A Survey of New Scheduling Strategies for Internet-Based Grids of Computers

Javier Díaz, Sebastián Reyes, Alfonso Niño, and Camelia Muñoz-Caro

Universidad de Castilla-La Mancha, Escuela Superior de Informática,
Paseo de la Universidad 4, 13071, Ciudad Real, Spain,
{javier.diaz,sebastian.reyes,alfonso.nino,camelia.munoz}@uclm.es

Abstract. Scheduling algorithms play an important role in heterogeneous computing systems. Here, some recent strategies to schedule tasks in this kind of environments are reviewed. In particular, we consider Quadratic Self-Scheduling (QSS), Exponential Self-Scheduling (ESS) and Root Self-Scheduling (RSS). In the tests performed QSS and ESS outperform the other self-scheduling algorithms. However, these algorithms depend on several parameters, which have to be optimized for the working environment. To tackle this problem, we have developed a heuristic approach, based in Simulated Annealing (SA), to optimize all the parameters of QSS and ESS. We find that the optimal SA results permit to reduce the overall computing time. Finally, we are working to improve this approach by considering the changes of the environment during the execution of the tasks. The preliminar results show a reduction in the overall execution time due to a better load balance.

1 Introduction

An essential issue in distributed high-performance computing is how to allocate efficiently the workload among the processors. This is specially important when the resources are not only distributed, but also heterogeneous, as in a computational Grid [13]. A computational Grid is a hardware and software infrastructure providing dependable, consistent, and pervasive access to resources among different administrative domains. The objective is to enable the sharing of these resources in a unified way, maximizing their use. A Grid can be used effectively to support large-scale runs of distributed applications. An ideal case to be run in Grid is that with many large independent tasks. This case arises naturally in parameter sweep problems. A correct assignment of tasks, so that computer loads and communication overheads are well balanced, is the way to minimize the overall computing time. This problem belongs to the active research topic of the development and analysis of scheduling algorithms. Different scheduling strategies have been developed along the years (for the classical taxonomy see [5]). In particular, dynamic self-scheduling algorithms are extensively used in practical applications. These algorithms represent schemes where tasks are allocated in run-time. Self-scheduling algorithms were initially developed to solve parallel loop scheduling problems in homogeneous memory-shared systems, see for instance [17]. Here, the loop iterations have not interdependencies, and they can be scheduled as independent tasks.

This kind of algorithms divides the set of tasks into subsets (chunks), and allocates them among the processors. In this way overheads are reduced [16]. The simplest of these algorithms is Chunk Self-Scheduling (CSS) [21], which divides the total number of tasks among the available processors, assigning the resulting number of tasks (the chunk) to each processor. We also have Guided Self-Scheduling (GSS) [19] which determine the chunk size dividing the number of remaining tasks by the total number of processors. Moreover, we have Factoring Self-Scheduling (FSS) where the tasks are scheduled in batches (or stages) of P chunks of equal size, where P is the number of processors [14]. The chunk size is determined by dividing the number of remaining tasks by the product of the number of processors and by a parameter (α). The last basic approach is Trapezoid Self-Scheduling (TSS). TSS uses a linear decreasing chunk function [22].

Although self-scheduling algorithms were derived for homogeneous systems, these algorithms have been tested successfully in distributed memory multiprocessor systems and heterogeneous clusters [3, 6, 23]. In addition, some works about their performance on heterogeneous environments such as computational Grids have been reported [7, 8, 18, 20]. However, the problem could be the flexibility of these algorithms (they may have not enough degrees of freedom) to adapt efficiently to a heterogeneous environment. In this sense, we have previously proposed several algorithms, which introduce additional degrees of freedom in the model. The first, called Quadratic Self-Scheduling (QSS) [6, 7], is based on a quadratic form for the chunks distribution function. Therefore, it has three degrees of freedom, which provide high adaptability to distributed heterogeneous systems. The second approach, called Exponential Self-Scheduling (ESS) [8, 9], is based on the slope of the chunks distribution function. In this case, we consider that the chunks distribution function decreases in an exponential way. This algorithm also provides a good adaptability in distributed heterogeneous systems using two parameters. Both cases are reviewed in the next Section.

2 New Self-Scheduling Schemes

This section describes the two different families of Self-Scheduling methods proposed in [8, 9]. Both of them use the chunk distribution function, C(t). This function gives the chunk size as a function of the t-chunk. The first family is based in the chunks distribution function, whereas the second focuses on the slope of C(t). Moreover, we comment the results found in the comparative study performed with other well-established self-scheduling algorithms.

2.1 Methods based in the chunks distribution function

In this case, we look for an explicit form for C(t), which we can consider given by a function f(t), the target function. From a general point of view, we can expand C(t) in a Taylor series as a function of t as shown in equation (1),

$$C(t) = f(t_0) + \left|\frac{df(t)}{dt}\right|_0 (t - t_0) + \frac{1}{2}\left|\frac{d^2 f(t)}{dt^2}\right|_0 (t - t_0)^2 + \ldots \qquad (1)$$

Depending on the size of the expansion we have methods of different order. Thus, for C(t) constant we have pure self-scheduling or chunk self-scheduling. For the linear approach we have TSS. Finally, for the quadratic case a new method called Quadratic Self-Scheduling (QSS) has been proposed [6, 7]. In QSS, C(t) is given by equation (2)

$$C(t) = a + bt + ct^2 \tag{2}$$

where t represent the t-th chunk assigned to a processor. To apply the QSS algorithm we need the a, b and c coefficients of equation (2). Thus, we can select three reference points (C(t), t) and solve for the resulting system of equations. Useful points are $(C_0, 0)$, $(C_{N/2}, N/2)$ and (C_N, N), where N is the total number of chunks. Solving for a, b and c, we obtain,

$$\begin{aligned} a &= C_0 \\ b &= (4C_{N/2} - C_N - 3C_0)/N \\ c &= (2C_0 + 2C_N - 4C_{N/2})/N^2 \end{aligned} \tag{3}$$

where N is defined [7] by,

$$N = 6I/(4C_{N/2} + C_N + C_0) \tag{4}$$

The $C_{N/2}$ value is given by,

$$C_{N/2} = \frac{C_N + C_0}{\delta} \tag{5}$$

where δ is a parameter. Assuming C_0 and C_N fixed, the $C_{N/2}$ value determines the slope of equation (2) at a given point. Therefore, depending on δ, the slope of the quadratic function for a t value is higher or smaller than that of the linear case, which corresponds to δ=2. In the tests shows in section 2.3, we kept the size of the initial chunk as I/2P, which is the optimal value determined with TSS by Tzen and Ni [22]. However, the size of the last chunk and the δ value were optimized for the execution environment.

2.2 Methods based in the slope of the chunks distribution function

Now, the starting point is the slope of the chunks distribution function, C(t). Thus, we are selecting the rate of variation of C(t) as a function of t. Therefore, if the slope is given by a decreasing function, f(t), we have the general expression,

$$\frac{dC(t)}{dt} = f(t) \tag{6}$$

Equation (6) defines a family of differential equations. After integration we will have an explicit functional form for C(t) as a function of t.

A first approach is to consider that the slope (negative) is proportional to the chunk size, equation (7).

$$\frac{dC(t)}{dt} = -kC(t) \tag{7}$$

Here, k is a parameter and t represents the t-th chunk assigned to a processor. Equation (7) can be integrated by separation of variables yielding equation (8),

$$C(t) = \left(\frac{I}{2P}\right) e^{-kt} \tag{8}$$

where we have used C(0)=I/2P, as proposed by Tzen and Ni [22], I is the total number of chunks and P is the total number of processors. Equation (7) defines a new self-scheduling method that we call Exponential Self-Scheduling (ESS). In this method, k is a parameter that will be optimized for our working environment.

A second approach is to consider that the slope (negative) is inversely proportional to C(t)

$$\frac{dC(t)}{dt} = -k/C(t) \tag{9}$$

Here, k is a parameter and t represents the t-th chunk assigned to a processor. Equation (9) can be integrated by separation of variables yielding,

$$C(t) = \sqrt{\left(\frac{I}{2P}\right)^2 - 2kt} \tag{10}$$

where we have used $C(0) = I/2P$, as proposed by Tzen and Ni [22]. Equation (9) defines a new self-scheduling method that we call Root Self-Scheduling (RSS). As in ESS, k is a parameter related to the working environment.

2.3 Comparative Study with other self-scheduling algorithms

In previous works [7–9], we compared at the application level, the performance of QSS, ESS and RSS against Chunk Self-Scheduling, Guided Self-Scheduling, Factoring Self-Scheduling and Trapezoid Self-Scheduling algorithms in an actual Internet-based Grid of computers. The different parameters of our self-scheduling algorithms were determined experimentally, and the parameters of the other self-scheduling algorithms were selected as theirs authors recommend. The tests were carried out in an Internet-based Grid of computers formed by three clusters. Two of them placed in Ciudad Real (Spain) and the other one placed in Puebla (Mexico).

In the tests, we considered sets of tasks without any predefined relationship among their durations. To cover a large range of applications, we used two groups of tests corresponding to very different complexities (payloads). In the first case, the application performed several times a product of square matrices of floating point numbers (payload of $O(n^3)$). Each chunk size corresponded to a number of matrix products (tasks). In the second group, the application sorted several times a vector of floating point numbers. The method used was heapsort (payload

of $O(n \cdot log_2 n)$) [15]. Each chunk size corresponded to a number of vector sorts (tasks).

To carry out the performance study, we considered, for each test group, a total of three test cases involving several thousand tasks. The cases correspond to different combinations of number of allocatable tasks and number of processors. Each case is labeled as: number of tasks/number of processors. With this convention the three test cases are: 2804/20, 5608/20 and 5608/26. The calculations needed for each test were performed three times, to obtain average results. As performance index we have used the speedup relative to the worst case.

Thus, Figure 1 and Figure 2 collect the speedups obtained in the tests performed in the matrix multiplication group and in the heapsort group, respectively.

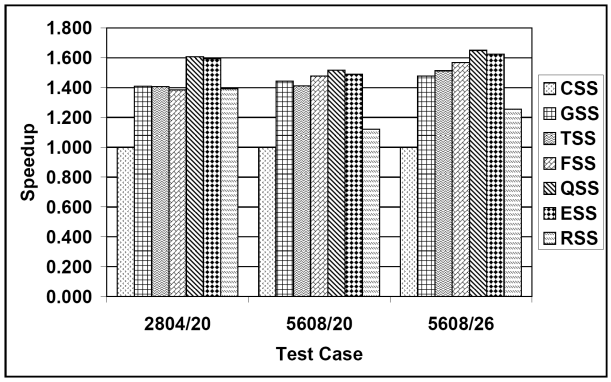

Fig. 1. Matrix multiplication group: Speedup (S), respect to CSS, for the considered self-scheduling algorithms in the tests performed.

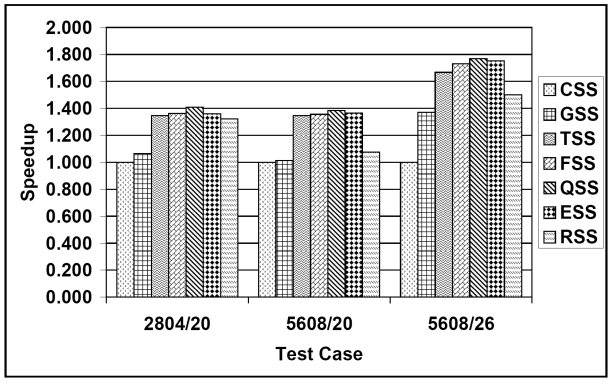

Fig. 2. Heapsort group: Speedup (S), respect to CSS, for the considered self-scheduling algorithms in the tests performed.

The tests collected in Figure 1 and 2 show that QSS outperforms all the other self-scheduling schemes and slightly ESS, because of its higher adaptability to a heterogeneous environment. CSS and RSS present the worst behaviour in the first group of tests (matrix multiplication). This is due to the use of very large chunks. This fact produces a large load imbalance. However, in the second group (heapsort) of tests, CSS and GSS present the worst behaviour. The poor performance of GSS can be due to the large amount of tasks that this algorithm assigns to the first chunks. Finally, FSS and TSS have similar behaviour in the different tests.

3 Handling Dynamic Grid Environments

The previous results show that QSS and ESS outperform the other self-scheduling algorithms tested, since they obtain better load balance and more reduction of the communication overhead. However, a computational Grid is made up of a large number of independent resource providers and consumers, which run concurrently, change dynamically, and interact with each other. Due to these environment characteristics, new approaches such as those based in heuristic algorithms [11, 24] have been proposed to address the challenges of Grid computing. These kinds of algorithms make realistic assumptions based on a priori knowledge of the concerning processes and of the system load characteristics. Braun et al [4] presented three basic heuristics, based on Nature, for Grid scheduling. These are Genetic Algorithms (GA) [1], Simulated Annealing (SA) [12] and Tabu Search (TS) [2].

3.1 Heuristic Approach to Task Scheduling

As presented above, the QSS and ESS self-scheduling algorithms depend on three and two parameters respectively. These parameters determine the behavior of the algorithms. Therefore, for a given computational environment it is necessary to select the most appropriate (optimal) values of these parameters to obtain a good load balance and to minimize the overall computation time. In previous studies, we obtained the best parameters from experimental measures on an actual system [7–9]. However, this is a slow and hard process that we would repeat each time the execution environment changes. In these conditions, the systematic exploration of the parameter space when several (more than two) parameters do exist is simply unmanageable. For this reason in [10], we presented a way to obtain optimal QSS and ESS parameters using a heuristic approach. To such an end, we simulate the execution environment (a computational Grid in our case). So, using the simulation, we obtain the computation time of each algorithm for a given value of its parameters. Then, we apply a heuristic algorithm to explore the behavior of the scheduling method for different values of the parameters, minimizing the overall computation time.

The heuristic algorithm selected was Simulated Annealing (SA). We consider that the function to minimize, the cost function (f), is the overall computation time needed to process a set of tasks, i.e., its makespan. In turn, we consider that the cost function depends on s, the set of parameters used by each self-scheduling algorithm. The cost function f(s) is obtained as the simulated time (in

seconds) necessary to solve all tasks in the specified execution environment. The tasks are scheduled according to QSS or ESS, and the optimal parameters are given by SA. The simulator is organized as follows. Each task has associated a value, from 1 to 10, which represents its duration (in seconds). Task durations are randomly generated. As previously commented, QSS and ESS allocate sets of tasks (chunks). So, the duration of a chunk is the sum of all tasks durations composing it. The computing (CPU) time for a chunk is calculated dividing its duration by the relative computing power of the processor where the chunk is executed. This computing power is referred to the fastest processor. Thus, lower values correspond to slower processors. To this value we add the temporal cost of transferring the chunk to the processor where it is executed. In addition, the scheduling cost introduced by the local queuing software is included as well [10].

3.2 Testing the Heuristic Approach

Several tests were performed in [10] to verify the effect of SA optimized parameters in the efficiency of QSS and ESS. Tests with 1000, 2000, 5000 and 10000 tasks were considered. The execution environment represented in the simulation was a replica of the Internet-Based Grid of computers presented in Section 2.3. The optimal values for the QSS and ESS parameters obtained by SA were used to compare the behavior of QSS and ESS. Moreover, we compared these results against the results obtained using the parameters experimentally determined for QSS and ESS in [8]. Each test was performed 100 times to obtain average results and determine standard deviations. Figures 3 and 4 collect the tests results.

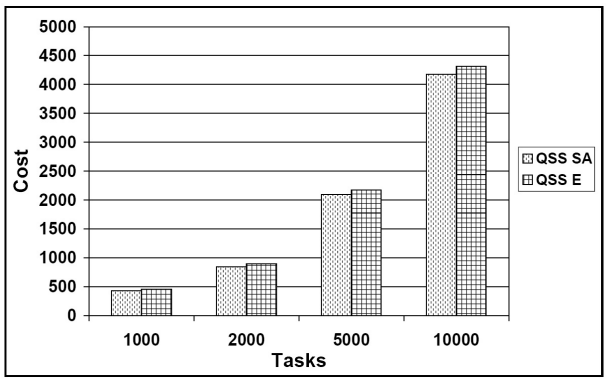

Fig. 3. Comparison between the cost of QSS optimized using SA (QSS SA) and the cost of QSS optimized experimentally (QSS E). The cost is given in seconds.

Figures 3 shows graphically a comparison of the QSS average results. We observe that the cost associated to the SA optimized parameters, "QSS SA", is always lower than that associated to the experimentally optimized QSS, "QSS E". In particular, we find that SA allows for an improvement between 3% and 6%. This is

an interesting result, since SA obtains the most appropriate parameter values in 12 to 60 seconds, whereas the experimental values need a lengthy time consuming procedure on the actual Grid system [8].

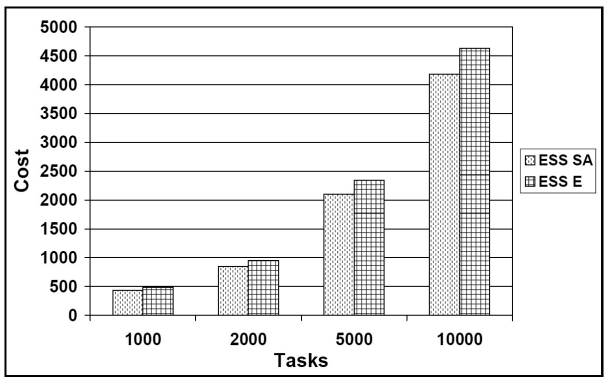

Fig. 4. Comparison between the cost of ESS optimized using SA (ESS SA) and the cost of ESS optimized experimentally (ESS E). The cost is given in seconds.

With respect to ESS, Figure 4 shows graphically a comparison of the ESS average results. We can appreciate that the cost of ESS with SA, "ESS SA", is always lower than that for the experimental parameters, "ESS E". In this case, simulated annealing gives us an improvement between 9% and 12%. Now, only 15 to 45 seconds are needed by SA to obtain the optimal results.

Finally, comparing the QSS and ESS behaviour, we see that both exhibit a similar performance. This result agrees with the data obtained in the experimental tests performed in [8].

4 Conclusions

We have considered in this work the problem of job scheduling at the application level in heterogeneous computing systems. In particular, the work is directed toward Internet-based Grids of computers. Focusing in self-scheduling schemes, we have reviewed recent algorithms proposed by us (QSS and ESS), as well as, their behaviour in an actual system. We have observed that they outperform the other self-scheduling algorithms tested. However, QSS and ESS depend on three and two parameters respectively. Therefore, for a given computational environment it is necessary to select the most appropriate (optimal) values of these parameters to obtain a good load balance and to minimize the overall computation time.

In this sense, we have proposed a way to obtain optimal QSS and ESS parameters using a heuristic approach. We observe that optimizing the parameters using SA permits to reduce the overall computing time up to a 12%. The performed tests show that the simulated performance of the scheduling algorithms is similar

to the experimental observations, with QSS outperforming slightly ESS. However, the time needed to obtain the optimal SA parameters for QSS and ESS in the simulated environment is negligible compared with the time needed for an experimental calibration in an actual Grid system. In addition, SA can optimize all the parameters in the scheduling algorithms, despite its number. In the general case, this is not possible using experimental measures. The test cases also show that the present heuristic approach is very efficient. In fact, we observe a simple linear increase of the execution time with the problem size.

Nevertheless, if the environment characteristics change strongly, we could have a lack of efficiency. Thus, we are working in an adaptive approach to take into account the possible changes in the environment during the execution. The idea is to obtain new optimal QSS and ESS parameters when the environment changes, to maintain a good load balance. Preliminar results show that using the new adaptive approach, we can guarantee a better load balance than using just the heuristic one.

Acknowledgements

This work has been cofinanced by FEDER funds, the Consejería de Educación y Ciencia de la Junta de Comunidades de Castilla-La Mancha (grant # PBI08-0008), and the fellowship associated to the grant # PBI-05-009. The Ministerio de Educación y Ciencia (grant # FIS2005-00293) and the Universidad de Castilla-La Mancha are also acknowledged.

References

1. M. Aggarwal, R. D. Kent and A. Ngom, "Genetic Algorithm Based Scheduler for Computational Grids," in *Proc. 19th Annu. Int. Symp. High Performance Computing Systems and Applications (HPCS'05)*, Guelph, Ontario Canada, 2005, pp.209-215.
2. S. Benedict and V. Vasudevan, "Improving Scheduling of Scientific Workflows Using Tabu Search for Computational Grids," *Information Technology Journal*, vol. 7, no. 1, pp. 91-97, 2008.
3. F. Berman, "High-performance schedulers," in *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Eds. San Fransisco, CA: Morgan-Kaufmann, 1999, pp. 279-309.
4. 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. Par. Dist. Com.*, vol. 61, no. 6, pp. 810-837, 2001.
5. T. L. Casavant and J. G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing," *IEEE Trans. Softw. Eng.*, vol. 14, no. 2, pp. 141-154, 1988.
6. J. Díaz, S. Reyes, A. Niño and C. Muñoz-Caro, "Un Algoritmo Autoplanificador Cuadrático para Clusters Heterogéneos de Computadores," *XVII Jornadas de Paralelismo*, Albacete, Spain, 2006, pp. 379-382.
7. J. Díaz, S. Reyes, A. Niño and C. Muñoz-Caro, "A Quadratic Self-Scheduling Algorithm for Heterogeneous Distributed Computing Systems," *Proc. 5th Int. Workshop Algorithms, Models and Tools for Parallel Computing Heterogeneous Networks* (HeteroPar '06), Barcelona, Spain, 2006, pp. 1-8.

8. J. Díaz, S. Reyes, A. Niño, and C. Muñoz-Caro, "New Self-Scheduling Schemes for Internet-Based Grids of Computers," *1st Iberian Grid Infrastructure Conf.* (IBERGRID), Santiago de Compostela, Spain, 2007, pp. 184-195.
9. J. Díaz, S. Reyes, A. Niño, C. Muñoz-Caro, "Derivation of Self-Scheduling Algorithms for Heterogeneous Distributed Computer Systems: Application to Internet-based Grids of Computers," *Future Generation Computer Systems*, ElSevier, , published online, DOI: 10.1016/j.future.2008.12.003.
10. J. Díaz, S. Reyes, C. Muñoz-Caro, and A. Niño, "A Heuristic Approach to Task Scheduling in Internet-based Grids of Computers," *2nd Int. Conf. on Advanced Engineering Computing and Applications in Sciences (ADVCOMP)*, Valencia, Spain, 2008, pp. 110-116.
11. F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," School of Computing, Queen's University, Kingston, Ontario, 2006.
12. S. Fidanova, "Simulated Annealing for Grid Scheduling Problem," *Proc. IEEE John Vincent Atanasoff 2006 Int. Symp. Modern Computing*, 2006, pp. 41-45.
13. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, San Francisco, CA: Morgan Kauffman Publishers, 1999.
14. S.F. Hummel, E. Schonberg, and L.E. Flynn, Factoring: A Method for Scheduling Parallel Loops, Comm. of the ACM 35 (1992) 90-101.
15. D. Knuth, The Art of Computer Programming, Volume 3: Sorting and Searching, second ed., Addison-Wesley, California, 1997.
16. C.P. Kruskal and A. Weiss, Allocating Independent Subtasks on Parallel Processors, IEEE Trans. Soft. Eng. 11 (1985) 1001-1016.
17. D. J. Lilja, "Exploiting the Parallelism Available in Loops," *IEEE Computer*, vol. 27, vo. 2, pp. 13-26, 1994.
18. S. Penmatsa, A. T. Chronopoulos, N. T. Karonis and B. Toonen, "Implementation of Distributed Loop Scheduling Schemes on the TeraGrid," *Proc. 21st IEEE Int. Parallel and Distrib. Proc. Symp. (IPDPS 2007), 4th High Performance Grid Computing Workshop*, 2007.
19. C.D. Polychronopoulos, and D. Kuck, Guided Self-Scheduling: a Practical Scheduling Scheme for Parallel Supercomputers, IEEE Trans. on Computers 36 (1987) 1425-1439.
20. P. J. Sokolowski, D. Grosu and C. Xu, "Analysis of Performance Behaviors of Grid Connected Clusters," in *Performance Evaluation of Parallel and Distributed Systems*, M. Ould-khaoua, and G. Min, Eds. Hauppauge, NY: Nova Science Publishers, 2006.
21. P. Tang, and P.C. Yew: Processor Self-Scheduling for Multiple Nested Parallel Loops, in: Proc. of the 1986 Int. Conf. on Parallel Processing, 1986, pp. 528-535.
22. T.H. Tzen and L.M. Ni, Trapezoid Self-Scheduling: A Practical Scheduling Scheme for Parallel Compilers, IEEE Trans. Parallel Distrib. Syst. 4 (1993) 87-98.
23. C.-T. Yang, and S.-C. Chang, "A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters," *9th Workshop Compiler Techniques for High-Performance Computing*, Academia Sinica, Taiwan, 2003.
24. J. Yu and R. Buyya, "Workflow Schdeduling Algorithms for Grid Computing," Grid Computing and Distrib. Systems. Lab., Univ. Melbourne, Australia, Tech. Rep., GRIDS-TR-2007-10, May 2007.