Apportionment Heuristics for Mapping Tasks in Heterogeneous Computing Systems

Igor Mishkovski[1], Sonja Filiposka[2], Dimitar Trajanov[2], Ljupco Kocarev[2, 3],

[1] Politecnico di Torino, Turin, Italy
igor.mishkovski@polito.it
[2] Faculty of electrical engineering and information technologies, Skopje, Macedonia
filipos@feit.ukim.edu.mk, mite@feit.ukim.edu.mk, lkocarev@feit.ukim.edu.mk
[3] Macedonian Academy of Sciences and Arts, Skopje, Macedonia
lkocarev@manu.edu.mk

Abstract. One of the biggest problems in heterogeneous computing is how tasks should be mapped in these kinds of environments. Because this problem of mapping tasks has been shown to be NP-complete, it requires heuristic techniques. Therefore, we present new schedulers based on the apportionment methods used in elections. In order to obtain the performances of these schedulers we compare them with other known and used heuristics in many different parameters. The presented heuristics can be used when the tasks are big and when they can be divided in smaller sub-tasks. The idea behind the new schedulers is to use apportionment methods (used for elections), such as: the Hamilton's method, Jefferson's Method, Webster's method, Huntington-Hill method, Balance method and pure proportional method. Intuitively the Hamilton's method favors the bigger tasks (i.e. gives them more CPU power). The comparison in this paper shows that these apportionment methods can cope well with the other methods when the number of tasks in the system is no bigger than a certain level. The new apportionment scheduler, based on Hamilton's method, copes well with the existing ones and it outperforms the other schedulers when some conditions are met.

Keywords: Heterogeneous Computing, Grid Computing, Schedulers, Mapping Heuristics, Apportionment Methods, Simulated Annealing.

1 Introduction

Distributed systems appear in the computer history as a result of many factors and influences. One of the most important factors was the price reduction of a computer system; this increased the number of computers that can be integrated in distributed. Heterogeneous computing (HC), as a part of distributed system, uses different types of machines, networks and interfaces in order to maximize their combined performance and cost-effectiveness [1, 2, 3]. The machines in this type of systems are independent and for the end user of the system they look as if one machine. Heterogeneous computing is used for solving computationally intensive problems. Thus the most important parameter of one HC system is its speed. The speed of the

HC system heavily depends on the schemes that assign tasks to machines (a.k.a. *matching*) and specify the order by which these tasks will execute on the proper machines (a.k.a. *scheduling*) [4]. This problem of matching and scheduling is known as mapping and it has been shown that is NP-complete [5, 6].

Given a set of independent tasks and a set of available resources, there are heuristics which try to minimize the total execution time of the set by finding an optimal mapping of tasks to machines. This process of mapping can be *dynamic* (map tasks as they arrive) or *static* (mapping is done prior to the execution of any of the tasks) [7, 8]. Static approach is adequate if the tasks to be mapped are known in advance, and if a good estimation exists for the resource's power and capacity. Dynamic mapping is used when if the tasks cannot be determined in advance and if the system performances are fluctuating.

The issue of mapping tasks in heterogeneous systems has been of major interest [9] and [11]-[19]. In [11] the scheduling is based on a predictor, which predicts the variance to make scheduling decisions and it fine-tunes the algorithm using a feedback mechanism. In [12] authors compare eleven static heuristics and they give a common ground for comparison and insights into circumstances where one technique outperforms another. Dynamic mapping heuristics are studied in [9]. In this paper two types of mapping heuristics are considered, immediate mode and batch mode heuristics. Furthermore, three new heuristics are introduced. Their simulation results reveal that the choice of which heuristic to use depends on: the structure of the heterogeneity among tasks and machines, and the arrival rate of the tasks. In [13] a compensation based algorithm is presented which uses a feedback mechanism to predict the execution time of the jobs. The authors in [14] describe Grid Computing systems that have a game theoretic approach in processor scheduling. They also provide an analysis and comparison of these systems. In [15] low-complexity efficient heuristics for task scheduling among heterogeneous processors are presented. The comparison study showed that their algorithms outperform the previous approaches in terms of performance and cost. In [16] the authors propose a metascheduler which uses various preemptive methods to minimize job execution time, such as stopping a bigger job so that a smaller one can proceed first. In [17] different scheduling schemes are characterized with respect to varying arrival times and burstiness in the job arrival rate. Using the insight, authors propose approaches to improve the strategies regarding turn-around time. In [18] authors propose, evaluate, and compare eight dynamic mapping heuristics. The tasks in the study have priorities and multiple soft deadlines. They calculate the value of the task using the priority of the task and the completion time of the task with respect to its deadlines.

In the simulations in this paper, it is assumed that no communication exists between the tasks, the tasks are very big (i.e. the tasks have many instructions) and that they can be additionally divided into smaller sub-task, each machine in the system executes one task at a time in the order in which tasks are assigned. The dynamic mapping applied in this paper is non-preemptive and it works in *batch mode* [9]. In the batch mode the scheduler does not map the tasks as they arrive, but instead it maps the tasks in prescheduled times, also called *mapping events*. The key feature of the scheduler is to finish the execution of the tasks as soon as possible, i.e. to reduce the *makespan* [10].

The major contribution of this work is introducing new heuristic which is based on apportionment methods. Another gain of this paper is that the comparison between the new heuristic and the existing ones is done using many different parameters, such as: the system load; the average number of busy machines; the average turnaround time of the executed tasks; the average response time of the executed tasks; how many tasks were finished; the number of deployments of the tasks that were involved and the load balance among the machines. Another important parameter that we consider is the *heuristic execution times.* This parameter tells how much time is needed for a certain heuristic to map the arrived tasks on the available machines. .

The remainder of the paper is organized as follows. Description of the apportionment methods used in elections and summary of their characteristics is found in Section 2. Section 3 describes how apportionment methods can be adjusted for the purpose of mapping tasks in heterogeneous computing environments. In Section 4 the simulation scenario is presented and this section examines the obtained results from the simulation study. In Section 5 the conclusion and future work are presented.

2 Methods of Apportionment

The problem of how many representatives should be allotted to some important institution (i.e. US Congress) exists since the beginning of democracy and elections. One might come up with an idea of one man, one vote, i.e. pure proportional method. But candidates are human beings and they cannot be divided. Thus, because this apportionment cannot be done perfectly it must be done in a manner as near perfection as can be. These methods of apportionment should fulfill certain rules in order to be objective [20], as example we refer to the US Congress:

1. No state's number of representatives should decrease, if the total number of representatives increases.
2. Every state should have within one (exclusive) of their quotient. For example if a state should receive 3.4 representatives it can receive 3 or 4. If the state should receive exactly 3 representatives, it should receive 3, but not 2 or 4. In future, we will call having this property -satisfying Quota.
3. All states abide by the same formula for representation.
4. Methods do not artificially favor large states at the expense of the smaller ones and vice versa.

2.1 Hamilton's method (HM)

The algorithm of this method is as follows:

1. Calculate the Standard Divisor (SD).
2. Calculate each state's Standard Quota (SQ).
3. Initially assign each state its Lower Quota (LQ).
4. If there are surplus seats, give them, one at a time, to states in descending order of the fractional parts of their Standard Quota.

The standard divisor is the average number of people per seat over the entire population and it can be calculated as *SD=TP / p*, where, *TP* is the total population and *p* is the number of seats in the congress.

The standard quota is the fraction of the total number of seats a state would be entitled if the seats were not indivisible and it is calculated as *SQ=SP / SD*, where, *SP* is the state population.

The lower quota (LQ) is the standard quota rounded down.

But this method violates the rule number one: an increase in the total number of seats to be apportioned causes a state to lose a seat (a.k.a. Alabama paradox).

2.2 Jefferson's method (JM)

Jefferson came up with what is known as the method of greatest divisors. Suppose we are given state populations p_1, p_2, .., p_N and representative apportionment a_1, a_2, .., a_N. We can calculate a divisor *L(s) = a(s)+1* for each state *s*. Then states can be ranked using the *p(s)/L(s)* ratios. The higher this ratio, the more deserving this state is to get another representative. In this method everybody starts with zero representatives. The representatives are always assigned to the state with the current highest ratio (rank-index). The first *N* representatives are assigned one to each state. This is naturally enforcing US Constitution rule about each state having a minimum of one representative. The divisor choice *L(s) = a(s)+1* is natural, because it ranks how much better off the state will be if it was given one more representative. The divisor choice Jefferson's Method uses is arbitrary, since other methods use divisors such as $L(S) = a(s) + 1/2$ - the Webster's method (WM), or $L(S) = \sqrt{a(s)*(a(s)+1)}$ - the Huntington-Hill method (HHM). The Jefferson's method favors large states and it does not satisfy lower quota. Webster's method does not satisfy upper or lower quota, but it does not favor large or small states and the Huntington-Hill method favors small states.

3 Using Methods of Apportionment as Heuristics for Mapping Tasks

The difference between mapping tasks in HC environments and allocation of places in some institution is that in HC environment machines can be divided among multiple tasks. Thus, the pure proportional model (PPM) can take part in the process of mapping tasks. With this proportional method each task gets a portion of the heterogeneous system as a whole. The portion that task *i* gets for the system is equal to:

$$port_i = \frac{size_i}{\sum_j size_j} \qquad (4)$$

where, $size_i$ is the size of the task *i* and the sum in the nominator is the sum of the instructions of the arrived tasks. It is obvious that with this method one machine can be shared among many tasks and the tasks are executed in Round-Robin fashion. This additionally requires time for context switch which certainly reduces the overall performance of the system. Having this fact in mind in this part we will briefly present how these apportionment methods used in the US Congress can be adjust for mapping tasks in HC environment.

The analogy is that instead the US Congress is now the HC environment and the available seats are the machines in the HC environment. The states are replaced by the tasks and the total population of a state is an equivalent to the number of instructions that a certain task has.

For example we will use the Hamilton's method. *TP* is the sum of the number of instructions of the available tasks and *p* is the number of available processors. Then the standard quota is calculated as the fraction of the total number of processors a task would be entitled to if the processors were not indivisible and for tasks *I*, i.e. $SQ_i=TS_i / SD$. The lower quota (*LQ*) is the standard quota rounded down.

For example, in Table 1 the task mapping is shown for 5 processors and 10 users in the system and where every user has a different task to be executed on the system.

Table 1 Hamilton's method example

Task	**# of million instructions**	**SQ**	**LQ**	**Leftover**	**Surplus**	**Final mapping**
1	6	0.02	0	0.02		0
2	271	0.68	0	0.68		1
3	85	0.21	0	0.21		0
4	153	0.39	0	0.39		0
5	200	0.50	0	0.50		0
6	406	1.02	1	0.02		1
7	161	0.41	0	0.41		0
8	242	0.61	0	0.61	1(3)	1
9	217	0.55	0	0.55	1(4)	1
10	243	0.61	0	0.61	1(2)	1
Totals	**1831**	**5**	**1**	**4**	**4**	**5**

As we can see, only tasks 2, 8, 9 and 10 would be executed in the system for that certain round, while the other tasks have to wait for another round. Intuitively, this heuristics gives excellent performance when the arrival of the tasks is not so dense and there is a big number of free machines. In this case one bigger task can be divided to several smaller ones and they can be executed in parallel on the machines in the HC environment. On the other hand, there will be some delay because of the task partitioning. If this delay is not so big then this is a perfect case when this apportionment heuristics can be used. But, when the number of tasks is very big and the HC environment is overloaded then these heuristics act as Biggest Job First heuristic. Thus, in their nature these heuristics change their attitude dynamically, i.e. in an overloaded cluster they use one-to-one mapping, and many-to-one mapping when the number of free machines is bigger. The Jefferson's approach will favour bigger tasks.

4 Simulation Model and Results

In order to analyze how the proposed heuristics cope with other heuristics used for mapping tasks in HC environments we have implemented the following ones: **OLB** (*Opportunistic Load Balancing*), **MET** (*Minimum Execution Time*), **MCT** (*Minimum Completion Time*), **Min-min**, **Max-min**, **SA** (*Simulated Annealing*).

All of these heuristics do not consider the case when tasks are very big and they can be divided in a certain number of sub-tasks. In that event, one task can be executed on a certain number of processors – space sharing. This can be done in computing systems where communication delays can be neglected. If this is a case one of the proposed apportionment mappings (in Section II and III) can be used for mapping tasks in HC environments.

All of the schedulers have been simulated in Matlab. Firstly, we create a virtual system with *p* heterogeneous machines and *u* users of the systems. We differentiate between small and big users of the system. The big users (called predators) generate big tasks with greater probability than the small users (called victims). The simulation is discrete, i.e. we use time steps in our simulation and every time step can last a certain number of seconds. The arrival time of the tasks is simulated using a uniform distribution. Each task is characterized by its size expressed in instructions and machines are characterized by how many instructions in second they can execute. In every time step the simulator schedules the arrived task to a certain machine depending on the chosen scheduler.

We analyze the schedulers using these eight parameters:

1. The system load;
2. The average number of busy machines;
3. The average turnaround time of the executed tasks;
4. The average response time of the executed tasks;
5. The number of finished tasks;
6. Provided that all tasks were finished, the time when they finished;
7. The number of deployments of the tasks that were involved
8. The load balance among the machines.

The simulation lasts 1000 rounds, the number of users in the system is 25 and every user generates 4 tasks. The number of the machines present in the system is 10. There are 20 predators and 5 victims. There are 20 users that generate big tasks (from 100 to 1000 million instructions) with probability of 0.9 and small tasks (from 1 to 100 million instructions) with 0.1. There are 5 users that generate small tasks with probability 0.9 and big ones with 0.1. The CPU power of each machine is obtained from the normal distribution where the mean is 10 million instructions in second and the variance is 3 million instructions.

First, we investigate the percentage of the simulation when the system was busy. In fig. 1 we can see that with the proportional scheduler the system is kept busy near 100% of the time, while with the Hamilton’s method (HM) this percentage is lower (about 50%), the other schedulers kept the system busy for almost all of the time. This means that in this scenario HM leaves space (i.e. computational power) for other tasks and it used only half of the time steps to execute the given set of tasks.

Another parameter is the average number of machines that were kept busy during the simulation. In fig 2 we can see that the PPM scheduler keeps most of the machines busy, the HM keeps busy about 4 machines, while the MET scheduler keeps busy only the best machine for the whole simulation.

Fig.3 shows how many tasks were finished during the simulation. We can see that the worst performance is given by the MET scheduler, while only HM and PPM succeed to execute all the given tasks.

Another important parameter is the average turnaround time achieved by a certain scheduler (fig. 4). As expected, the best performance is given by the HM scheduler, and the worst performance is obtained using MET and then by Min-min. The HM is better than the rest because it may map one task to a number of machines and it is better from PPM because no time is lost for context switch which is needed for the Round-Robin used in the PPM. However, PPM and HM must divide the task and this means that we assume that tasks can be parallelized.

Fig.5 shows that the HM finishes all the tasks earlier than the other schedulers that succeed to finish all the tasks before the last round. The MET heuristic does not to finish all the tasks till the end of the simulation.

The average rounds that the tasks spend in the queue (i.e. the time between its deployment and arrival) is shown in fig.6. We can see that the HM copes well with the other heuristics. The best performances are given by Max-Min and the worst by Min-Min.

Depending on the type of the scheduler, tasks can be deployed to one or to many machines. Here the PPM and the HM scheduler do not cope with the rest of the schedulers (fig. 7). These results are obvious and mean that if all of these schedulers were implemented, the most time will be lost for deploying the tasks using the PPM and HM method.

From the next figure we can see that the HM and PPM schedulers give good load balance among the machines, expectedly the worst load balance is given by the MET scheduler. We can see that good balance is also shown by the following: Min-Min, Max-Min, Simulated Annealing and MCT favors machines 3, 4 and 8 because these machines work on higher frequencies than others.

In fig. 9 the heuristics execution time is shown. The graph was made relative to Min-Min heuristics because this heuristic shows worse results than the other heuristics. It is obvious that the MCT heuristic loses the smallest amount of time for scheduling and it is also easy to implement. It is worth mentioning that the Simulated Annealing is very dependable on its scheduling parameters (i.e. heating rate and number of rounds). In this simulation the HM heuristic shows similar results as Simulated Annealing.

For broader analysis we reduced the number of machines present in the system from 10 to 5. This means that the system was more overloaded than in the first scenario and that the HM scheduler will dynamically change the mapping from one task-to-many machines to one task-to-one machine and that it is similar to bigger tasks first kind of scheduler and certainly its performances will degrade. Results showed that Max-Min outperforms others in most of the performances and that HM copes well with the others in load balance. HM also finished the jobs earlier that the other schedulers.

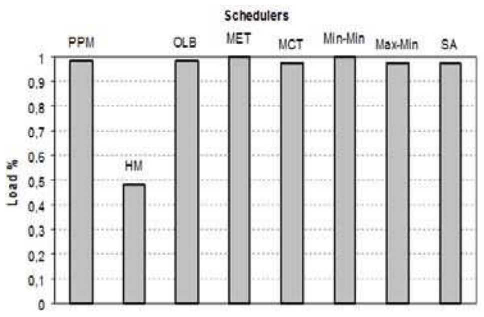

Fig. 1. The load of the system

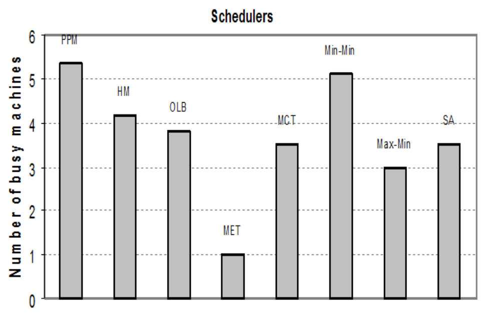

Fig. 2. Average number of busy machines

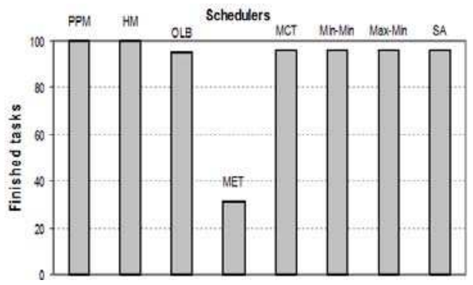

Fig. 3. Number of finished jobs

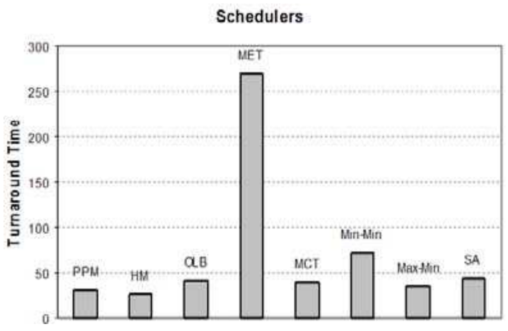

Fig. 4. The average turnaround time

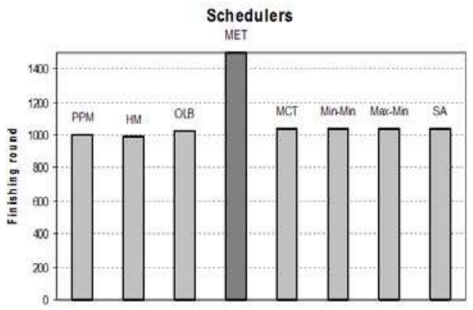

Fig. 5. Finishing round

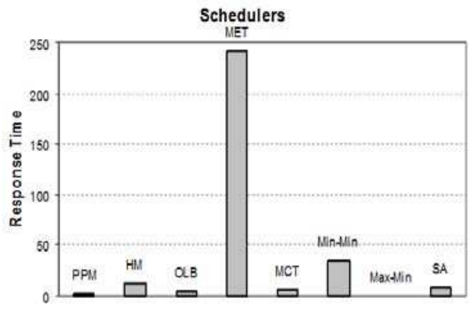

Fig. 6. Average response time

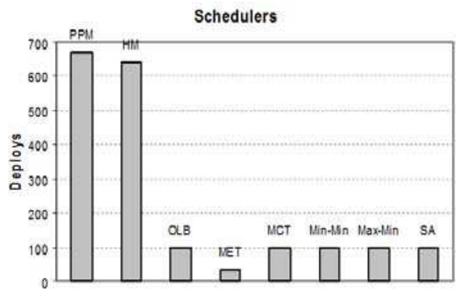

Fig. 7. Number of deploys

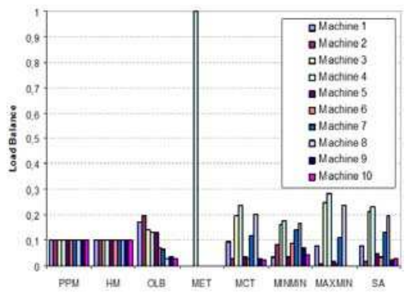

Fig. 8. Load balance among the machines

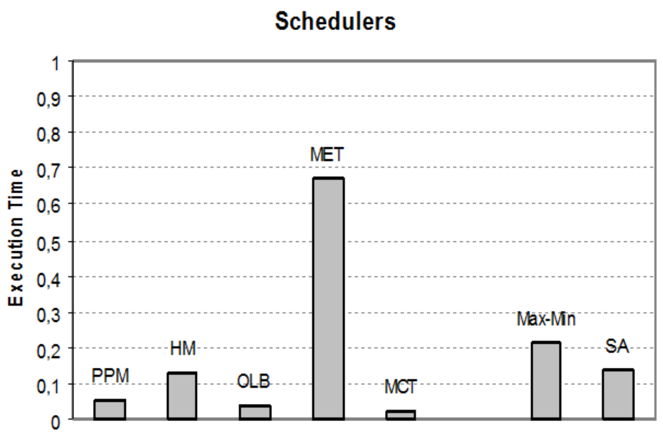

Fig. 9. Execution time for different heuristics

5 Conclusion

In this paper we have implemented new heuristic based on the apportionment method used in elections. We also implemented other schedulers which are known and used in HC environments. The mappings were dynamic in batch mode. The implemented method is called the Hamilton's method. The other mappings were: OLB, MET, MCT, Min-min, Max-min and Simulated Annealing. We also suggested a way how and when these apportionment methods can be used in mapping tasks in HC environments. Additionally, they can be used in every situation where resources need to be shared.

Additionally, we made a comparison of these heuristics in different parameters: the system load; the average number of busy machines; the average turnaround time of the executed tasks; the average response time of the executed task; the number of finished task; provided that all tasks were finished, the time when they finished; the number of deployments of the tasks that were involved and the load balance among the machines. The comparison is made using simulation in matlab and the results show that the Hamilton's method copes well with the other schedulers and it has good load balance. Moreover it finishes all the tasks earlier than any other scheduler.

This comparison shows under which circumstances one should choose the right scheduler in Heterogeneous Computer Systems.

In our future work we intend to compare this schedulers with all of the remaining schedulers (i.e. HEFT, CPOP, Sufferage, A*, Tabu etc.) and to use some other of the apportionment methods described in Section III as a heuristic for mapping tasks in heterogeneous computing systems.

References

1. M. M. Eshaghian, Ed., Heterogeneous Computing, Artech House, Norwood, MA, 1996.
2. R. F. Freund and H. J. Siegel, Heterogeneous processing, IEEE Comput. 26, 6 (June 1993), 13_17.
3. M. Maheswaran, T. D. Braun, and H. J. Siegel, Heterogeneous distributed computing, in Encyclopedia of Electrical and Electronics Engineering (J. G. Webster, Ed.), Wiley, New York, Vol. 8, pp. 679_690, 1999.
4. T. D. Braun, H. J. Siegel, N. Beck, L. Bo_ lo_ ni, 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, in "1998 IEEE SRDS," pp. 330_335, 1998.
5. D. Fernandez-Baca, Allocating modules to processors in a distributed system, IEEE Trans. Software Engrg. 15, 11 (Nov. 1989), 1427_1436.
6. IBARRA, O H, AND KIM, C E Heuristic algorithms for scheduling independent tasks on nonidentical processors J. ACM 24, 2 (April 1977), 280-289.
7. T. D. Braun, H. J. Siegel, N. Beck, L. Bo_ lo_ ni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, R. F. Freund, and D. Hensgen, A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems, in "8th IEEE HCW '99," pp. 15_29, 1999.
8. L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, J. Parallel Distrib. Comput. 47, 1 (Nov. 1997), 8_22.
9. M. Maheswaran, S. Ali, H. J. Siegel, 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, no. 2, Nov 1999, pp. 107-131.
10. M. Pinedo, Scheduling: Theory, Algorithms, and Systems, Prentice Hall, Englewood Cliffs, NJ, 1995.
11. L. Yang, J. Schopf, I. Foster, "Conservative Scheduling: Using Predicted Variance to improve Scheduling Decisions in Dynamic Environments", ACM/IEEE SC2003 Conference (SC'03).
12. T. Braun, et al.. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, 61(6):810–837, 2001.
13. Y. Tao, X. Wang and J. Gozali, "A Compensation-based Scheduling Scheme for Grid Computing", 7th ICHPCG in Asia Pacific Region, 2004.
14. Sara Forghanizadeh, Grid Processor Scheduling based on Game Theoretic Approach, CPSC532A Final Project, Fall 2005
15. Salim Hariri Haluk Topcuoglu and Min-You Wu. Task scheduling algorithms for heterogeneous processors. 8th IEEE HCW'99, pages 3–14, April 1999.
16. S. Vadhiyar, J. Dongarra, A Metascheduler for the Grid, 11th IEEE international Symposium on High Performance Distributed Computing HPDC-2002.
17. Praveen Holenarsipur , Vladimir Yarmolenko , José Duato , Dhabaleswar K. Panda , P. Sadayappan, Characterization and enhancement of Static Mapping Heuristics for Heterogeneous Systems, Proc. of the 7th Int. Conf. HPC, p.37-48, December 17-20, 2000.
18. Jong-Kook Kim et al., Dynamic Mapping in a Heterogeneous Environment with Tasks Having Priorities and Multiple Deadlines, Proc. of the 17th Int. Symp. on PDP, p.98.1, April 22-26, 2003.
19. http://www.ctl.ua.edu/math103/apportionment/appmeth.htm
20. Roman Shapiro, Methods of Apportionment, Apportionment of Representatives in the United States Congress House of Representatives and avoiding the 'Alabama Paradox'. Engineering, University of Bridgeport, USA 5-13 Dec. 2008.