2012 International Conference on Industrial and Intelligent Information (ICIII 2012)
IPCSIT vol.31 (2012) © (2012) IACSIT Press, Singapore

Scheduling Scientific Workflows using Imperialist Competitive Algorithm

Farzaneh Fatemipour[1+] , Farnoush Fatemipour[2]

[1]Sama Technical and Vocational Training College, Islamic Azad University, Mashhad Branch, Mashhad, Iran
[2]Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

Abstract. Computational grid is a growing technology in large-scale computing environment, as it provides more flexibility and easier access to distributed computing resources at a lower cost. In grid computing, applications are regarded as workflows. Scheduling complex workflows plays an important role in the performance of grid applications. In this paper we propose a meta-heuristic scheduling method based on a novel evolutionary algorithm called Imperialist Competitive Algorithm (ICA). ICA enhances the global search capability and it balances the exploration and exploitation abilities of the scheduling algorithm. Many of existing workflow scheduling algorithms can only tackle the problems with a single QoS parameter. Proposed scheduling algorithm deals with two important problems: cost optimization under deadline constraint, and execution time optimization under budget constraint. Experiment results illustrate the algorithm performance and feasibility for scheduling workflow applications and solving constrained satisfaction problems. Furthermore, solved examples represent that ICA can be effectively used to solve optimization problems with discrete variables.

Keywords: Imperialist Competitive Algorithm, Workflow Scheduling, Grid Computing, Evolutionary Algorithm, Meta-Heuristic.

1. Introduction

Grid computing is a form of distributed computing whereby resources of many computers in a network are used at the same time to solve a single problem. Grid systems can be thought of as distributed and large-scale cluster computing, while they tend to be more loosely coupled, heterogeneous, and geographically dispersed compared to conventional cluster computing systems.

Grid computing is making big contributions to scientific research, helping scientists around the world to analyze and store massive amounts of data by sharing computing resources. Potential fields of scientific Grid and Cloud applications are climate and environment, genome research, energy research and nanocomputing. Grid Computing enables users to access and deploy applications from anywhere in the world on demand at competitive costs depending on users Quality of Service (QoS) requirements [1]

To exploit the non-trivial power of Grid resources, effective task scheduling approaches are necessary. The interaction between different services provided by computational grid requires resource management and scheduling solutions which allow the coordinated usage of the services.

When processing a computing application in grids, we often regard the application as a workflow. Workflows constitute a common model for describing a wide range of applications in distributed systems. Usually, a workflow is described as a Directed Acyclic Graph (DAG) in which each computational task is represented by a node, and each data or control dependency between tasks is represented by a directed edge between the corresponding nodes [2].

Workflow scheduling is one of the key issues in the management of workflow execution. Scheduling is a process that maps and manages execution of inter-dependent tasks on distributed resources. It allocates

[+] Corresponding author. Tel.: +98-915-3118433; fax: +98-511-5593216.
E-mail address: f.fatemipour@mshdiau.ac.ir

suitable resources to workflow tasks so that the execution can be completed to satisfy objective functions specified by users. Proper scheduling can have significant impact on the performance of system [3].

In this paper we propose a new workflow scheduling approach based on a newly developed evolutionary algorithm called Imperialist Competitive Algorithm (ICA) [4-6] to solve performance optimization problem in scientific workflows.

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 gives an introduction to the Imperialist Competitive Algorithm (ICA). In Section 4, the proposed algorithm is described. Experimental evaluation is represented in Section 5 and the advantages of ICA are discussed. At the end in Section 6 the conclusions are presented.

2. Related Works

In general, the problem of mapping tasks on distributed services belongs to a class of problems known as NP-hard problems [7]. For such problems, no known algorithms are able to generate the optimal solution within polynomial time. Solutions based on exhaustive search are impractical as the overhead of generating schedules is very high. In Grid environments, scheduling decisions must be made in the shortest time possible, because there are many users competing for resources, and time slots desired by one user could be taken up by another user at any moment [3].

Due to the importance of workflow applications, many Grid projects such as Pegasus [8], ASKALON [9], and GrADS [10] have designed workflow management systems to define, manage, and execute workflows on the Grid. Meta-heuristic methods are also used to tackle this problem. Although these methods have a good performance, they usually are more time consuming.

In Quan et al. [11], the performance of six different algorithms, i.e., Tabu Search, Simulated Annealing, Iterated Local Search, Guided Local Search, Genetic Algorithm and Estimation of Distribution Algorithm are compared with each other in solving the problem of cost optimization under deadline constraint.

Chen et al. [12] proposed an Ant Colony Optimization (ACO) algorithm, which considers three QoS parameters: time, cost and reliability. The algorithm optimizes one of the parameters under constraints defined for the other ones.

Pandey et al. [13] used Particle Swarm Optimization (PSO) for workflow scheduling. This algorithm takes into account only the cost parameter and does not include the time parameter in its optimization process.

Yu et al. [14] used Genetic Algorithm to solve workflow scheduling problem considering two QoS parameters: time and cost. This algorithm determines a constraint for one of the parameters and tries to optimize the other one.

Among all intelligent algorithms, the Imperialist Competitive Algorithm (ICA), proposed by [4], leads to better results in optimization. Algorithms such as GA, ACO and PSO and their combinations have been repeatedly used in optimization problems. In contrast, the potentiality of the young ICA has yet to be studied in its entirety. In this paper we use ICA for scheduling workflows considering two important QoS parameters: time and cost. We test the ability of ICA to solve constraint based problems and to deal with discrete values.

3. Imperialist Competitive Algorithm

Imperialist Competitive Algorithm (ICA) [4] is a new evolutionary algorithm which is based on the human's socio-political evolution. This section provides a brief description of ICA and its operators.

The algorithm starts with an initial random population called countries. In an N_{var} dimensional optimization problem, a country is a $1 \times N_{var}$ array. This array is defined as following

$$\text{country} = [P_1, P_2, P_3, P_4, \ldots, P_{Nvar}] . \qquad (1)$$

where P_is are the variables to be optimized. Each country's cost is found by evaluating the cost function f at variables $(P_1, P_2, P_3, P_4, \ldots, P_{Nvar})$. So we have

$$\text{cost} = f(\text{country}) = f(P_1, P_2, P_3, P_4, \ldots, P_{Nvar}) . \qquad (2)$$

The algorithm starts with N_{pop} initial countries. The N_{imp} of the most powerful countries are chosen to form the empires. The remaining N_{col} of the countries will be the colonies each of which belongs to an empire. The initial distribution of the colonies between imperialists is based on the imperialists' power. For this purpose, the normalized cost of an imperialist is defined as

$$C_n = c_n - \max\{c_i\} \ . \tag{3}$$

where c_n is the cost of nth imperialist and C_n is its normalized cost. Having the normalized cost, the normalized power of each imperialist is calculated by

$$P_n = \left| \frac{C_n}{\sum_{i=1}^{N_{imp}} C_i} \right| \tag{4}$$

From another point of view, P_n is the portion of colonies that should be possessed by the imperialist. Then, the initial number of colonies of an empire will be

$$N.C_n = \mathrm{round}\ \{ P_n.N_{col} \} \ . \tag{5}$$

where $N.C_n$ is the initial number of colonies of nth empire. To distribute the colonies, $N.C_n$ of colonies are randomly chosen and given to *imperialist*$_n$.

The total power of each imperialist is determined by the empire power plus percents of its colonies average power.

$$T.C_n = \mathrm{Cost}(\mathrm{imperialist}_n) + \xi \ \mathrm{mean}\{\mathrm{Cost}(\mathrm{colonies\ of\ empires}_n)\} \ . \tag{6}$$

where $T.C_n$ is the total cost of the nth empire and ξ is a positive small number. A little value for ξ causes the total power of the empire to be determined by just the imperialist and increasing it will increase the role of the colonies in determining the total power of an empire. The value of 0.1 for ξ has shown good results in most of the implementations.

3.1. Assimilation

The imperialist countries absorb the colonies towards themselves using the absorption policy shown in Fig. 1. This policy causes the countries move towards their minimum optima. In Fig. 1 the new position of a colony is shown in a darker colour. The direction of the movement is a vector from the colony to the imperialist. In Fig. 1, x is a random variable with uniform (or any proper) distribution. Then

$$x \sim U(0, \beta\times d) \ . \tag{7}$$

where β is greater than 1 and is near to 2.

To increase the ability of searching more area around the imperialist, a random amount of deviation (θ) is added to the direction of movement. Parameter θ has a uniform (or any proper) distribution. Then

$$\theta \sim U(\gamma, -\gamma) \ . \tag{8}$$

where γ is a parameter that adjusts the deviation from the original direction.

3.2. Revolution

In each iteration some of the weakest colonies are selected to be replaced with new ones, randomly. The replacement rate is named as the revolution rate.

The revolution increases the exploration of the algorithm and prevents the early convergence of countries to local minimums. The revolution rate in the algorithm indicates the percentage of countries in each colony which will randomly change their positions.

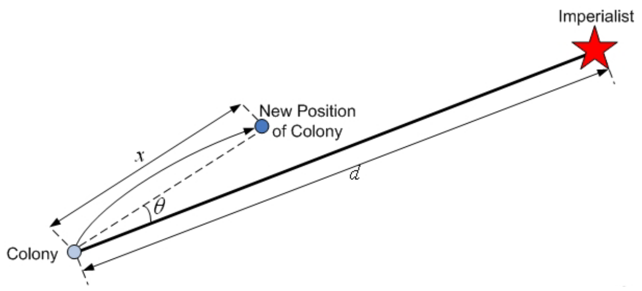


Fig. 1: Moving colonies toward their relevant imperialist in a randomly deviated direction [4]

While moving toward the imperialist or during the revolution process, a colony might reach to a position with lower cost than the imperialist. In this case, the imperialist and the colony change their positions. Then the algorithm will continue by the imperialist in the new position and the colonies will be assimilated by the imperialist in its new position.

3.3. Imperialistic Competition

In this algorithm, the imperialistic competition has an important role. During the imperialistic competition, weak empires will lose their power and their colonies. To start the competition, first a colony of the weakest empire is chosen and then the possession probability of each empire is calculated. The possession probability P_P is proportionate to the total power of the empire. The normalized total cost of an empire is simply obtained by

$$\mathrm{N.T.C}_n = \mathrm{T.C}_n - \max\{\mathrm{T.C}_i\} \ . \tag{9}$$

where $T.C_n$ and $N.T.C_n$ are total cost and normalized total cost of nth empire, respectively. Having the normalized total cost, the possession probability of each empire is given by

$$\mathrm{P}_{\mathrm{p}_\mathrm{n}} = \left| \frac{\mathrm{N.T.C_n}}{\sum_{\mathrm{i=1}}^{\mathrm{N_{imp}}} \mathrm{N.T.C_i}} \right| \tag{10}$$

After a while all the empires except the most powerful one will collapse and all the colonies will be under the control of this unique empire. In such a condition the imperialist competition ends and the algorithm stops.

From a specific point of view, ICA can be thought of as the social counterpart of genetic algorithms. ICA is a mathematical model and a computer simulation of human social evolution, while GA is based on the biological evolution of species. Considering this, we will compare the ICA and GA scheduling results in section 5.

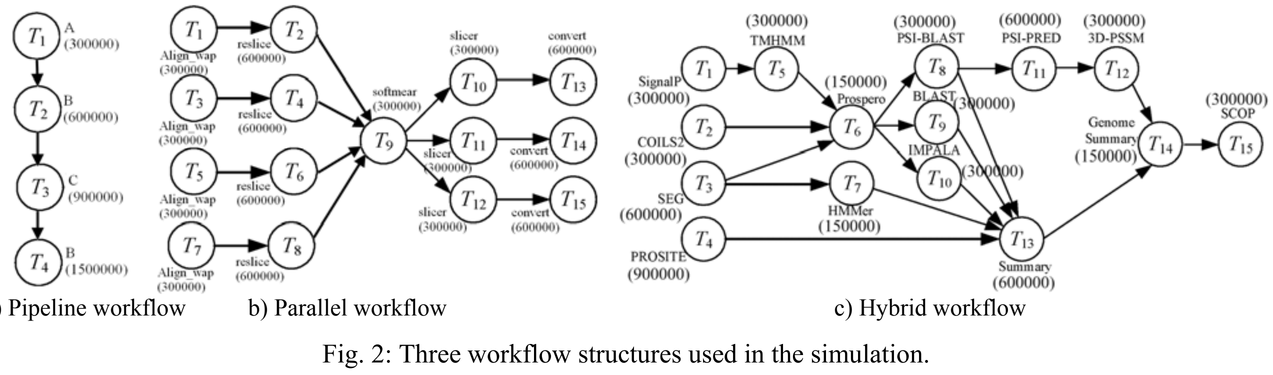
4. Proposed Algorithm

Most of scheduling algorithms minimize only a single QoS parameter and neglect all the others. But typically, service providers charge higher prices for higher quality of service they offer. So it is reasonable to enable users to specify which parameter they prefer to minimize. Sometimes they may prefer to use cheaper services with a lower QoS that is sufficient to meet their requirements.

In this study we address both execution time and execution cost of a workflow. In the proposed model a user can specify a budget or deadline constraint and optimize the other QoS parameter. In deadline constraint mode, total execution time of workflow must not be larger than a user-defined deadline. In this case user prefers to optimize the execution cost of the application. On the other hand in budget constraint mode, the total cost of the workflow must not be larger than the user-defined budget. In this case the user prefers to optimize the execution time of the application. So the objective of the proposed algorithm is to find a schedule that satisfies the user-defined QoS constraint as well as optimizes the user preferred QoS parameter.

A workflow is modelled as a DAG which is implemented as a 2D matrix named *WF*. Fig. 2 represents three sample workflows used in the experiments. The set of nodes $V=\{T_1,T_2,...,T_n\}$ corresponds to the tasks of the workflow. A relationship between T_i and T_j can be represented by $WF[i,j]$.

The set of services available is represented as $S=\{S_1,S_2,...,S_m\}$. Each task in the workflow requires a



a) Pipeline workflow b) Parallel workflow c) Hybrid workflow

Fig. 2: Three workflow structures used in the simulation.

certain type of service. For example, as it is shown in Fig. 2(c) task T_1 requires service type *SignalP* while task T_6 requires *Prospero.* Each service type is supported by different service providers with varied processing capability delivered at different prices.

A schedule is implemented as an array as depicted in Fig. 3. Each task is represented by an array element and the number in each element represents the service assigned to the task.

4.1. Fitness Function

We use the fitness function defined in [14] to measure the quality of the schedules produced. In [14] two fitness functions are defined: *cost-fitness* and *time-fitness.* The cost fitness function of an individual (*I*) is defined by

$$F_{cost}(I) = \frac{c(I)}{B^{\alpha} \times maxCost^{(1-\alpha)}} \tag{11}$$

where *c(I)* is sum of the task execution cost and data transmission cost of *I*, *maxCost* is the most expensive solution of the current population, and *B* is the budget of the workflow. For budget constraint scheduling, the cost-fitness function encourages the formation of solutions that satisfy the budget constraint and for deadline constrained scheduling, it encourages the algorithm to choose solutions with less cost.

The time fitness function of an individual (*I*) is defined by

$$F_{time}(I) = \frac{t(I)}{D^{\beta} \times maxTime^{(1-\beta)}} \tag{12}$$

where *t(I)* is the completion time of *I*, *maxTime* is the largest completion time of the current population, and *D* is the deadline of the workflow. For the budget constrained scheduling, the time-fitness formula is designed to encourage the genetic algorithm to choose solutions with earliest completion time from the current population and for deadline constrained scheduling, it encourages the formation of solutions that satisfy the deadline constraint.

The final fitness function combines two parts and it is expressed as

$$F(I) = \begin{cases} \alpha \times F_{cost}(I) + \beta \times F_{time}(I), & \text{if } F_{cost}(I) > 1 \text{ or } F_{time}(I) > 1 \\ (F_{cost}(I))^{\alpha} \times (F_{time}(I))^{\beta}, & \text{otherwise} \end{cases} \tag{13}$$

4.2. Initial Values

Considering the fact that ICA and GA depend on initial values, the initial countries (ICA) and population (GA) are selected from those schedules that satisfy the constraint specified. This procedure generates better initial values to start both algorithms.

4.3. Discrete Imperialist Competitive Algorithm

The original version of Imperialist competitive algorithm operates on continuous values. However, with a simple modification the Imperialist competitive algorithm can be made to operate on discrete problems.

Assimilation: The assimilation operator is implemented as follows: (1) Two random points are selected from the schedule order of the colony. (2) All services between these two points are replaced by the imperialist's schedule in the same position.

Revolution: It is implemented as follows: (1) A task is randomly selected from the schedule order of the colony. (2) An alternative service is randomly selected to replace the current task allocation.

4.4. Genetic Algorithm Operators

Crossover: This operation is similar to the assimilation operation in ICA: (1) Two parents are chosen at random in the current population. (2) Two random points are selected from the schedule order of the first parent. (3) The locations of all tasks of the crossover points between parent1 and parent2 are exchanged.

Mutation: The probability of mutating a task is called mutation_rate. The mutation operator is similar to the revolution operator in ICA. If a task is selected to be mutated, an alternative service is randomly selected

T_1	T_2	T_3	T_4	T_5	T_6	T_7
2	5	1	4	1	2	6

Fig. 3: Sample schedule. The numbers in the elements represent the services assigned to the tasks.

to replace the current task allocation.

5. Experimental Results

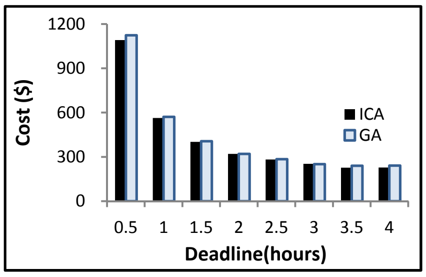
In order to evaluate the proposed approach, we implemented the ICA scheduling algorithm described in previous section and GA heuristic. Fig. 2 depicts three common workflow structures which we use for our experiments: pipeline, parallel and hybrid. A pipeline application executes a number of tasks in a single sequential order. A parallel application executes multiple pipelines in parallel. A hybrid structure application is a combination of both parallel and sequential applications. We select a neuro-science workflow [15] for the parallel application and a protein annotation workflow [16] for the hybrid workflow structure application.

As it is shown in Fig. 2 each task in the experimental workflow applications requires a certain type of service. Each service type is supported by 10 different service providers with different processing capabilities which are represented by Million Instructions per Second (MIPS). The values of MIPS for services range from 100 to 5000. The QoS parameters (execution time and cost) of all service instances follow the rule that for the same task, a service instance with shorter execution time may cost more money and vice versa. Each task has a number of instructions to execute which is indicated in brackets next to the task in Fig. 2. Each task in the workflow has input and output files ranging from 10MB to 1024MB.

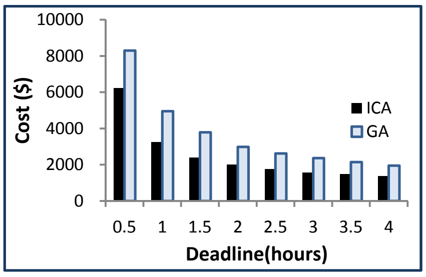
The performance of the proposed scheduling algorithm is tested under two scenarios: Deadline Constraint and Budget Constraint. To make comparison between GA and ICA schedulers, we choose the initial population of 100 for all the experiments. For ICA the initial number of imperialists is 5 and ξ and revolution probability are 0.1. For GA, mutation rate is 0.1 and crossover probability is 0.8.

Fig. 4 and Fig. 5 represent the results for deadline constraint and budget constraint problems respectively. As ICA and GA are random-guided search algorithms, the graphs are plotted by averaging the results obtained after 30 independent executions. Since both algorithms have met the constraint set, the graphs of the constrained parameter are eliminated from the paper.

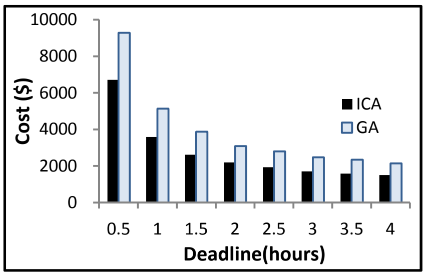
As it is shown in Fig. 4 and Fig. 5, ICA obtained a lower cost than GA in different deadline and budget constraints. In a tight constraint situation that finding a feasible solution to meet the user-defined QoS constraint is very difficult, it can be seen that the difference between ICA and GA results is high. On the contrary, in a loose constraint environment, it is easy to find feasible solutions so the algorithm is able to concentrate only on optimizing the user-preferred QoS parameter. Therefore under loose constraints, as finding feasible solutions is a simpler task, the results of two algorithms become closer. In other words, these results prove the effectiveness of the proposed algorithm especially under rigid constraints.



a) Pipeline application

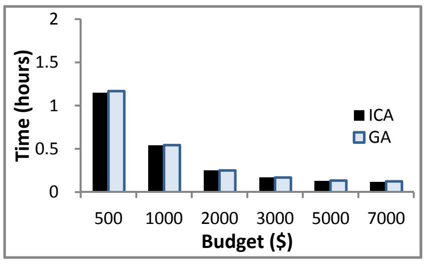


b) Parallel application

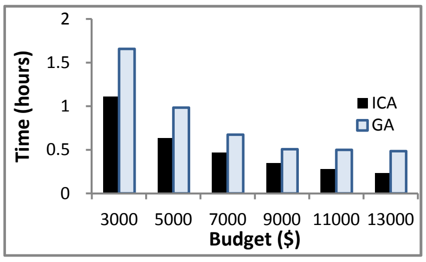


c) Hybrid application

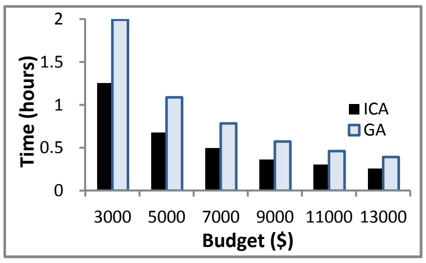
Fig. 4: Comparison of GA and ICA execution cost for different deadline constraints



a) Pipeline application



b) Parallel application



c) Hybrid application

Fig. 5: Comparison of GA and ICA execution time for different budget constraints

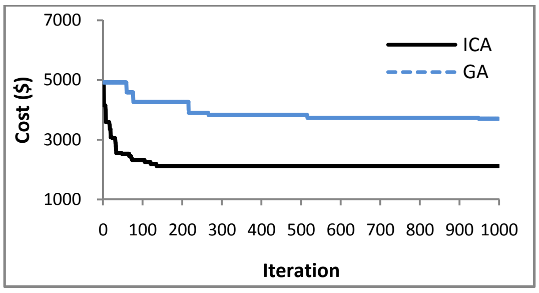


Fig. 6: Convergence speed of ICA and GA for hybrid application, Deadline Constraint = 2 hours

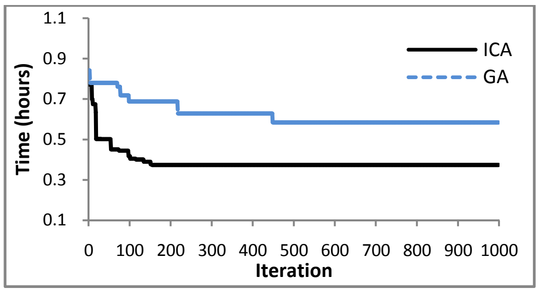


Fig. 7: Convergence speed of ICA and GA for hybrid application, Budget Constraint = $9000

Fig. 6 and Fig. 7 show the convergence rate for two senarios. The budget and deadline constraints for these experiments are chosen in a way that they are not too tight or too loose. ICA finds the feasible solution at iteration 136 for deadline constraint senario and at iteration 155 for budget constraint senario. As the figures depict, ICA manages to find feasible solutions in the very early iterations which is its superiority to GA.

6. Conclusion

In this paper we presented a workflow scheduling algorithm based on Imperialist Competitive Algorithm (ICA). ICA is a novel meta-heuristic search method which uses imperialistic competition process as a source of inspiration. The algorithm is originally designed to work with continuous variables but the experimental results show that it can deal with discrete variables as well as continuous variables.

Compared with most existing scheduling algorithms, the proposed approach targets solving budget and deadline constrained optimization problems. The algorithm allows scheduling tasks onto resources by either minimizing the monetary cost while meeting users' deadline constraint, or minimizing the execution time while meeting users' budget constraint.

The proposed approach is implemented, evaluated and compared with Genetic Algorithm scheduler using three different types of DAGs. The results show that ICA is a robust algorithm for finding the global optimum and has a better convergence rate compared to GA method.

7. References

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, "Cloud computing and emerging IT platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility", Future Generation Computer Systems, vol 25, issue 6, June 2009, pp. 599–616.

[2] S. Abrishami, M. Naghibzadeh, D. Epema, "Cost-driven scheduling of grid workflows using partial critical paths", Delft University of Technology. Parallel and Distributed Systems Technical Report, Aprill 2011.

[3] J. Yu, R. Buyya, K. Ramamohanarao, "Workflow scheduling algorithms for grid computing", Meta-heuristics for Scheduling in Distributed Computing Environments, F. Xhafa and A. Abraham (eds), ISBN: 978-3-540-69260-7, Springer, Berlin, Germany, 2008.

[4] E. Atashpaz-Gargari and C. Lucas, "Imperialist Competitive Algorithm: an algorithm for optimization inspired by imperialistic competition," IEEE Congress on Evolutionary Computation (CEC 2007), pp 4661-4667, 2007.

[5] E. Atashpaz-Gargari, F. Hashemzadeh, R. Rajabioun, C. Lucas, "Colonial competitive algorithm: A novel approach for PID controller design in MIMO distillation column process", International Journal of Intelligent Computing and Cybernetics, No. 3, 1(2008) 337-55.

[6] A. Khabbazi, E. Atashpaz-Gargari, C. Lucas, "Imperialist competitive algorithm for minimum bit error rate beamforming". International Journal of Bio-Inspired Computation. 125–133 (2009).

[7] J. D. Ullman, "NP-complete scheduling problems", Journal of Computer and System Sciences, 10:384-393, 1975.

[8] E. Deelman et al., "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," Sci. Program., vol. 13, pp. 219–237, 2005.

[9] M. Wieczorek, R. Prodan, and T. Fahringer, "Scheduling of scientific workflows in the ASKALON grid environment," SIGMOD Rec., vol. 34, pp. 56–62, 2005.

[10] F. Berman et al., “New grid scheduling and rescheduling methods in the grads project,” Int’l J. Parallel Program., vol. 33, pp. 209–229, 2005.

[11] D. M. Quan and D. F. Hsu, “Mapping heavy communication grid-based workflows onto grid resources within an SLA context using meta-heuristics”, International Journal of High Performance Computing Applications, August 2008 vol. 22 no. 3, pp 330-346.

[12] W. N. Chen and J. Zhang, “An ant colony optimization approach to grid workflow scheduling problem with various QoS requirements,” IEEE Trans. on Systems, Man, and Cybernetics, vol. 39, no. 1, pp. 29–43, 2009.

[13] S. Pandey, L. Wul, S. M. Guru, R. Buyya., "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments", Proc. AINA '10 Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 400-407, 2010.

[14] J. Yu and R. Buyya, “Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms,” Sci. Program., vol. 14, no. 3,4, pp. 217–230, 2006.

[15] Y. Zhao et al., “Grid middleware services for virtual data discovery, composition, and integration”, In 2nd Workshop on Middleware for Grid Computing, October 18, 2004, Toronto, Ontario, Canada.

[16] A. O’Brien, S. Newhouse and J. Darlington, “Mapping of scientific workflow within the e-Protein project to distributed resources”, In UK e-Science All Hands Meeting, Nottingham, UK, Sep. 2004.