 242-249.

[14] Jingyi Ma, "A Novel Heuristic Genetic Load Balancing Algorithm in Grid Computing", Proceeding of 2nd

International Conference on Intelligent Human-Machine Systems and Cybernetics, 2010, pp. 166-169.

[15] Manpreet Singh, and P.K.Suri, "An Efficient Decentralized Load Balancing Algorithm for Grid", Proceeding of 2nd IEEE International Conference on Advanced Computing, 2010, pp. 10-13.

[16] Manpreet Singh, Sandip Kumar Goyal, and Vishal Gupta, "An Adaptive Load Balancing Algorithm for Computational Grid", Journal of Engineering and Technology, Vol. 1, No. 2, 2011, pp. 70-73.

Sandip Kumar Goyal received his B.Tech., M.Tech. from Kurukshetra University, Kurukshetra, India and is currently enrolled as a Ph.D. scholar in the department of Computer Science and Engineering at M.M.University, Haryana, India. He is presently serving as Assoc. Professor in Computer Engineering Department of M.M. Engineering College, Mullana, Ambala. He is in teaching since 2000. He has published 8 research papers in International and National journals and conferences. He has supervised several M.Tech. Dissertations. His research area is Load balancing Methodologies in distributed environment.

Dr. Manpreet Singh received his B. Tech., M.Tech. and Ph.D. from Kurukshetra University, Kurukshetra, India. He is presently serving as Professor and Head, Computer Science and Engineering Department of M. M. Engineering College, Mullana, Ambala. He has about 13 years of experience in teaching and research. He has published 30 research papers in International and National journals and conferences. His current research interest includes Grid Computing, Cloud Computing, Distributed Databases, and MANETs.