Genetic Algorithm for Network–Aware Job Scheduling in Grid Environment

Saumesh Kumar[1], Naveen Kumar[2], Padam Kumar[3]

Department of Electronics and Computer Engineering

Indian Institute of Technology Roorkee

Roorkee, India

[1]saumeshkumar@gmail.com

[2]navincse2005@gmail.com

[3]padamfec@iitr.ernet.in

***Abstract*— In this paper we present the implementation of Genetic Algorithm based grid scheduler that minimizes the jobs finalization time. For data and compute intensive jobs, storage and computing resources need to communicate with each other over the networks, so the network should be used in an efficient way. The proposed scheduling algorithm not only takes processing power of resources into account but also network characteristics when making scheduling decisions. We have implemented and tested the proposed scheduling algorithm in GridSim. Results show that the jobs finalization time and makespan are reduced when network characteristics are also considered in scheduling decisions.**

***Keywords*— Job scheduling, makespan, genetic algorithms, GridSim, network based scheduling, grid systems.**

I. INTRODUCTION

Grid systems are highly distributed and dynamic in nature because of high degree of heterogeneity of jobs and resources. Grids are made of geographically distributed and independent virtual organizations that share their resources. Computing resources are under different administrative domain having their own access policy, local resource management system and other constraints. Grid technology has emerged as an enabling technology for many data and/or computing intensive applications. Through the use of Grid technologies it is possible to aggregate geographically dispersed heterogeneous resources for solving large-scale parallel applications in science, engineering and commerce [1].

Grid resource management provides the functionality for discovery and publishing resources as well as scheduling, submission and monitoring of jobs. The development or adaption of application for grid environment is being challenged by the need of scheduling a large number of jobs to resources efficiently. A scheduler is a component of the resource management system on a grid. The scheduler is responsible to allocate the set of tasks/ jobs to available compute resources in accordance with the best available schedule. A job scheduler, as defined by M. Aggarwal et at. [6], is designed to satisfy one or more of the following objects: (a) minimizing turn-around time for an application; (b) maximizing system throughput; (c) maximizing resources utilization.

Job scheduling problems are NP –complete [5], so use of heuristics is one of the known approaches to cope up with its difficulty. There are many classes of heuristics like Simulated Annealing, Tabu Search, Ant Colony and Particle Swarm Optimization to solve job scheduling problem in grid environment [2]. Classical algorithms are not dynamic, and hence they cannot be used in grids because of its dynamic nature, to achieve the optimal schedule. For example, in a given time slot, the Grid may need to schedule a single task, multiple independent tasks, or a collaborative task on varying number of heterogeneous and geographically dispersed resources.

In many scientific domains particularly High Energy Physics (HEP), experiments produce huge amount of the data that need to be processed and analyzed [7]. Grid technology is best suited to develop application for these kinds of experiments which requires handling of large amount of data. A well known example of such application is grid-based worldwide data-processing infrastructure deployed for the LHC (Large Hadron Collider) experiment at CERN. Recently, grid applications are becoming increasingly network dependent. These applications like providing analysis data for remote visualization need to access huge amounts of remote input data or exchange data between distant machines [8]. Most of the current schedulers (like Nimgrod/G [11], Condor/G [9]) consider computing power and utilization of resources, budget and other factors, in their scheduling metrics. Most of the existing scheduler does not take into account network characteristics when making scheduling decisions. According to L. Thomas et al. [8] and A. Anjum et al. [10], schedulers that do not consider network characteristics when making scheduling decisions might produce the schedule that is not optimal. For example, scheduler might decide that the most powerful or unloaded resources are best suited to run user's job. However, if there is some overloaded link in the path between data sources and computing resource, a better choice would be to run the job on other less powerful resource with less loaded network. Hence, network characteristics must also be considered when making scheduling decisions.

In this paper, we have implemented the multi-objective genetic algorithm for the jobs scheduling in grid computing. GridSim [17] has been used to simulate the grid environment. We have implemented the proposed genetic algorithm in Java. There are several objectives to be considered for job scheduling, but we have mainly focused on minimizing the jobs' finalization time and makespan by minimizing the jobs' required data transfer time between data storage location and computing site over the network.

II. RELATED WORK

In recent years a number of grid schedulers have been developed with their own scheduling strategies. Most of the existing schedulers can be embedded within resource management component like GRAM (Grid Resource Allocation Manager, a component of Globus Toolkit [12]) or can be used as an external scheduler to grid middleware. The Globus toolkit [12] has emerged as the *de facto* standard for open grid computing infrastructure. Globus allows Condor/G [9] and Nimgrod/G [11] schedulers to be used as external scheduler.

Nimgrod/G [11] was designed by modifying Nimgrod for operation with Globus toolkit. Nimgrod follows deadline, economy and budget based scheduling for grid systems. However, this scheduler does not provide functionality to access remote data repositories. Nimgrod/G is a part of Grid Architecture for Computational Economy (GRACE) framework. The broker of Nimgrod/G works with other components like bid-manager and directory server of GRACE to maximize the performance goals.

Condor/G [9] can run both sequential and parallel jobs and provides jobs checkpointing. It supports MPI and PVM for massive parallel jobs. One of the main features of this scheduler is that it includes DAGMan which provides a mechanism to describe job dependencies. Condor/G is one of the job scheduler supported by GRAM (Grid Resource and Allocation Manager), a component of Globus Toolkit [12].

A. Anjum et al. [10] have presented the architecture for Data Location and Network Aware (DIANA) job scheduling in grid system. They have also proposed network and computation cost estimators. L. Thomas et al. ([8], [16]) have improved the GridWay metascheduler with network aware scheduling. They have also compared the performance of the algorithm under different scheduling policies. I. M. Llorente et al. [7] have proposed the scheduling for data–intensive jobs. The authors had implemented several functions to reduce the data replication cost at storage resources.

There are several genetic algorithm (GA) based schedulers for computational grid. However, in most of the GA based schedulers, scheduling decisions are focused on maximizing resource utilization, minimizing makespan and flowtime or load balancing. M. Aggarwal et al. [6] have proposed GA based scheduler that uses a Directed Acyclic Graph (DAG) to represent subtask of a job and arbitrary precedence constraints among them. The authors have proposed a scheduler that minimizes makespan and idle time of the available computational resources.

Y. Gao et al. [13] have proposed two models for predicting the completion time of jobs in a service grid. Authors have developed two algorithms that use the prediction models to schedule jobs at both system level and application level. The genetic algorithm in application level scheduling has been used to minimize the average completion time of jobs through optimal job allocation on each resource.

Authors of [14] have proposed a mathematical model for genetic algorithm by using "pattern theory" and "building block hypothesis" technologies. The theoretical principle was to extract some "character pattern" from the excellent individual (chromosome).

N. J. Vavimipour et al. [15] have proposed Linear Genetic Representation (LGR) method to represent chromosome for genetic algorithms. Modular arithmetic (congruent modulo n) has been used to represent individuals. Authors have also used DAG to represent precedence among jobs and execution times.

Carretero, Xhafa and Abraham [3] have proposed a multi-objective genetic algorithm for scheduling that minimizes the makespan and flowtime. Chromosomes have been represented by two different encodings: direct and indirect encoding and each encoding have different crossover operators. Authors have used a model, Expected Time to Compute (ETC), for estimating the execution time of jobs on resources. They have also compared the performance of different genetic operators (crossover and mutation) for different number of evolution steps.

III. NETWORK-AWARE JOB SCHEDULING

The proposed scheduling problem can be viewed as mapping of *n* independent jobs *J = {J1, J2, ..., Jn}* to a set of *m* resources *R = {R1, R2, ..., Rm}* with objectives of minimizing the jobs' finalization time and makespan, that is, the time when latest job finishes. We have used *Expected Time to Compute (ETC),* defined in [18] and [3], to estimate the computation time of a job on a particular resource. ETC is two dimensional matrix of size 'n x m', where *n* and *m* are the number of jobs to be scheduled and number of resources respectively. Each position *ETC[j][r]* indicates the expected time to compute job *'j'* on resource *'r'*. ETC model requires that we know the computing capacity of resources and workload of each job. The computing power (in Million Instructions Per Second, MIPS) of resources can be obtained by Grid Information Service (GIS) or MDS (Monitoring and Discovery System, in Globus), and workload (in Million Instructions) of jobs can be obtained from job submission file. In Globus [12], JSDL (Job Submission Description Language) is used for creating job submission template file.

To capture the network characteristics and data requirements of jobs, we have used four arrays: *data_size[n], data_source[n], prop_delay[m][m]* and *BW[m][m]* in our scheduling algorithm. The entries of these four arrays indicate followings:

data_size[i] = size (in bytes) of data required for the execution of the job J. (1 <= i <= n).

data_source[i] = Rx; resource at which data for job Ji is stored. (1 <= i <= n)

prop_delay[x][y] = propagation delay (in seconds) from resource Rx to Ry.

BW[x][y] = bottleneck (minimum) bandwidth (in bits per second, bps) from resources Rx to Ry. (1 <= {x, y} <= m)

We have assumed that the required data of a job is stored at only single resource. In real grid environment, the arrays *prop_delay* and *BW* can be obtained by open source utilities like Iperf [19] and BwPing [20]. The matrix *prop_delay[m][m]* is not symmetric because the propagation delay in two direction (i.e. from resource *Rx to Ry* and vice-versa) might not be the same. Similarly, the matrix *BW[m][m]* is also not symmetric because bandwidth in two direction might be different.

The entries in *ETC[i][x]* are computed as follows:

*ETC[i][x] = MI of job Ji / MIPS of resource Rx + prop_delay[data_source[i]][x] + (data_size[i] * 8) / BW[data_source[i]][x]*

The makespan of resources, as explained by J. Carretero et al. [3], is computed with the help of three vectors: *completion_time[m], ready_time[m]* and *schedule[n]*. The entry in *schedule[i]* indicates the resource at which job *Ji* is scheduled by the scheduler. The *ready_time[x]* indicates the time at which resource *Rx* will finalize the processing of the previous jobs. The entries of *completion_time* vector indicate the time in which resources will finish the execution of all the jobs (previously assigned jobs and already planned for that resource). Thus, *ready_time[x]* and *completion_time[x]* are computed as follows:

ready_time[x] = (Total MIs of all jobs in resource Rx's queue) / MIPS of resource Rx.

$$completion\_time[x] = ready\_time[x] + \sum_{\{i \in Jobs \mid schedule[i] = x\}} ETC[i][x]$$

Then, makespan can be expressed in terms of *completion_time* vector as:

$$makespan = \max \{completion\_time[i] \mid i \in R\}$$

Our aim is to minimize the jobs' finalization time which can be achieved by minimizing the data transfer time. But minimizing only data transfer time might not be enough because the resource to which job is scheduled might be heavily loaded (i.e. very large value in *ready_time[x]* for resource *Rx*). So we also need to minimize the makespan to minimize the jobs' finalization time.

IV. GENETIC ALGORITHM

Genetic algorithm (GA) is an efficient searching mechanism that has great application for solving optimization problems. According to R. Buyya et al. [4], job scheduling problems are optimization problems, hence GAs (population based heuristics) are well suited to solve the job scheduling problems in grid systems. In GAs, the evolution usually starts from a population of randomly selected individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified to form a new population by means of genetic operators (crossover and mutation) to form new individuals (solutions). The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached. So GA does not guarantee the best possible solutions, but normally, provides the optimum solution.

In multi-objective genetic algorithms, there are two approaches of optimization: hierarchical and simultaneous [3]. In hierarchical approach the optimization criteria are sorted by their importance, in a way that if a criteria Ci is of smaller importance than criteria Cj, then the value of criteria Cj cannot be varied while optimizing according to Ci. However, in simultaneous approach, both criteria are optimized simultaneously.

V. IMPLEMENTATION

The proposed genetic algorithm based scheduler has been designed to schedule multiple heterogeneous jobs, where each job requires some input data for its execution. We have implemented the proposed genetic algorithm based scheduling in Java using the generic genetic algorithm template specified in [3]. GridSim [17] has been used for the simulation of grid environment. Following components of the genetic algorithm have been implemented for the proposed scheduling algorithm:

- **Schedule Encoding (Representation):** The encoding represents the chromosome (individual) in the real world. We have implemented the *Direct Representation* to encode the feasible solutions. A chromosome is represented by a vector *schedule* of size *n,* where *schedule[i]* indicates the resource where job *Ji* is scheduled.

- **Population:** Population is a set of possible solutions (schedules). It is responsible to hold the solutions from one generation to next generation. Population has been implemented by extending *LinkedList* in Java.
- **Selection operators:** Selection operators are used to select the individuals on which the crossover operators will be applied. *Select Best* policy has been used to implement the selection operator. In this policy, individuals with better fitness have higher probability of being selected for genetic operation (crossover and mutation).
- **Crossover operators:** The aim of crossover operator is to mix up the genetic information coming from different chromosomes to make one or more new chromosomes. We have implemented *Two-point* crossover operator.
- **Mutation operators:** Mutation operator selects one chromosome and then produces new child from it by a slight change over the parent. *Swap* mutation has been implemented for the proposed scheduling algorithm. Swap mutation interchanges the allocation of two different jobs which are allocated to different resources.
- **Replacement Operator:** This operator is responsible for passing the individuals from one generation to the next generation. We have used *survival of fittest* mechanism to implement replacement operator. This mechanism has been implemented in such a way that the unfit chromosomes (whose fitness is very low) of the parent generation are replaced by the fitter chromosomes (whose fitness is good) of the current generation to generate the population for next generation.
- **Fitness Function:** This function is used to evaluate the fitness of chromosomes. This also controls the genetic operators. Hierarchical optimization approach has been used to implemented fitness function. The first criteria for the implemented hierarchical optimization is to minimize the data transfer time for jobs (i.e. minimizing *ETC[i][x]* for job *Ji* on resource *Rx*) and second low priority criteria is to minimize the makespan by choosing lightly loaded resources.

VI. EXPERIMENT AND RESULTS

Experiments to test the proposed scheduling algorithm are carried out under the simulation environment created by GridSim. The network topology that has been used for the experimental setup is shown in Fig. 1.

The configuration of resources that has been used for the simulation of grid environment is given in Table 1. In GridSim, a grid resource is represented as a cluster. So each grid resource has a list of machines, each machine has a list of PEs (Processing Elements) and computing power of PE is defined in terms of MIPS (Millions Instructions Per Second). The computing power (in MIPS) of the grid resource is the sum of MIPS of all the PEs in that resource. Workload of the jobs (gridlets in GridSim terminology) is specified in terms of MI (Million Instructions). *InfoPacket* utility of GridSim package has been used to measure the available bandwidth and propagation delay between the resources.

The proposed scheduling algorithm has been verified for 50 data gridlets (jobs) of workload that follows a normal distribution N (μ = 500, σ = 100). The size (in MB) of input

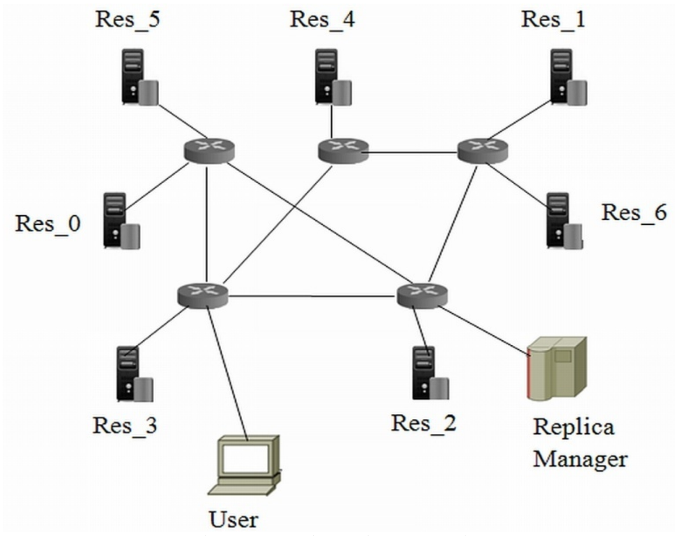

Fig.1: Network topology scenario.

TABLE 1
CONFIGURATION OF THE RESOURCES.

Resource name	**Number of machines in the resource**	**Number of PE in each machine**	**MIPS rating of each PE**	**Storage capacity (in GB)**
Res_0	5	4	10	500
Res_1	3	3	12	200
Res_2	4	3	11	200
Res_3	6	2	12	100
Res_4	1	1	15	5
Res_5	1	1	15	5
Res_6	1	1	15	5

TABLE 2
PARAMETERS FOR GENETIC ALGORITHM.

Population size	120
Crossover Operator	Two-Point
Cross probability	0.75
Selection operator	Select Best
Mutation operator	Swap
Mutation probability	0.2

data required for the execution of the gridlet is uniformly distributed with mean 250 and variance 50.The parameters of the genetic algorithm that we have used for the network-aware scheduling are given in Table 2.

The proposed genetic algorithm for network-aware scheduling has been compared with the non network-aware scheduling for makespan, finalization time and waiting time of jobs. The non network-aware scheduling aware scheduling algorithm considers only computing power of resources, load on resources, amount of memory available, and other scheduling factors; but does not consider network characteristics (network bandwidth and latency). The comparisons of network-aware and non network-aware scheduling algorithm are:

(a) Comparison of the makespan of network-aware scheduling with non network-aware scheduling algorithm for two-point crossover operator against the different number of evolutions steps is shown in Fig. 2. From the figure it is clear that makepsan (in seconds) of the schedule with network-aware scheduling is very less than that of the non network-aware scheduling algorithm.

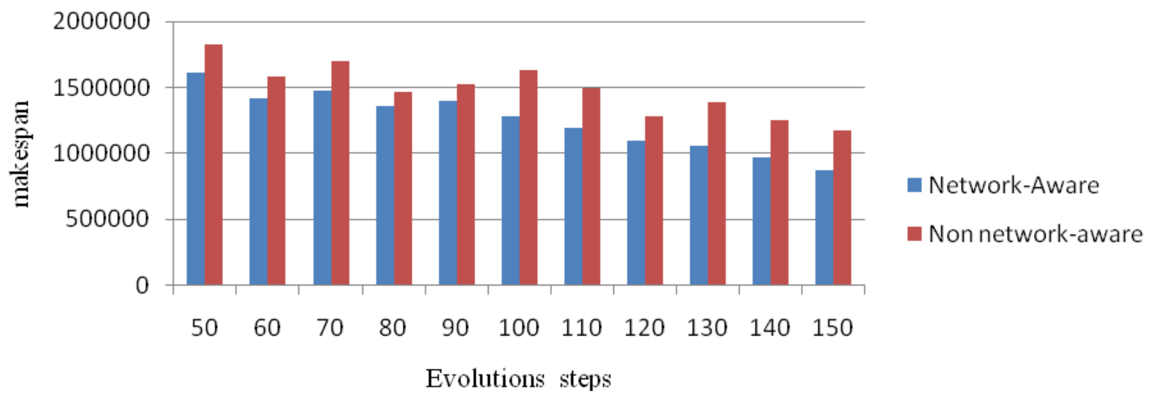

Fig.2: Comparison of makespan.

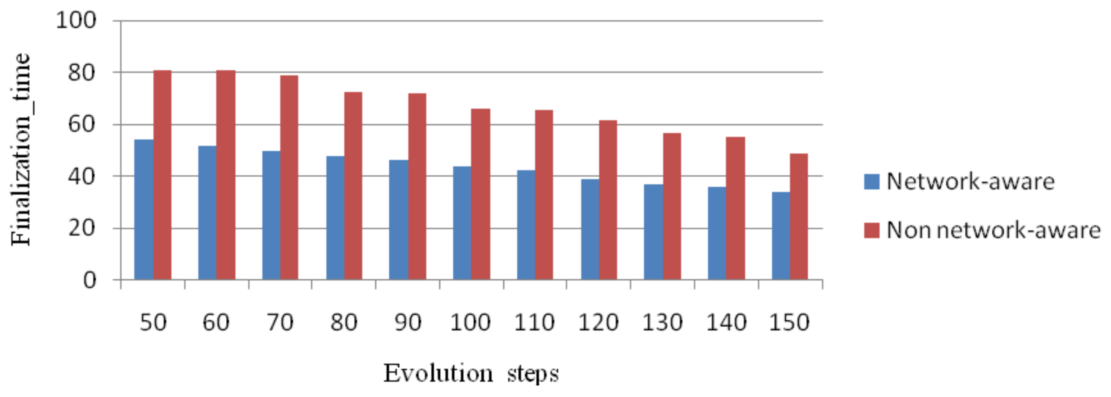

Fig.3: Comparison of finalization time.

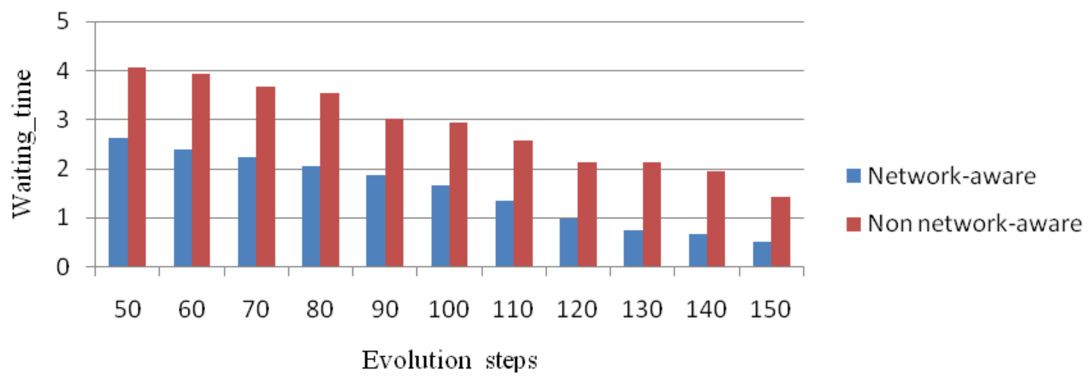

Fig.4: Comparison of waiting time.

(b) Comparison of the finalization time of jobs under network-aware and non network-aware scheduling algorithm is shown in Fig. 3. Finalization time of jobs is computed as the average finalization time of all jobs in the schedule. Two-point crossover genetic operator with swap mutation has been used to compare the finalization time for different number of evolution steps. It can be seen that finalization time of jobs in network-aware scheduling is less than that of the non network-aware scheduling.

(c) Comparison of waiting of jobs for network-aware and non network-aware scheduling is shown in Fig. 4. Form the figure it is clear that jobs' waiting time has been reduced under network-aware scheduling than that of under non network-aware scheduling.

VII. CONCLUSIONS

We have designed and tested the Genetic Algorithm for network-aware job scheduling in grid environment. Network-aware scheduler functionality has been verified under the simulation environment for 50 jobs with varying data requirements. These jobs are data intensive jobs which require data movement over network from data sources to computing resources. The experimental results show that the network-aware job scheduling not only minimizes waiting time of jobs but also reduces the finalization time of jobs and minimizes makespan.

In this paper, we have considered non-preemptive behavior of resources. The proposed scheduling algorithm can be further improved by preempting the resource during the data transfer of the job in execution, and allocating the resource to the other jobs, whose required data has already been transferred to the resource.

REFERENCES

[1] I. Foster and C. Kesselman, *The grid: blueprint for a new computing infrastructure*: Morgan Kaufmann, 2004.

[2] A. Yarkhan and J. Dongarra, "Experiment with scheduling using simulated annealing in a grid environment," in *3rd International Workshop on Grid Computing*, pp. 232 – 242, 2002.

[3] J. Carretero, F. Xhafa, and A. Abraham, "Genetic algorithm based schedulers for grid computing systems," *International Journal of Innovative Computing, Information and Control (ICIC),* vol. 3, pp. 1349-4198, 2007.

[4] A. Garrido, M. A. Salido, F. Barber, and M. A. López, "Heuristic methods for solving job-shop scheduling problems," 2000, pp. 36–43.

[5] A. Abraham, R. Buyya and B. Nath, "Nature's Heuristics for scheduling Jobs on Computational Grids," in Proc. of *8th IEEE International Conference on Advanced Computing and Communication, India*, 2000.

[6] M. Aggarwal, R. D. Kent and A. Ngom, "Genetic Algorithm Based Scheduler for Computational Grids," in Proc. of *19th IEEE International Symposium on High Performance Computing Systems and Applications*, 2005.

[7] A. D. Peris, J. Hernandez, E. Huedo and I. M. Llorente, "Data Location-Aware Job Scheduling in the grid. Application to the GridWay Metascheduler," in Proc. of *17th International Conference on Computing in High Energy and Nuclear Physics, Journal of Physics*, 2010.

[8] L. Thomas, A. Caminero, B. Caminero and C. Carrion, "Improving GridWay with Network Information: Tuning the Monitoring Tool," 2009.

[9] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A computation management agent for multi-institutional grids," *Cluster Computing,* vol. 5, pp. 237-246, 2002.

[10] R. McClatchey, A. Anjum, H. Stockinger, A. Ali, I. Willers, and M. Thomas, "Scheduling in data intensive and network aware (DIANA) grid environments," *Journal of Grid Computing,* 2007.

[11] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid," 2000, p. 283.

[12] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," *Network and Parallel Computing,* pp. 2-13, 2005.

[13] Y. Gao, H. Rong, and J. Z. Huang, "Adaptive grid job scheduling with genetic algorithms," *Future Generation Computer Systems,* vol. 21, pp. 151-161, 2005.

[14] H. Liu, W. Zhang, J. Hao, J. Sun, and Y. Li, "Research on Scheduling Algorithm Based on Genetic Algorithm for Grid Tasks," 2010, pp. 1-4.

[15] L. M. Khanli, S. N. Razavi, and N. J. Navimipour, "LGR: The New Genetic Based Scheduler for Grid Computing Systems," *CIMCA 2008, IAWTIC 2008, and ISE 2008,* pp. 639-644, 2008.

[16] L. Tomás, A. Caminero, B. Caminero, and C. Carrión, "Studying the influence of network-aware grid scheduling on the performance received by users," *On the Move to Meaningful Internet Systems: OTM 2008,* pp. 726-743, 2008.

[17] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and Computation: Practice and Experience,* vol. 14, pp. 1175-1220, 2002.

[18] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," 1999, p. 30.

[19] NLANR/DAST, "Iperf – The TCP/UDP Bandwidth Measurement Tool," Web page at http://iperf.sourceforge.net/.

[20] BwPing, Web page at http://bwping.sourceforge.net/.