A Particle Swarm Optimization Approach for Workflow Scheduling on Cloud Resources Priced by CPU Frequency

Thiago A. L. Genez*, Ilia Pietri†, Rizos Sakellariou†, Luiz F. Bittencourt* and Edmundo R. M. Madeira*
*Institute of Computing, University of Campinas, Campinas, São Paulo, Brazil
†School of Computer Science, University of Manchester, U.K.

***Abstract*—In this paper, we propose a procedure based on Particle Swarm Optimization (PSO) to guide the user in splitting an amount of CPU capacity (sum of frequencies) among a fixed number of resources in order to minimize the execution time (makespan) of the workflow. The proposed procedure was evaluated and compared with a naïve approach, which selects only identical CPU frequency configurations for resources. Simulation results show that, by keeping the overall amount of provisioned CPU frequency constant, the proposed PSO-based approach was able to reduce the makespan of the workflow by carefully selecting different CPU frequencies for resources.**

***Index Terms*—Cloud computing; workflow scheduling; PSO;**

I. Introduction

Recently, users have considered the benefits of using public cloud providers to execute scientific workflows [1], [2]. When processing workflows with data dependencies, jobs can be placed on different resources and the estimated execution time (makespan) of the workflow depends directly on the computational power chosen for each resources. A scheduler is an agent that is responsible for distributing workflow jobs to different resources in order to speed up the execution of the application [3]. To minimize the makespan, the scheduler attempts to select resources based on their processor's speed which is based on the CPU frequency configuration.

In this paper we assume that the user is interested in leasing cloud resources priced by CPU frequency to execute workflows, and he is essentially charged by the provider according to the total of CPU frequency allocated for resources [4]. In line with that, given an amount of CPU capacity (sum of frequencies) to be split among a fixed number of resources, a question that arises is *what is the appropriate CPU frequency configuration of each resource such that the execution time of the workflow is minimal?* A naïve solution for the user is to select identical resources (same CPU frequency) to execute the workflow. This solution, however, may not be the best choice to minimize the makespan of the workflow. It is noted that, by defining a range of CPU frequencies in which the resource can operate while the sum of CPU frequency remains constant, it is possible to find an appropriate CPU frequency for each resource that can make the reduction of the makespan achievable. Dividing an amount of frequency into equal shares among resources is considered a *naïve approach* in this work.

In an attempt to provide an answer to the above question, the main contribution of this paper is to propose a procedure to assist the user to select an appropriate CPU frequency for resources such that the makespan of the workflow is minimized. Therefore, given a number of resources and amount of CPU frequency to be split among those resources, the contribution of the present paper is to show the user that it is possible to reduce the makespan of the workflow by splitting this amount of frequency in different shares in order to run the workflow in resources operating in different speeds.

The proposed procedure employs a method called Particle Swarm Optimization (PSO) [5] in conjunction with the HEFT scheduling algorithm [6]. The proposed procedure aims to assist the user to split an amount of CPU capacity among resources that minimizes the makespan of the workflow. As a result, it is orthogonal to the available budget, and we assume that the user has a budget which is sufficient to pay for this amount of CPU capacity that makes use of the naïve approach (all resources run at the same CPU frequency). Simulation results show that, by keeping the amount of CPU frequency constant, the use of the proposed procedure instead of the naïve approach to split this amount of frequency among resources can reduce the overall makespan by up to 20%.

This paper is organized as follows. Section II presents related work, and Section III describes the background and concepts. The proposed procedure is introduced in Section IV and evaluated in Section V. The results are discussed in Section VI. Section VII concludes the paper.

II. Related Work

Most of the works in the literature evaluate their workflow scheduling algorithms by performing simulations or experiments using traditional model of leasing predefined resources from cloud provider [7], [8], especially from Amazon EC2[1]. Our work, on the other hand, considers cloud providers that provide on-demand resources in a flexible manner, such as the ElasticHosts cloud provider[2], where the user can customize the resource at a fine-grain level by specifying, for example, the CPU frequency that the resource will operate. Actually, few works in the literature consider this type of provider on workflow scheduling.

Pietri et al. described in [4] an algorithm that achieves cost-aware provisioning by selecting different CPU frequencies

[1]http://aws.amazon.com/ec2
[2]http://www.elastichosts.com/

978-0-7695-5697-0/15 $31.00 © 2015 IEEE
DOI 10.1109/UCC.2015.40

for each resource in order to execute workflows within the deadline. The algorithm selects a CPU frequency for each provisioned resource in order to reduce the costs without missing the deadline of the workflow execution. In contrast to [4], we attempt to achieve a CPU frequency for each resource such that the execution time of the workflow is reduced. Another difference is that the workflow scheduling in this paper is limited by a single overall amount of CPU frequency that should be split among a fixed number of resources.

III. Background and Key Concepts

A. Workflow Representation

A workflow application is generally represented by a directed acyclic graph (DAG) in which nodes represent jobs, and edges represent job dependencies. Each job depicts how much computing power a job requires to be executed, and each edge depicts how much data must be transferred between two jobs. We consider a non-cyclic workflow as a representation of a sequence of jobs in a program that solves a scientific problem. A job can only start after all of its parent jobs have finished and all necessary data has been received.

B. Scheduling Workflows in Public Clouds

Cloud computing environments facilitate the execution of workflows by providing dynamically virtualized resources on-demand on a pay-per-use basis, usually for hours of use. The model of resource provisioning by public cloud providers helps users increase or decrease the available computational power according to the needs of the workflow execution as well as the user's budget [9].

C. Cloud Computing Model

In this paper, we assume that an organization has a broker in its premises, which is responsible for leasing resources from public cloud providers and preparing those resources to deploy and execute the applications. This broker receives requests for workflow execution and leases resources based on the schedules produced by the scheduler. We also assume that the resources are on-demand provisioned during the execution of applications and the user is charged according to the total of CPU frequency allocated to the resources, following a linear pricing model employed by the provider [4]. According to the decisions of the proposed procedure, the broker leases the resources by providing the specification of the CPU frequency configuration of each resource. For simplicity, we assume that the other computational characteristics (e.g., disk, memory) are fixed, and we focus only on the CPU configuration in which the frequency value can be adjusted by the broker at the moment that the resource is being leased.

IV. Proposed Procedure based on the Particle Swarm Optimization

A. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a self-adaptive global search based optimization technique that was developed by comparing the social behaviour of animals such as in a bird flock or fish school [5]. The movement of each particle in the swarm is co-ordinated by a velocity, which has both magnitude and direction. The performance of the movement of a particle is measured by a fitness function, which calculates a fitness value according to the current position of the particle. At any moment, the particle that represents the best fitness value, that is, the particle that contains the minimum (or the maximum) fitness value, represents the best candidate solution to the problem. Therefore, the location of the best particle in the search-space represents a region that contains good candidates for the solution of the problem, and the PSO attempts to move the particles of the swarm toward this region. This process is repeated until the stop condition is reached and the PSO returns the best solution achieved so far, if any.

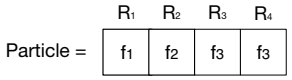

Fig. 1: A sample particle of $m = 4$ dimensions. For each resource i, a CPU frequency value f_i is assigned to it.

PSO is a popular tool due to its simplicity and its effectiveness in solving a wide range of NP-Complete problems at a low computational cost [5], [10]. However, PSO makes few or no assumptions about the problem being optimized, and thus this technique does not guarantee the bounds for the optimality of the achieved solution.

B. Representation of a Particle in the Swarm

The PSO algorithm is similar to other population-based algorithms, such as Genetic algorithms, with the main difference that there is no re-combination (i.e., mutation or crossover) between individuals of the population. In PSO, the population is composed of a number of particles that encompasses the search-space of the problem. Initially, the particles are placed randomly in the search-space, and each one represents a solution for the problem. A particle in the PSO is composed by $m > 0$ dimensions. In this paper, m is the number of resources to be leased by the user, and the value assigned to each particle's dimension represents the value for the CPU frequency configuration to the corresponding resource of this dimension. Figure 1 shows a particle as an array of length m.

C. Fitness Function

In this paper, a workflow scheduler is responsible for measuring the performance of the particle by estimating the makespan of the workflow when using the information of the CPU frequency of resources given by the particle. We employ the HEFT algorithm [6] as a makespan-aware scheduler, but other schedulers for DAGs [11] could be used instead. By using the makespan as fitness value, the PSO is able to manage the particle's position of the entire population at each iteration and achieve a satisfactory solution to the problem.

The provider determines a minimum CPU frequency configuration, f_{min}, allowed to be assigned to a resource. Let $\mathcal{F}$ be the amount of CPU capacity (sum of frequencies) to be split

Algorithm 1 Fitness function

Require: a particle $p[]$; a scheduler s; a workflow $\mathcal{W}$; a set of resources $\mathcal{R}[]$; and a value $\mathcal{F}$ of the sum of frequencies
Ensure: The fitness value of the particle p is equal to the makespan of W in R by using the scheduler s

1: **if** the constraint defined in the equation 1 violated **then**
2: **Return** ∞ as the makespan estimate of the schedule of $\mathcal{W}$ in $\mathcal{R}$
3: **end if**
4: Set the CPU frequency value of the resource R[i] as equal to p[i]
5: Make the scheduling of W in R by using the scheduler s
6: **Return** the makespan of W in $\mathcal{R}$ as the fitness value of the particle

into m resources. We assume that the user has a budget to pay for this amount of CPU capacity. Both $\mathcal{F}$ and m are the input data of the problem addressed in this paper, and we assume that these values are given by the user. A particle represents a *qualified solution* if the following constraint is met:

$$\sum_{i=1}^{m} f_i \leq \mathcal{F}, \quad \text{such that} \quad f_{min} \leq f_i \leq f_{max} \text{ and } f_i \bmod k = 0 \quad (1)$$

where f_{max} is the maximum frequency value that a resource can operate so that the solution represented by the particle is not disqualified. To prevent the production of useless particles (solutions) to the problem, the modulo operation in $f_i \bmod k$ was employed to generate only valid CPU frequency values that were accepted by the provider (by setting a value for k). The f_{max} value is defined as follows:

$$f_{max} = \mathcal{F} - \left[(m-1) \times f_{min}\right] \quad (2)$$

Equations 1 and 2 are used by the PSO when the fitness function is called. Once the particle is created, each dimension receives a number between f_{min} and f_{max}. To calculate the particle's fitness value, we employed the Algorithm 1. Line 1 checks if the constraint represented by equation 1 is not violated. If it is, the solution represented by the particle is not qualified and, in line 2, the makespan is given a value of infinity. Otherwise, it is a qualified solution and, in line 4, the value of each particle dimension, which represents a valid CPU frequency value, will be assigned to the corresponding resource. After that, in line 5, the information about the resources and workflow are used as an input to the scheduler that produces a schedule for the workflow. Finally, in line 6, the makespan of the schedule is returned to the PSO as the fitness value of the particle. As the numbers of particles and iterations to run the PSO are constants, the time complexity of the proposed procedure is $\mathcal{O}(n \times m^2)$, when considering the HEFT to schedule a DAG with n nodes on m resources.

V. Evaluation Methodology

A. Workflows

The proposed procedure is evaluated using simulations with synthetic workflows from the workflow generator gallery[3], which model real-world applications, such as CyberShake, LIGO and SIPHT applications [12]. Workflows with different sizes such as 50, 100, 500 and 1000 nodes are used. For each workflow application and for each workflow size, 20 different workflows were generated differing in the weights of the DAG nodes and DAG edges. Due to space limitation, we present only results for the CyberShake application.

[3]http://pegasus.isi.edu/

TABLE I: Samples of cloud resources.

Input data of the problem	Samples								
	A	B	C	D	E	F	G	H	I
m	2	2	3	3	4	4	5	5	6
$\mathcal{F}$ (GHz)	2.0	4.0	3.0	6.0	4.0	8.0	5.0	10.0	6.0

TABLE II: Parameters used by the fitness function.

Fitness function parameters	Samples								
	A	B	C	D	E	F	G	H	I
f_{min} (GHz)	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
f_{max} (GHz)	1.5	3.5	2.0	5.0	2.5	6.5	3.0	8.0	3.5

B. Cloud Scenario

The resource scenarios to execute the workflow are shown in Table I. They employ resources using as a basis real configurations offered by the ElasticHosts cloud provider. In order to simulate the leasing of resources, we employ 9 samples of resources where the number of resources varies from 2 to 6 and the sum of CPU frequencies of the resources varies from 2 GHz to 10 GHz. Table II shows the f_{min} and f_{max} values of each sample. Important to highlight that the ElasticHosts provider accepts valid frequency values only when $k = 5$. The availability of bandwidth of 1 Gbps was considered in the communication channels that connect the resources inside the provider. We assume in this paper that the resources are exclusively used for the workflow execution.

C. Experimental Parameters

We used the JSwarm-PSO package[4] to conduct the evaluation of the proposed procedure. The number of particles was set to 25 and the number of iterations was set to 200.

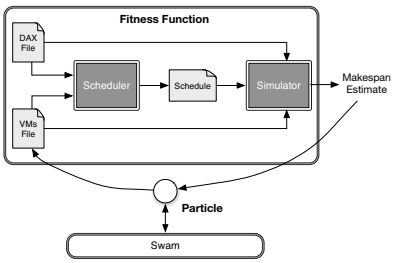

Fig. 2: Overview of the steps of the evaluation.

D. Experimental Simulator

Figure 2 shows an overview of how the evaluations were carried out. The data inputs for the scheduler are a DAX file and a VMs file. The DAX file contains the description of the workflow in XML format, while the VMs file contains information about the resources which are re-configured according to the information of the CPU frequencies represented by the

[4]http://jswarm-pso.sourceforge.net/

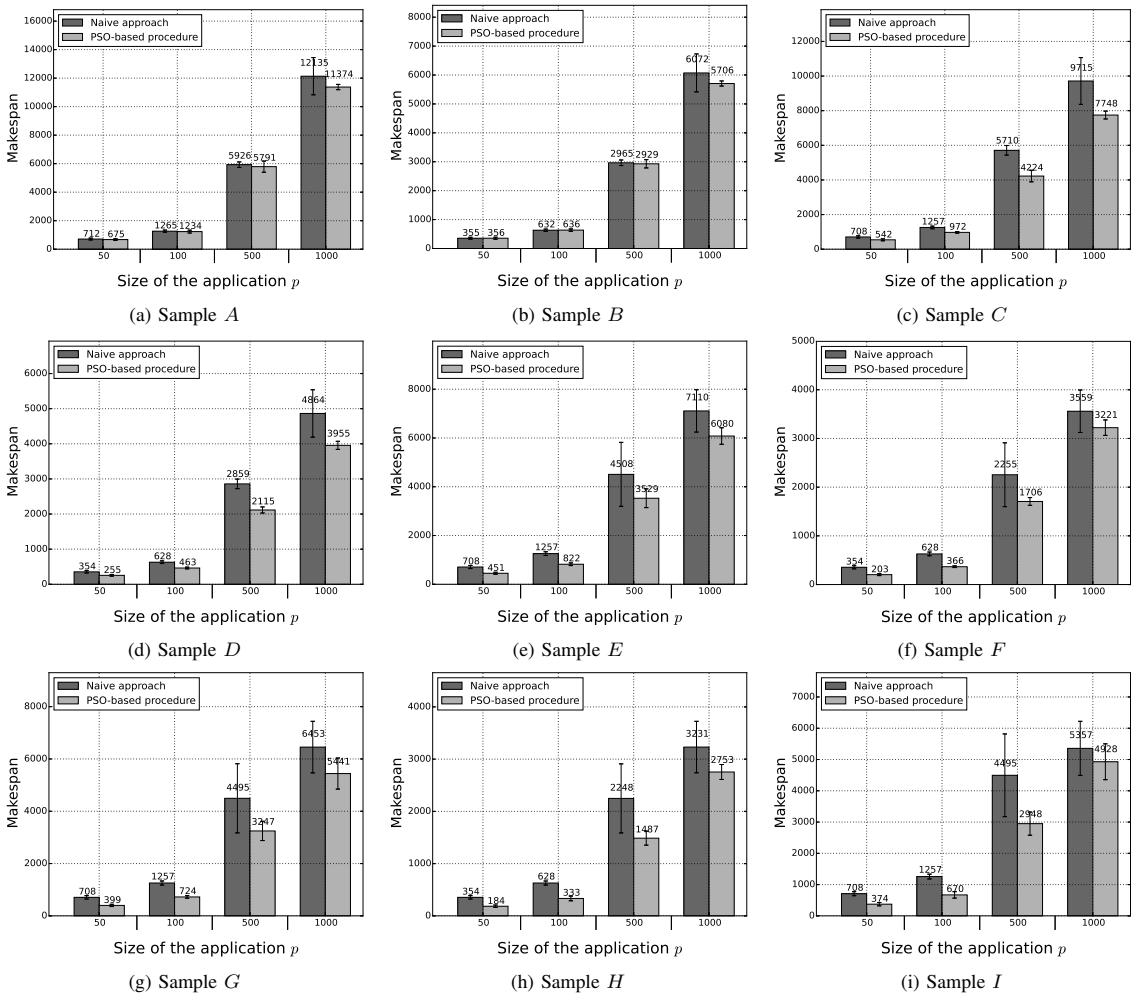

Fig. 3: Results for the CyberShake application

particle. After the schedule is produced by the scheduler, it is used as an input to the simulator, which calculates the values of the makespan of the execution of the workflow.

E. The Naïve Approach

We consider the selection of identical resources as a natural user's attitude, and we call such approach as the *naïve approach*. The use of the naïve approach was employed in this paper with the purpose of comparing the efficiency of the results obtained from the PSO-based proposed procedure.

VI. Results

A. Impact on the Makespan Estimates by using the Proposed PSO-based Procedure

Figure 3 shows the results for CyberShake as a function of the size of the workflow. This application is composed by several computationally intensive jobs with high CPU-utilization [12], and therefore its makespan estimate is greatly affected/influenced by the computational speed of the resources. As a result, keeping the amount of frequency constant, different CPU frequency configurations result in different makespan estimates. For example, by using the PSO-based proposed procedure, the scheduler achieves better results when compared with the naïve approach for all the samples used in the evaluation. By using the sample E, for example, where $m = 4$ and $\mathcal{F} = 8$ GHz, there was a reduction of 37%, 35%, 22%, and 15% in the average makespan in the evaluation of the CyberShake applications with the size of 50, 100, 500 and 1000 jobs, respectively (Figure 3e). Moreover, the average reduction of the makespan achieved by the proposed procedure for all samples was of approximately of 24% when compared with the naïve approach. In other words, with the same computing power, simply changing the configuration of the CPU frequencies of each resource, the average runtime of the CyberShake application was almost one quarter faster by not using identical resources indicated by the naïve approach.

TABLE III: % of average CPU frequency distribution among resources of the evaluations of Figure 3.

Workflow Size	SAMPLES																																	
	A		*B*		*C*			*D*			*E*				*F*				*G*					*H*					*I*					
	0	1	0	1	0	1	2	0	1	2	0	1	2	3	0	1	2	3	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	5
50	47	53	31	69	17	38	45	8	29	63	12	12	34	41	6	7	33	53	10	10	11	32	38	5	6	10	30	48	8	9	9	11	29	34
100	47	53	38	62	17	40	43	9	33	58	12	13	35	40	6	7	32	54	10	10	11	32	38	5	7	9	34	45	8	9	9	10	30	35
500	40	60	31	69	20	33	47	14	31	55	16	18	29	37	7	12	29	52	10	13	16	23	38	6	7	11	22	54	8	9	11	15	25	32
1000	38	62	24	76	22	28	50	15	25	60	15	19	27	39	10	14	26	50	11	13	16	24	36	7	9	17	24	43	9	10	12	16	19	34

Simulations with the SIPHT application show similiar results to the CyberShake because this application is also composed of jobs with high CPU-utilization [12]. On the other hand, the savings in makespan of the LIGO results were lower compared with the CyberShake and SIPHT results because the LIGO application is composed several data-intensive jobs.

The results discussed in this section show that the makespan of CPU-intensive workflows are affected more from the change of CPU frequencies compared with the case of data-intensive ones. Although the reduction of the makespan of data-intensive workflows was not significant, the proposed procedure was able to achieve better results than the naïve approach.

B. Distribution of the Amount of CPU frequency on Resources

Table III presents the percentage of the average distribution of the amount of CPU frequency among the resources of the evaluations of CyberShake application. Basically, for each sample, regardless of the size of the workflow, the proposed procedure was able to split the amount of CPU frequency among m resources of the sample into similar shares. For example, in the sample B, where $m = 2$, the average splitting of 4 GHz was between 24% to 38% for one resource, and 62% to 76% for the other resource. Therefore, by carefully selecting different CPU frequencies for resources, the proposed procedure was able to reduce the makespan of the workflow.

TABLE IV: Average runtime (s) of the evaluations of Figure 3

Workflow Size	SAMPLES								
	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*
50	1	1	2	3	1	1	0	0	0
100	2	2	2	5	2	2	1	0	1
500	46	23	16	12	10	3	7	2	3
1000	90	68	76	20	19	37	10	2	7

C. Average Runtime of the Proposed Procedure

Table IV presents the average runtime of the evaluation of Figure 3. The average runtime of the naïve approach was less than 1 second for all simulations. Due to the simplicity of this approach, the schedule is performed directly by the HEFT scheduler which produces only one solution to the problem. The proposed PSO-based procedure, on the other hand, assesses a population of candidate solutions and its execution time took more than one second to achieve a solution to the problem. By using the sample E, for example, the proposed PSO-based procedure took, on average, 1, 2, 10, and 19 seconds to achieve solutions to the CyberShake application with 50, 100, 500, and 1000 jobs with the average makespan reduced by 36%, 34%, 22%, and 51%, respectively.

Although the average running time of the proposed procedure was greater than 1 second, the majority of executions took less than 30 seconds to achieve better solutions than the solutions appointed by the naïve approach.

VII. Conclusion

In this paper, we propose a procedure based on Particle Swarm Optimization (PSO) to guide the user in splitting an amount of CPU capacity (sum of frequencies) among a fixed number of resources in order to minimize the execution time (makespan) of the workflow. The proposed procedure was evaluated and compared with a naïve approach, which divides the amount of CPU frequency into equal shares among resources. Simulation results show that, by keeping the amount of frequency constant, the proposed PSO-based procedure can reduce the overall makespan by up to 20%. Future work includes investigate CPU configurations for resources that achieve a balance between makespan and costs.

Acknowledgment

This work was carried out while the first author was on a 12-month visit to the University of Manchester, with the financial support offered by the State of São Paulo Research Foundation (FAPESP – 2014/08607-4), which is gratefully acknowledged.

References

[1] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for e-Science: Scientific Workflows for Grids*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[2] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the use of cloud computing for scientific workflows," in *IEEE International Conference on eScience*, Dec 2008, pp. 640–645.

[3] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, 2008.

[4] I. Pietri and R. Sakellariou, "Cost-efficient provisioning of cloud resources priced by CPU frequency," in *7th International Conference on Utility and Cloud Computing*, December 2014.

[5] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *IEEE Int'l Conf. on Neural Networks*, vol. 4, Nov 1995, pp. 1942–1948 vol.4.

[6] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, Mar 2002.

[7] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds," in *24th IEEE/ACM International Conference on Supercomputing*, 2012, pp. 10–16.

[8] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira, "Workflow Scheduling for SaaS/PaaS Cloud Providers Considering Two SLA Levels," in *Network Operations and Management Symposium*, 2012.

[9] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Fut. Generation Computer Systems*, vol. 29, no. 1, pp. 158 – 169, 2013.

[10] S. Pandey, L. Wu, S. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *24th IEEE Int'l Conf. on Advanced Information Networking and Applications*, April 2010, pp. 400–407.

[11] L.-C. Canon, E. Jeannot, R. Sakellariou, and W. Zheng, "Comparative evaluation of the robustness of dag scheduling heuristics," in *Grid Computing*, 2008, pp. 73–84.

[12] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682–692, Mar 2013.