KMITL Sci. Tech. J. Vol. 11 No. 1 Jan. - Jun. 2011

Scheduling Independent Tasks on Grid Computing Systems by Differential Evolution

Amid Khatibi Bardsiri[1] and Marjan Kuchaki Rafsanjani[2]*

[1]Bardsir Branch, Islamic Azad UniversityKerman, Iran
[2]*Department of Computer Science,Shahid Bahonar,University of Kerman,Kerman, Iran

Abstract

Grid Computing aims to allow unified access to data, computing power, sensors and other resources through a single virtual laboratory. In this paper, we present Differential Evolution algorithm based on schedulers for efficiently allocating jobs to resources in a grid computing system. Scheduling is a key problem in emergent computational systems, such as Grid and P2P, in order to benefit from the large computing capacity of such systems. The general problem of optimally mapping tasks to machines in a heterogeneous computing suite has been shown to be NP-complete. Experimental results show that our algorithm improves the performance of static instances compared to the results of other algorithms reported in the literature.

Keywords: Grid computing, Heuristics, Differential Evolution, Makespan

1. Introduction

The emerging paradigm of grid computing and the construction of computational grids [1] are making the development of large scale applications possible from optimization and other fields. The development or adaptation of applications for grid environments is being challenged by the need of scheduling a large number of jobs to resources efficiently. Moreover, the computing resources may vary in regard to their objectives, scope and structure as well as to their resource management policies such as access and cost. Grid computing and distributed computing, dealing with large scale and complex computing problems, are one of the hot topics in the computer science and research. Mixed-machine heterogeneous computing (HC) environments utilize a distributed suite of different machines, interconnected with computer network, to perform different computationally intensive applications that have diverse requirements [2]. Miscellaneous resources should be orchestrated to perform a number of tasks in parallel or to solve complex tasks divided to variety of independent subtasks [3]. Indeed, the grid environment is dynamic and, also the number of resources to manage and the number of jobs to be scheduled are usually very large making thus the problem a complex large scale optimization problem. Task scheduling is mapping

*Corresponding author: E-mail: kuchaki@mail.uk.ac.ir

a set of tasks to a set of resources to efficiently exploit the capabilities of such resources. It has been shown, that an optimal mapping of computational tasks to available machines in an HC suite is a NP-complete problem [4] and hence, it is a subject to various heuristic and meta-heuristic algorithms. The heuristics applied to the task scheduling problem include Min-min heuristic, Max-min heuristic, longest job to fastest resource- shortest job to fastest resource (LJFR-SJFR) heuristic, Sufferage heuristic, Work queue heuristic and others [5-7]. Different criteria can be used for evaluating the efficiency of scheduling algorithms; the most important of them is makespan. Makespan is the time when grid computing system finishes the latest task. The objectives of scheduling algorithm are increasing the system throughput measure, reducing task completion time, better resource utilization rate, and balancing the load well [8-10]. In this study, we present the implementation of DE algorithm for job scheduling on computational grids that optimizes the makespan. The heuristic approaches have been tested using the benchmark model of Braun *et al.* [6].

2. Related Work

A set of heuristic algorithms have been designed to schedule meta-tasks to grid computing systems. 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.

2.1 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 [6].

2.2 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.

$$M = \left\{ \min_{1 \le t \le n \ , \ 1 \le j \le m} \left(completion\_time(T_t, M_j) \right) \mid T_t \in U \right\}$$

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].

2.3 Max-min

The Max-min heuristic is very similar to Min-min and its metric is MCT too. It begins with the set *U* of all unmapped tasks. Then, the set of minimum completion times (M) is found as mentioned in previous section. Next, the task with the overall maximum 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 [6, 11].

2.4 LJFR-SJFR

LJFR-SJFR heuristic begins with the set U of all unmapped tasks. Then the set of minimum completion times is found the same as Min-min. Next, the task with the overall minimum completion time from M is considered as the shortest job in the fastest resource (SJFR). Also the task with the overall maximum completion time from M is considered as the longest job in the fastest resource (LJFR). At the beginning, this method assigns the m longest tasks to the m available fastest resources (LJFR). Then this method assigns the shortest task to the fastest resource, and the longest task to the fastest resource alternatively [5, 13].Though the above stated heuristic algorithms have advantages, they do have their own disadvantages. The experimental results from [5-8, 14-16] show that other heuristic (such as OLB, MET, Work Queue, Maxstd and Tabu) do not produce good mappings. In contrast, others are able to deliver good performance and more, only they are intended. Among the stated algorithms, Min-min is the simple and fastest algorithm.

3. Differential Evolution

Differential Evolution (DE) is a reliable, versatile and easy to use stochastic evolutionary optimization algorithm. DE is a population-based optimizer that evolves real encoded vectors representing the solutions to given problem. The DE starts with an initial population of N real-valued vectors. The vectors are initialized with real values either randomly or so that they are evenly spread over the problem domain [17]. The latter initialization usually leads to better results of the optimization process. During the optimization, DE generates new vectors that are perturbations of existing population vectors. The algorithm perturbs vectors with the scaled difference of two randomly selected population vectors and adds the scaled random vector difference to a third selected population vector to produce so called trial vector. The trial vector competes with a member of the current population with the same index. If the trial vector represents a better solution than the population vector, it takes its place in the population. Differential evolution is parameterized by two parameters [18]. Scale factor $F \in (0, 1)$ controls the rate at which the population evolves and the crossover probability $C \in [0, 1]$ determines the ratio of bits that are transferred to the trial vector from its opponent. Figure 1 shows the pseudo code of DE algorithm.

1. Population initialization $X(0) \leftarrow \{x_1(0), \ldots, x_m(0)\}$
2. g ← 0
3. Compute $\{f(x_1(g)), \ldots, f(x_m(g))\}$
4. While the stopping condition is false do
5. For i = 1 to m do
6. $y_i \leftarrow$ generate Mutant$(X(g))$
7. $z_i \leftarrow$ Crossover$(x_i(g), y_i)$
8. If $f(z_i) < f(x_i(g))$ then
9. $x_i(g+1) \leftarrow z_i$
10. Else
11. $x_i(g+1) \leftarrow x_i(g)$
12. End if
13. End for
14. g ← g + 1
15. Compute $\{f(x_1(g)), \ldots, f(x_m(g))\}$
16. End while

Figure 1. Differential Evolution Algorithm.

In the proposed scheduling algorithm, the solution is represented as an array of lengths, which are equal to the number of jobs. The value corresponding to each position *i* in the array represent the resource to which job *i* is allocated. The job-to-resource representation is illustrated in Figure 2.

2	3	1	6	4	5	1	13	…

Figure 2. 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 Differential Evolution, we define a fitness function *fit(S): Sched* → *R* that evaluates each schedule:

$$fit(S) = Makespan(S)$$

We have implemented Differential Evolution for scheduling of independent tasks on heterogeneous independent environments. Storn and Price [17] suggested total ten different working strategies of DE and some guidelines in applying these strategies to any given problem. The Differential Evolution algorithm has implemented as DE/rand/1/exp [19]. The DE algorithm has used with parameters summarized in Table I. The parameters have set after brief initial tuning.

Table 1. A summary of DE parameters

Parameter	Value
Population size	20
Terminating generation	60000/10000
Probability of crossover	0.3
Scaling factor	0.9

4. Problem Definition

The main goal of the task scheduling in grid computing systems is the efficiently allocating tasks to machines. Tasks are originated from different users/applications, and are independent. We use the ETC (Expected Time to Compute) matrix model introduced by Shoukat et al. for formulating the problem [20]. 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 have 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.
- Ready[m] represents the ready time of the machine after completing the previously assigned tasks.
- ETC matrix of size $n \times m$, where n represents the number of tasks and m represents the number of machines.

A meta-task is defined as a collection of independent task (i.e. task doesn't require any communication with other tasks) [21]. Tasks derive mapping statically. For static mapping, the number of tasks, n and the number of machines, m is known 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) [5, 10].

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

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

A flow chart of DE is shown in Figure 3 and r0, r1, r2 and I are distinct indices which are not made explicit in this figure.

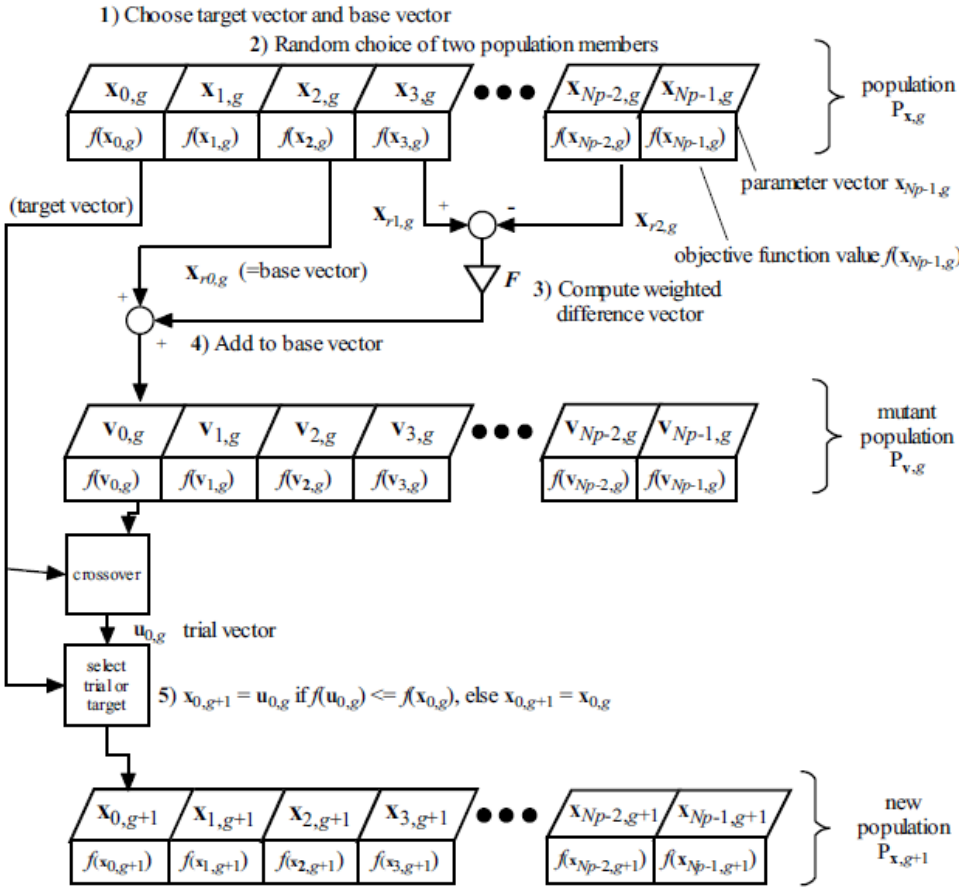

Figure 3. A flow chart of DE's generate-and-test loop

5. Experiments

In this section, after the benchmark description, various heuristic scheduling algorithms were developed to compare with the DE algorithm. The experimental results, discussed below, have obtained by a PC with 2 GHz processor and 2 GB of RAM using Matlab environment. The implemented DE operates on a population of 20 chromosomes (possible mappings) for a given meta-task. Each chromosome is a 512*1 vector. The initial population is generated using two methods: (a) 20 randomly generated chromosomes from a uniform distribution, or (b) one chromosome that is the Min-min solution (i.e., mapping for the meta-task) and 19 random solutions. The last method is called seeding the population with a Min-min chromosome. For the first method, 60000 iterations have been used and the second method was repeated 10000 iterations.

5.1 Benchmark Description

In this paper, we used the benchmark proposed in [6]. 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 (uniform distribution is used for generating the matrices). 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 Experimental results

Among most popular and extensively studying 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 is compared in Tables II. Figure 4 shows the geometric mean of makespan for the 12 considered cases. The relative performance order of the heuristics from best to worst is: 1-DE/Method b, 2-Minmin, 3-DE/Method a, 4-MCT, 5- LJFR-SJFR, and 6-Maxmin. DE provided the best mappings for the 12 cases. In the second method of DE, the initial population is upgraded with vector obtained by Min-min heuristic. The best DE solution always came from one of the populations that had been seeded with the Min-min solution. However, the additional searching capabilities afforded to DE by performing crossover and scaling factor are beneficial. It is important to set DE parameters and feeding initial population.

Table 2. Comparison of obtained makespan values by heuristics

Instance	MCT	Max-min	LJFR-SJFR	Min-min	DE/Method a	DE/Method b
c_hihi	1.13E+07	1.20E+07	1.20E+07	8.38E+06	1.01E+06	8.21E+06
c_hilo	1.58E+05	1.80E+05	1.75E+05	1.33E+05	1.68E+05	1.32E+05
c_lohi	3.83E+05	4.05E+05	4.04E+05	2.81E+05	3.46E+05	2.67E+05
c_lolo	5.30E+03	6.05E+03	5.87E+03	4.50E+03	5.63E+03	4.35E+03
i_hihi	4.28E+06	7.25E+06	6.34E+06	3.58E+06	4.88E+06	3.49E+06
i_hilo	7.39E+04	1.23E+05	1.06E+05	6.65E+04	8.12E+04	6.61E+04
i_lohi	1.45E+05	2.46E+05	2.15E+05	1.21E+05	1.45E+05	1.16E+05
i_lolo	2.47E+03	4.11E+03	3.55E+03	2.26E+03	2.48E+03	2.07E+03
p_hihi	6.23E+06	9.27E+06	8.38E+06	4.86E+06	5.53E+06	4.35E+06
p_hilo	9.82E+04	1.48E+05	1.31E+05	8.43E+04	9.77E+04	7.89E+04
p_lohi	2.10E+05	3.12E+05	2.82E+05	1.64E+05	1.95E+05	1.60E+05
p_lolo	3.32E+03	4.96E+03	4.38E+03	2.83E+03	3.39E+03	2.81E+03

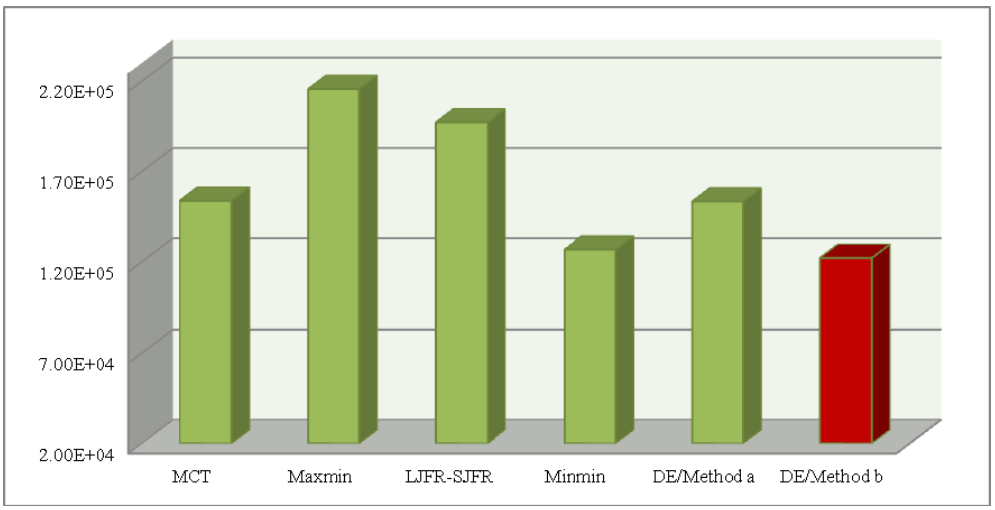

Figure 4. Comparison results between heuristics on makespan

6. Conclusions

Scheduling in grid computing systems is an NP-complete problem. Therefore, using heuristic algorithms is a suitable approach in order to cope with its difficulty in practice. In this paper, we have presented DE algorithm for scheduling in grid computing environments. The goal of the scheduler in this paper is minimizing makespan. The implementation of proposed heuristic scheduling algorithm and various existing algorithms are tested using the benchmark simulation model for distributed heterogeneous computing systems by Braun et al. (2001). The experimental results show that DE algorithm performs better performance than the existing heuristic algorithms in various systems and settings and also it delivers improved makespan. The optimization technique of Differential Evolution (DE) has been used for solving the multi-objective parameters in grid scheduling. Presented algorithm has a number of parameters including C, F and NP. Fine tuning of DE parameters are subject of our future work.

References

[1] Foster, I. and Kesselman C., **1998**. The Grid - Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers.

[2] Tracy, M., Braun, T. D. and Siegel, H., **1998**. High-performance Mixed-machine Heterogeneous Computing. 6th Euro-micro Workshop on Parallel and Distributed Processing, pp. 3-9.

[3] Fernandez-Baca, D., **1989**. Allocating Modules to Processors in a Distributed System. IEEE Transaction, Software Engineering, pp. 1427–1436.

[4] Fidanova, S. and Durchova, M., **2006**. Ant Algorithm for Grid Scheduling Problem, Large Scale Computing. LNCS, *3743*, Springer, pp. 405-412.

[5] Izakian, H., Abraham, A. and Snasel, V., **2009**. Comparison of Heuristics for Scheduling Independent Tasks on Heterogeneous Distributed Environments. Proceedings of the International Joint Conference on Computational Sciences and Optimization, IEEE, *1*, pp. 8-12.

[6] Braun, R., Siegel, H., Beck, N., Boloni, L., Maheswaran, M., Reuther, A., Robertson, J., Theys, M., Yao, M., Hensgen, D. and Freund, R., **2001**. 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)*, pp. 810-837.

[7] Munir, E., Jian-Zhong, L., Sheng-Fei, S. and Rasool, Q., **2007**. Performance Analysis of Task Scheduling Heuristics in Grid. Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC), *6*, pp. 3093-3098.

[8] Baghban, H. and Rahmani, A., **2008**. A Heuristic on Job Scheduling in Grid Computing Environment. Proceedings of the seventh IEEE International Conference on Grid and Cooperative Computing, pp. 141-146.

[9] Zhang, Q. and Zhen, L., **2009**. Design of Grid Resource Management System Based on Divided Min-min Scheduling Algorithm. IEEE First International Workshop on Education Technology and Computer Science, pp. 613-618.

[10] Xhafa, F. and Abraham, A., **2008**. Meta-heuristics for Grid Scheduling Problems, **In:** Meta-heuristics for Scheduling in Distributed Computing Environments. *146*, pp. 1-37, Springer, Germany.

[11] Freund, R. and Siegel, H,. **1993**. Heterogeneous Processing. IEEE Computer, *26(6)*, pp. 13-17.

[12] Freund, R. and Gherrity, M., **1998**. Scheduling Resources in Multi-user Heterogeneous Computing Environment with Smart Net. Proceedings of the 7th IEEE HCW.
[13] Abraham, A., Buyya, R. and Nath, B., **2000**. Nature's Heuristics for Scheduling Jobs on Computational Grids. Proceedings of the International Conference on Advanced Computing and Communications.
[14] Macheswaran, M., Ali, S., Siegel, H., Hensgen, D. and Freund, R., **1999**. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. Journal of Parallel Distributed Computing , *59 (2)*, pp. 107-131.
[15] Xhafa, F., Barolli, L. and Durresi, A., **2007**. Immediate Mode Scheduling in Grid Systems. International Journal of Web and Grid Services, *3(2)*, pp. 219-236.
[16] Munir, E., **2008**. MaxStd: A Task Scheduling Heuristic for Heterogeneous Computing Environment. Information Technology Journal, ISSN-1812-5638.
[17] Price, K., Storn, R. and Lampinen, J., **2005**. Differential Evolution A Practical Approach to Global Optimization. Natural Computing Series, Springer-Verlag, Berlin, Germany.
[18] Price, K. and Storn, R., **1997**. Differential Evolution: Numerical Optimization Made Easy. Dr. Dobb's Journal, pp. 18–24.
[19] Price. K. and Storn, R., **1995**. Differential Evolution: a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical Report, TR-95-012, ICSI.
[20] Ali, S., Siegel, H., Maheswaran, M. and Hensgen, D., **2000**. Modeling Task Execution Time Behavior in Heterogeneous Computing Systems. Tamkang Journal Science and Engineering, pp. 195-207.
[21] Braun, R., Siegel, H., Beck, N., Boloni, L., Maheswaran, M., Reuther, A., Robertson, J., Theys, M. and Yao, M., **1998**. A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-machine Heterogeneous Computing Systems. Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems, pp. 330-335.