A New Heuristic Approach for Scheduling Independent Tasks on Grid Computing Systems

Sadegh Nejatzadeh[1], Mahmood Karamipour[2] and Mohammad Eskandari[1]

[1]*Shahid Beheshti University, Tehran, Iran*
[2]*K. N. Toosi University of technology, Tehran, Iran*

[1]*nejatzadeh@gmail.com*

Abstract

Grid computing is a promising technology for future computing platforms and is expected to provide easier access to remote computational resources that are usually locally limited. Scheduling is one of the active research topics in grid environments. The goal of grid task scheduling is to achieve high system throughput and to allocate various computing resources to applications. The Complexity of scheduling problem increases with the size of the grid and becomes highly difficult to solve effectively. Many different methods have been proposed to solve this problem. Some of these methods are based on heuristic techniques that provide an optimal or near optimal solution for large grids. In this paper, a new heuristic algorithm for scheduling meta-tasks in grid computing system is presented which tries to consider the execution time and machine state simultaneously by a mapping function. According to the experimental results, the proposed algorithm confidently demonstrates its competitiveness with previously proposed algorithms.

Keywords: *Grid Computing, Heuristic, Makespan, ETC Matrix*

1. Introduction

Grid computing and distributed computing, dealing with large scale and complex computing problems, is a hot topic in the computer science and research. According to Foster and Keselman in [1], grid computing is hardware and software infrastructure which offer a cheap, distributable, coordinated, and reliable access to powerful computational capabilities. Miscellaneous resources should be orchestrated to perform a number of tasks in parallel or to solve complex tasks atomized to variety of independent subtasks. To exploit the different capabilities of a suite of grid resources, typically a resource management system (RMS) allocates the resources to the tasks and the tasks are ordered for execution on the resources [2, 19]. Task scheduling is mapping a set of tasks to a set of resources to efficiently exploit the capabilities of them. As mentioned in [3, 20] optimal mapping tasks to machines of a grid computing environment is an NP-complete problem and therefore the use of heuristics is one of the suitable approaches. The heuristics applied to the grid scheduling problem include Min-min, Max-min, Longest job to fastest resource-Shortest job to fastest resource (LJFR-SJFR), Sufferage, Work queue and others [2-7]. Different criteria can be used for evaluating the efficiency of scheduling algorithms, the most important of which are makespan and execution time. Makespan is the time when grid computing system finishes the latest task. An optimal schedule will be the one that minimizes the makespan and execution time. The objectives of scheduling algorithms are increasing the system throughput measure, reducing task completion time, better resource utilization rate and balancing the load well [8, 9].

This paper presents a new scheduling algorithm for static mapping to achieve better performance. The employed heuristic approach uses a simple mapping function which tries to show the execution time and machine state together. The maximum point of this mapping is used to assign works on machines. The proposed was tested using the benchmark model of Braun *et al.*, [3] and the experimental results were satisfactory. However, the results are proposed method is not better than Min-Min method in terms of makespan, but its results are comparable while its running time is three times faster than Min-Min. Therest of paper is organized as follows. Section 2 describes problem definition. The descriptions of the related works are given in Section 3. In Section 4, we describe the proposed algorithm. In Section 5, the proposed method is described. Finally in Section 6, the paper is concluded and several issues for future works are indicated.

2. Problem Definition

A heterogeneous computing environment (*e.g.*, Grid environment) is composed of computing resources where these resources can be a single PC, a cluster of workstations or a supercomputer. The main goal of the task scheduling in grid computing environments is the efficiently allocating tasks to machines. The efficiency means that we are interested to allocate tasks as fast as possible and optimizing the makespan objective. We use the ETC (Expected Time to Compute) matrix model introduced by Shoukat *et al.*, for formulating the problem [15]. It is assumed that an accurate estimate of the expected execution time for each task on each machine is known prior to execution and contained within an ETC matrix. Each row of the ETC matrix contains the estimated execution times for a given task on each machine. Similarly, each column of the ETC matrix consists of the estimated execution times of a given machine for each task. ETC[i,j] is the expected execution time of task *i* in machine *j*. For the simulation studies, characteristics of the ETC matrices were varied in an attempt to represent a range of possible heterogeneous environments. Using the ETC matrix model, the scheduling problem can be defined as follows:

- A number of independent tasks to be allocated to the available resources. Because of Non-preemptive scheduling, each task has to be processed completely in a single machine
- Number of machines is available to participate in the allocation of tasks
- The workload of each task (in millions of instructions)
- The computing capacity of each resources (in MIPS)
- Ready[m] represents the ready time of the machine after completing the previously assigned tasks
- ETC matrix of size $m \times n$, where n represents the number of tasks and m represents the number of machines.

3. Related Works

Many algorithms proposed for static task scheduling in grid environments. Eleven different static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems (*e.g.*, grid environment) are introduced in [3] and the performance of them are compared together. It is assumed that the heuristic derive mapping statically for the collection of independent meta-task. The scheduling problem is computationally hard even

though there are no data dependencies among the jobs. Some famous heuristic approaches are as follows:

3.1. Opportunistic Load Balancing (OLB)

OLB assigns each job in random order to the next available machine without considering the job's expected execution time on the machine [2].

3.2. Minimum Execution Time (MET)

In contrast to OLB, Minimum Execution Time (MET) assigns each task, in arbitrary order, to the machine with the best expected execution time for that task, regardless of that machine availability. The motivation behind MET is to give each task the best machine. This can cause a severe load imbalance across machines [3].

3.3. Minimum Completion Time (MCT)

Minimum Completion Time (MCT) assigns each task, in arbitrary order, to the machine with the minimum expected completion time for that task. This causes some tasks to be assigned to machines that do not have the minimum execution time for them [3].

3.4. Min-min

Min-min heuristic uses minimum completion time (MCT) as a metric, meaning that the task which can be completed the earliest is given priority. This heuristic begins with the set *U* of all unmapped tasks. Then the set of minimum completion times (M), is found.

$$M = \left\{ \begin{matrix} min\left(completion\_time\left(T_i, M_j\right)\right) \mid T_i \in U \\ 1 \le i \le n \quad , \quad 1 \le j \le m \end{matrix} \right\}$$

M consists of one entry for each unmapped task. Next, the task with the overall minimum completion time from M is selected and assigned to the corresponding machine and the workload of the selected machine will be updated. And finally the newly mapped task is removed from U and the process repeats until all tasks are mapped [11, 12].

3.5. Max-min

This algorithm begins with a set of all unmapped tasks. Then, the completion time for each job on each machine is calculated: the machine that has the minimum completion time for each job is selected. From the set, the algorithm maps the job with the overall maximum completion time to the selected machine. The mentioned procedure is repeated with the remaining unmapped tasks. Similar to Min-min, Max-min also considers all unmapped tasks at a time [3].

3.6. SA Based Methods

Simulated Annealing (SA) is an iterative technique that considers only one possible mapping for each meta-task at a time. Simulated annealing uses a procedure that probabilistically allows poorer solutions to be accepted to attempt to obtain a better search of the solution space based on a system temperature. It exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system. SA's major advantage over other methods is an ability to avoid becoming trapped at local minima. The initial SA

procedure is as follows. The first mapping is generated from a uniform random distribution. The mapping is mutated in the same manner as the GA, and the new makespan is evaluated. If the new makespan is better, the new mapping replaces the old one. If the new makespan is worse (larger), a uniform random number z ∈ [0, 1) is selected. Then, z is compared with y, where:

$$y = \frac{1}{1 + e^{\left(\frac{old_{makespan} - new_{makespan}}{temprature}\right)}}$$

If z>y the new (poorer) mapping is accepted; otherwise it is rejected, and the old mapping is kept. After each mutation, the system temperature is reduced to 90% of its current value. (This percentage is defined as the cooling rate.) This completes one iteration of SA. The heuristic stops when there is no change in the makespan for 150 iterations or the system temperature approaches zero [14].

3.7. GA Based Method

The proposed method in [15] is based on Genetic algorithms (GAs) that is a technique used for searching large solution spaces. This method operates on a population of chromosomes for a given meta-tasks. The initial population is generated by two methods. In the first method, a chromosome is generated randomly from a uniform distribution. In the second method, a chromosome is generated by Min-min and it is called "seeding" the population with a Min-min chromosome. The GA implemented here operates on a population of 200 chromosomes (possible mappings) for a given meta task. In the GA algorithm, the solution is represented as an array of length equal to the number of jobs. Each chromosome is a *n×1* vector, where position *i* ($0 \leq i < n$) represents task t_i , and the entry in position *i* is the machine to which the task has been mapped. The job-to-resource representation (Chromosome) is illustrated in Figure 1.

2	3	1	6	4	5	1	13

Figure 1. Job-to-Resource representation (chromosome)

Assume schedule *S* from the set of all possible schedules Sched. Each chromosome has a fitness value, which is the makespan that results from the matching of tasks to machines within that chromosome. For GA, define a fitness function *fitness(S): Sched → R* that evaluates each schedule:

$$fitness(S) = Makespan(S)$$

After the generation of the initial population, all of the chromosomes in the population are evaluated based on their fitness value, with a smaller fitness value being a better mapping.

3.8. GSA method

The Genetic Simulated Annealing (GSA) heuristics is a combination of the GA and SA heuristics. GSA follows the procedures similar to the GA. For the selection process, GSA uses the SA cooling schedule and system temperature. Specifically, the initial system temperature was set to the average makespan of the initial population and reduced to 900 of its current value for each iteration. Whenever a mutation or crossover occurs, the new chromosome is compared with the corresponding original chromosome. If the new makespan

is less than the original makespan plus the system temperature, then the new chromosome is accepted. Otherwise, the original chromosome survives to the next iteration [3].

Though the above stated heuristic algorithms have advantages, they have their own disadvantages. Figure 1 shows the geometric mean of makespan for the 10 considered cases using Braun et al. benchmark. For the situations considered, the relative performance of the mapping heuristics varied based on the characteristics of the HC environments. The GA always gave the best performance. If mapped execution time is also considered, Min-min gave excellent performance (within 12% of the best) and had a very small execution time. SA and GSA had similar results.

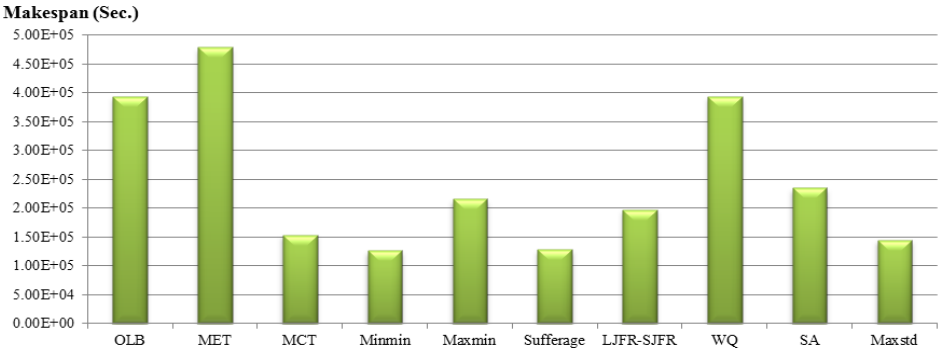

Figure 2. Makespan obtained by some heuristic algorithms

4. Proposed Approach

A meta-task is defined as a collection of independent task (i.e. task that does not require any communication with other ones) [16]. For static mapping, n : the number of tasks and m: the number of machines are known as a priori. Assume that $C_{i,j}$ ($i \in \{1,2,...,n\}$, $j \in \{1,2,..., m\}$) is the completion time for performing ith task in jth machine and W_j ($j \in \{1,2,...,m\}$) is the previous workload of M_j, then Eq. (1) shows the time required for M_j to complete the tasks included in it. According to the aforementioned definition, makespan can be estimated using Eq. (2) [2, 3].

$$\sum C_j + W_j \qquad (1)$$

$$Makespan = \max_{j \in 1,\dots,m} \{\sum C_j + W_j\} \qquad (2)$$

Here, the presented method proposes a new idea as a heuristic solution to deal with the imprecision situation of mapping tasks and machines. The proposed idea is based on converting execution times and states of machines in each iteration of solving approach into functions, so that prioritizing and mapping each task in appropriate machine improves. In each iteration, execution times and state of each machine are normalized. These normalized values are used to provide a mapping function *(Map)* which considers both execution time and machine state simultaneously. It is obvious that state of a machine is high when there is no task on it and it is low in high load. Therefore, a maximum value of *Map* shows a machine with low state time together with a task with high execution time. The steps of the proposed algorithm are as follows: In the first step, convert the $T \times M$ execution times matrix of T tasks in M machines into EF matrix. In the second step, convert the $1 \times M$ matrix of machine work

time into WF matrix. Then the product of these two matrixes are found as *Map* matrix. The task which its value is maximum, is selected and removed from set of unmapped tasks.

The sequence of the execution of the proposed heuristic scheduling algorithm is as follows:

(1) While there are tasks to schedule

(2) U=Empty

(3) For all $Task_i$ to schedule

(4) For all $Machine_j$

(5) $EF_{ij} = \frac{E_{max} - E_{ij}}{E_{max} - E_{min}} \quad \forall i = 1 \ldots n \ , \ \forall j = 1 \ldots m$

(6) $WF_j = \frac{W_{max} - W_j}{W_{max} - W_{min}} \quad \forall j = 1 \ldots m$

(7) End for

(8) $Map_{ij} = EF_{ij} * WF_i \quad \forall i = 1 \ldots n \ , \ \forall j = 1 \ldots m$

(9) End for

(10) Select the maximum value from *Map* matrix

(11) Allocate related task and update workload

(12) Delete task from unmapped tasks

(13) End while

5. Experimental Results

In this section, after the benchmark description, heuristic scheduling algorithms were compared with the proposed algorithm. These heuristics are implemented using Matlab environment and run on 12 different types of ETC matrices. For each heuristic and each type of ETC matrix, the results were averaged over 50 different ETC matrices of the same type (*i.e.*, 50 mappings). The experimental results discussed below were obtained on a PC with 2.4 GHz processor and 4 GB of RAM.

5.1. Benchmark Description

In this paper, we used the benchmark proposed in [3]. The simulation model is based on expected time to compute (ETC) matrix for 512 tasks and 16 machines. The instances of the benchmark are classified into 12 different types of ETC matrices according to the three following metrics: task heterogeneity, machine heterogeneity, and consistency. In ETC matrix, the amount of variance among the execution times of tasks for a given machine is defined as task heterogeneity. Machine heterogeneity represents the variation that is possible among the execution times for a given task across all the machines. Also an ETC matrix is said to be consistent whenever a machine M_j executes any task T_i faster than machine M_k; in this case, machine M_j executes all tasks faster than machine M_k. In contrast, inconsistent matrices characterize the situation where machine M_j may be faster than machine M_k for some tasks and slower for others. Partially-consistent matrices are inconsistent matrices that

include a consistent sub-matrix of a predefined size. Instances consist of 512 tasks and 16 machines and are labeled as x-yy-zz as follows:

- *x* shows the type of inconsistency; *c* means consistent, *i* means inconsistent and *p* means partially-consistent

- *yy* indicates the heterogeneity of the tasks; *hi* means high and *lo* means low.

- *zz* represents the heterogeneity of the machines; *hi* means high and *lo* means low.

For example, c-lohi means low heterogeneity in tasks, high heterogeneity in machines and consistent environment.

5.2. Makespan

Among most popular and extensively studied optimization criterion is the minimization of the makespan. Small values of makespan mean that the scheduler is providing good and efficient planning of tasks to resources. The obtained makespan using mentioned heuristics are compared in Table 1. The results are obtained as an average of 50 simulations.

Table 1. Comparison of makespan values obtained by heuristics

Instance	MET	MCT	Min-min	Proposed
c_hihi	45304079	11275776	8370083.5	8585059.5
c_hilo	929580.9	156778.1	132762.36	134202.2
c_lohi	1528951	381638.5	280522.46	285054.68
c_lolo	31396.96	5284.24	4470.48	4534.08
i_hihi	4599640	4317386	3626967.62	3752637.8
i_hilo	124957.5	73400.86	66018.66	66390.08
i_lohi	147744.2	144064.6	119015.72	122585.6
i_lolo	4250.18	2489.48	2236.28	2253.54
p_hihi	22875294	6288654	4885073.46	4987418.9
p_hilo	594521.9	98204.48	83648.26	85101.06
p_lohi	769969.8	210893.1	165045.92	167728.7
p_lolo	20097.68	3331.3	2818.36	2844.2

Figure 3, 4, 5, 6 represents the makespan values of proposed heuristic scheduling algorithm over others in all 12 different types of instances based on the three metrics: Task heterogeneity, machine heterogeneity and consistency. The proposed heuristic performed very well for ETC matrices, giving the second best result in each case

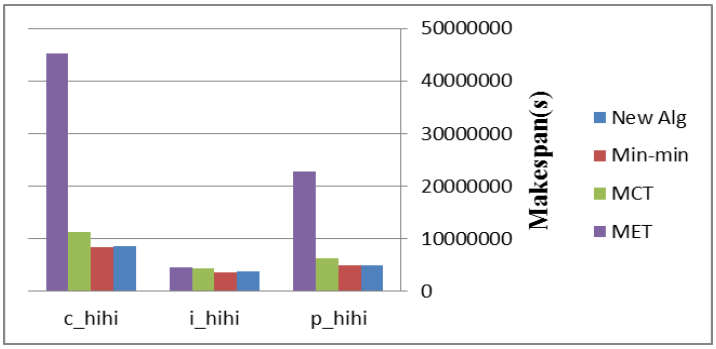

Figure 3. makespan values obtained by heuristics (hihi)

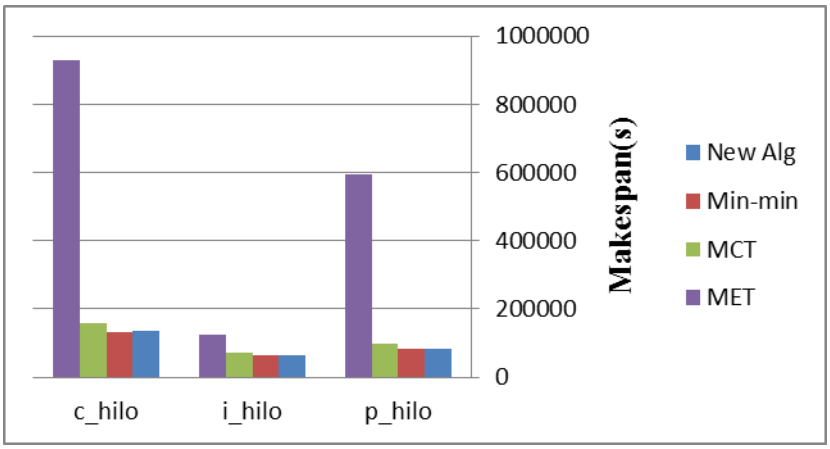

Figure 4. makespan values obtained by heuristics (hilo)

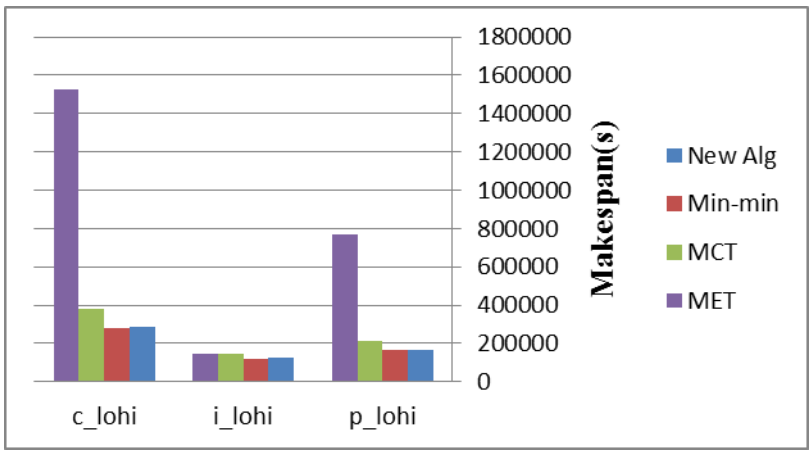

Figure 5. makespan values obtained by heuristics (lohi)

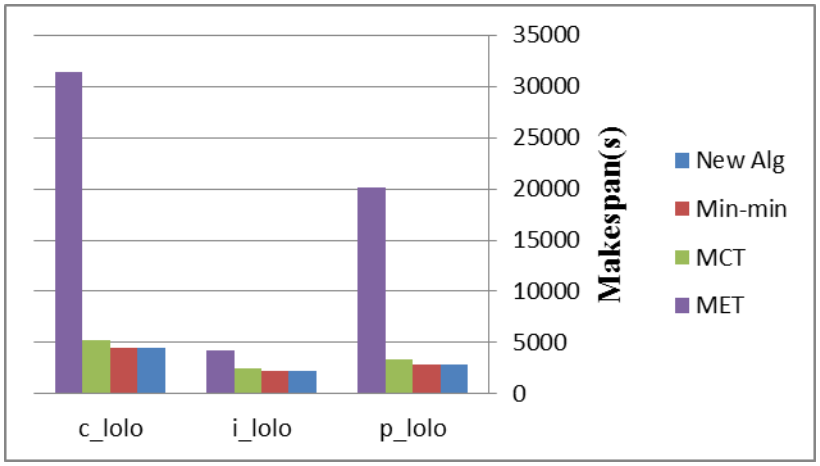

Figure 6. makespan values obtained by heuristics (lolo)

5.3. Execution Times

When comparing mapping heuristics, the execution time of the heuristics themselves is a challenge. For the 4 heuristics that were compared, the execution times varied greatly. The heuristic execution times are the average time each heuristic took to compute a mapping for a single 512 task×16 machine ETC matric, averaged over 50 different matrices (all for inconsistent, high task, high machine heterogeneity). Table 2 shows the results. Comparing these algorithms shows that Min-min is more time consuming than proposed method, however, it has a little better makespan.

Table 2. Comparison results between heuristics on execution time

Heuristic	Execution Time(s)
MET	0.0021
MCT	0.0070
Min-min	0.7736
Proposed	0.2209

6. Conclusion

Task Scheduling is a critical design issue of distributed computing. A computational grid is a highly distributed environment. This paper investigates the task scheduling algorithm in grid environments as an optimization problem. Selecting the appropriate machine for the specific task is one of the challenging works in computational grids. This paper proposes a new method to schedule the tasks in computational grids. The goal of the scheduler in this paper is minimizing makespan and execution time. The proposed heuristic scheduling algorithm and various existing algorithm have been tested using the benchmark simulation model for distributed heterogeneous computing systems by Braun *et al.*, (2001). The experimental results showed that the proposed algorithm retains the advantage of Min-min while reduces the execution time, which in turn leads to better performance. The future research will be directed towards the factors such as CPU workload, communication delay and so on.

References

[1] I. Foster and C. Kesselman, "The Grid - Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers, **(1998)**.
[2] H. Izakian, A. Abraham and V. Snasel, "Comparison of Heuristics for Scheduling Independent Tasks on Heterogeneous Distributed Environments", Proceedings of the International Joint Conference on Computational Sciences and Optimization, IEEE, vol. 1, **(2009)**, pp. 8-12.
[3] R. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen and R. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems", J. of Parallel and Distributed Computing, vol. 61, no. 6, **(2001)**, pp. 810-837.
[4] E. U. Munir, L. Jian-Zhong, S. Sheng-Fei and Q. Rasool, "Performance Analysis of Task Scheduling Heuristics in Grid", Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC), vol. 6, **(2007)**, pp. 3093-3098.
[5] H. Baghban and A. M. Rahmani, "A Heuristic on Job Scheduling in Grid Computing Environment", Proceedings of the seventh IEEE International Conference on Grid and Cooperative Computing, **(2008)**, pp. 141-146.
[6] M. Macheswaran, S. Ali, H. J. Siegel, D. Hensgen and R. F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems", J. of Parallel Distributed Computing, vol. 59, no. 2, **(1999)**, pp. 107-131.
[7] E. U. Munir, "Maxstd: A Task Scheduling Heuristic for Heterogeneous Computing Environment", Information Technology J., **(2008)**, ISSN-1812-5638.

[8] F. Xhafa and A. Abraham, "Meta-heuristics for Grid Scheduling Problems", In Meta-heuristics for Scheduling in Distributed Computing Environments, Springer, vol. 146, **(2008)**, pp. 1-37.
[9] Q. Zhang and L. Zhen, "Design of Grid Resource Management System Based on Divided Min-min Scheduling Algorithm", IEEE First International Workshop on Education Technology and Computer Science, **(2009)**, pp. 613-618.
[10] R. Armstrong, D. Hensgen and T. Kidd, "The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-time Predictions", Proceedings of the 7th IEEE Heterogeneous Computing Workshop (HCW '98), **(1998)**, pp. 79-87.
[11] R. F. Freund and H. Siegel, "Heterogeneous Processing", IEEE Computer, vol. 26, no. 6, **(1993)**, pp. 13-17.
[12] R. F. Freund, *et al.*, "Scheduling Resources in Multi-user Heterogeneous Computing Environment with Smart Net", **(1998)**, Proceedings of the 7th IEEE HCW.
[13] A. Abraham, R. Buyya and B. Nath, "Nature's Heuristics for Scheduling Jobs on Computational Grids", Proceedings of the 8th IEEE International Conference on Advanced Computing and Communications, **(2000)**.
[14] S. Fidanova, "Simulated Annealing for Grid Scheduling Problem", IEEE John Vincent Atanasoff International Symposium on Modern Computing, **(2006)**.
[15] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms", in 5th IEEE Heterogeneous Computing Workshop (HCW '96), **(1996)**, pp. 86-97.
[16] F. Xhafa, L. Barolli and A. Durresi, "Immediate Mode Scheduling in Grid Systems", International J. of Web and Grid Services, vol. 2, **(2007)**, pp. 219-236.
[17] S. Ali, H. J. Siegel, M. Maheswaran and D. Hensgen, "Modeling Task Execution Time Behavior in Heterogeneous Computing Systems", Tamkang J. Science and Engineering, **(2000)**, pp. 195-207.
[18] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys and B. Yao, "A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-machine Heterogeneous Computing Systems", Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems, **(1998)**, pp. 330-335.
[19] L. Wenzheng and Z. Wenyue, "An Improved Scheduling Algorithm for Grid Tasks", International Symposium on Intelligent Ubiquitous Computing and Education, **(2009)**, pp. 9-12.
[20] F. Xhafa and A. Abraham, "Computational Models and Heuristic Methods for Grid Scheduling Problems", Proceedings of Future Generation Computer System, **(2010)**, pp. 608-621.
[21] M. Tracy, T. D. Braun and H. Siegel, "High-performance Mixed-machine Heterogeneous Computing", 6th Euro-micro Workshop on Parallel and Distributed Processing, **(1998)**, pp. 3-9.
[22] D. Fernandez-Baca, "Allocating Modules to Processors in a Distributed System", IEEE Trans., Software Engineering, **(1989)**, pp. 1427–1436.
[23] T. Hagerup, "Allocating Independent Tasks to Parallel Processors", An Experimental Study, J. of Parallel and Distributed Computing, vol. 47, **(1997)**, pp. 185-197.