Scheduling Dynamic Parallel Loop Workflow in Cloud Environment

Sucha SMANCHAT[*] and Kanchana VIRIYAPANT

Faculty of Information Technology, King Mongkut's University of Technology North Bangkok, Wongsawang, Bangsue, Bangkok 10800, Thailand

([*]Corresponding author's e-mail: sucha.s@it.kmutnb.ac.th)

Received: 7 March 2016, Revised: 29 July 2016, Accepted: 26 August 2016

Abstract

Scientific workflows have been employed to automate large scale scientific experiments by leveraging computational power provided on-demand by cloud computing platforms. Among these workflows, a parallel loop workflow is used for studying the effects of different input values of a scientific experiment. Because of its independent loop characteristic, a parallel loop workflow can be dynamically executed as parallel workflow instances to accelerate the execution. Such execution negates workflow traversal used in existing works to calculate execution time and cost during scheduling in order to maintain time and cost constraints. In this paper, we propose a novel scheduling technique that is able to handle dynamic parallel loop workflow execution through a new method for evaluating execution progress together with a workflow instance arrival control and a cloud resource adjustment mechanism. The proposed technique, which aims at maintaining a workflow deadline while reducing cost, is tested using 3 existing task scheduling heuristics as its task mapping strategies. The simulation results show that the proposed technique is practical and performs better when the time constraint is more relaxed. It also prefers task scheduling heuristics that allow for a more accurate progress evaluation.

Keywords: Workflow, parallel loop, workflow scheduling, cloud computing

Introduction

Cloud computing [1] has become a prominent platform for scientific computation as it can offer high computation power on demand without full investment in hardware. Cloud computing resources have been utilized in many scientific applications including scientific workflows, which have been used to orchestrate and automate scientific experiments that are composed of multiple steps and involve different computer programs represented as workflow tasks.

Parallel loop workflow is a workflow that is repeated numerous times. This type of workflow has been used in parametric studies (also known as parameter sweep application) to explore the effect of a set of parameters by varying their values in each loop iteration. Parallel loop workflow is distinct from the traditional loop in that each loop iteration is independent of other iterations, making it possible to execute these loops at the same time to speed up the whole execution [2].

In order to dynamically execute a parallel loop workflow in the cloud computing environment, it is necessary to schedule tasks in the workflow so that the cost incurred from cloud resource usage is reasonably controlled and that the execution does not take too long. Although many workflow scheduling techniques have been proposed [3], most of them are designed to schedule a single instance of a workflow. In our context, each loop iteration of a parallel loop workflow is instantiated dynamically at run-time. This dynamism nullifies the traditional evaluation of workflow cost and time based on workflow structure such that existing scheduling techniques cannot be effectively applied. In this paper, we propose a novel technique that can schedule dynamic parallel loop workflow subject to a deadline constraint while reducing the cost of cloud usage. Our contribution consists of 1) a new method to evaluate workflow execution progress without relying on traversing workflow graph, 2) a workflow instantiation control, and 3) a cloud resource adjustment mechanism.

The rest of this paper is organized as follows. The next section reviews notable workflow scheduling techniques that are designed for execution on cloud environment. The third section describes and discusses our approach to scheduling parallel loop workflow. The simulation results are then presented and a conclusion is made along with future research directions.

Related work

Workflow scheduling has long been attracting researchers to tackle different types and characteristics of workflows. Beginning in 2008, many techniques have been proposed to cope with the distinct cloud resource model involving both cost and time aspects [3]. Workflow scheduling techniques can be grouped as batch scheduling and dependency based scheduling. Those of the former type have been used since the Grid and cluster computing era mostly focusing on minimizing workflow makespan. They iteratively schedule tasks that are ready for execution in each scheduling iteration based on certain resource performance metrics such as execution time [4], sufferage value [5] and resource competition [6]. However, these techniques cannot handle multiple scheduling objectives such as cost and time so well because they do not traverse the workflow to estimate the total makespan and cost. Thus, they are rarely applied to cloud workflow where the cost objective is dominant. The latter type of technique schedules workflow by traversing the workflow graph. Most of the existing cloud workflow scheduling techniques are of this group because cost and makespan can be estimated and controlled as the workflow is traversed. A number of approaches are employed including deadline distribution, critical path scheduling and workflow partitioning.

The IC-PCP algorithm [7] recursively schedules the critical path of a workflow backward from the last task. The tasks in the critical path of the current scheduling iteration are assigned to the cheapest resource that can finish them within the deadline. The IC-PCPD2 algorithm [7], another variant of the IC-PCP, distributes the deadline into individual deadline for each task, which is then scheduled to the cheapest resource that can finish it within its own deadline. The HCOC algorithm [8] partitions a workflow into groups of tasks. Each group contains the tasks in the same path, which are then scheduled to the same resource to avoid data transfer. The algorithm also employs rescheduling in the case that the deadline is expected to be violated. The PBTS algorithm [9] employs time based partitioning of a workflow based on the charge period of cloud provider (such as 1 h according to Amazon EC2). It tries to optimize the number of virtual machines in each partition based on a workflow graph and then reuses, launches or terminates virtual machines as necessary to minimize the total cost.

In almost every cloud workflow scheduling technique, it is necessary to evaluate the makespan and the cost of a workflow. This usually relies on traversing the workflow graph. However, with the dynamic parallel loop workflow running as multiple concurrent workflow instances, the workflow graph cannot be traversed because the workflow instances are instantiated at run-time and may not be instantiated all at once. In such cases, the batch scheduling is more appropriate but a new method for evaluating the execution progress must be devised. Also, the control of workflow instantiation should be appropriate to the progress of execution, and additional virtual machines should be started as necessary to cope with the deadline of the workflow.

Materials and methods

In our context of a dynamically executing parallel loop workflow, each independent loop is considered to be an independent workflow instance. Thus, these workflow instances can be concurrently executed to accelerate the entire execution [2]. As mentioned earlier, the necessary mechanisms to handle such dynamism are the evaluation of workflow execution progress, the control of workflow instance arrival (i.e. workflow instantiation), and the adjustment of cloud resources.

Problem definition

A parallel loop workflow is defined as a directed acyclic graph $W = (V, E, L)$, where V is a set of vertices representing tasks in the workflow and E is a set of edges representing precedence dependencies between tasks. For a task to be ready for execution, all of its preceding tasks must have completed their execution. It is assumed that the number of loops L is known to the scheduler. Due to our workflow execution context, we distinguish a specific loop using the term "workflow instance" (i.e. each loop is executed as a separate workflow instance). Thus, the specific tasks in each workflow instance are referred to as "task instances".

A cloud provider P offers an unlimited number of virtual machine instances with different types. Each type is associated with a processing power denoted as Resource Capability Unit (RCU) and a cost per time unit. Each virtual machine instance can execute at most one task instance at a time. It is assumed that at least one virtual machine instance from P is initially running to host the input files for the workflow. We also assume that the workflow execution takes place within a single cloud provider region. This assumption, according to most commercial cloud providers, infers that the time and the cost for data transfer are negligible.

Thus, given a parallel loop workflow W and a cloud provider P, our scheduling objective is to find the mapping between the task instances in W to the virtual machine instances that can be requested from P so that the overall makespan is within a specified deadline D while reducing execution cost.

Evaluating execution progress

To determine the progress of a workflow execution without traversing the workflow graph, we instead compare the number of completed task instances against the total number of task instances. The latter can be determined by simply multiplying the number of parallel loops L with the number of tasks $|V|$ in the workflow. The progress ratio PR of the workflow is thus defined as follows;

$$PR = \frac{number\ of\ completed\ task\ instances}{L \times |V|} \tag{1}$$

To establish a time-based benchmark, we define an elapse time ratio ER by comparing the elapsed time since the start of the workflow execution against the deadline D as follows;

$$ER = \frac{elapsed\ time}{D} \tag{2}$$

In order to determine whether the current progress would satisfy the deadline of the workflow, the progress evaluation PE is defined as the ratio between PR and ER as shown in Eq. (3);

$$PE = \frac{PR}{ER} \tag{3}$$

The PE can indicate whether the workflow execution is currently progressing at the proper pace. If the value of PE is less than 1, then the progress is estimated to be too slow to meet the deadline and actions may be taken. On the other hand, a value of 1 or higher indicates that the current progress is satisfactory to meet the deadline. This ratio is necessary in our technique for not only progress evaluation but also for instance arrival control and resource adjustment.

Workflow instantiation control

Workflow instantiation control is necessary when a parallel loop workflow is dynamically executed. The instantiation of each parallel loop should be controlled so that the scheduling process is not flooded with too many task instances in one scheduling iteration. Otherwise, these task instances would be scheduled to a few running virtual machine resources before additional ones are launched causing an overall delay. Nevertheless, the control should also consider the current progress of the execution and generate more workflow instances if the progress is estimated to violate the deadline.

After instantiating the first workflow instance at the start of the execution, it is not possible to evaluate the progress because no task instance has been completed (i.e. the number of completed tasks is zero and thus *PE* is zero), thus the workflow instantiation is suspended. After one task instance is completed, the instantiation control considers 2 conditions. The first condition is that the number of unscheduled task instances in each scheduling iteration is at least 6 times the number of idle virtual machines. This *threshold* guarantees that there is always a sufficient number of task instances to be scheduled as proven in [6]. The second condition depends on the value of *PE*. A workflow instance is generated if *PE* is less than 1 and that the progress of the execution has changed (i.e. a task instance is completed) since the latest workflow instance has been generated. The latter condition is for avoiding indefinitely instantiating workflow without re-evaluating the progress.

Cloud resource adjustment

Depending on the progress evaluation, additional virtual machine instances may be launched to speed up the execution to meet the deadline. Similar to the instance arrival control, cloud resource adjustment is suspended at the start of execution because it is not possible to evaluate the progress; it is resumed after the first task instance is completed.

The first step in the resource adjustment is to check whether an idle virtual machine instance is available and whether the value of *PE* is more than or equal to one. If any of these conditions is true, then no additional virtual machine instance is launched. In addition to reusing existing virtual machines, this is to avoid launching virtual machine instances unnecessarily, which would increase the cost.

If the value of *PE* is less than 1 and the progress of the execution has changed since the latest resource adjustment, then one virtual machine instance will be launched in each scheduling round. This is also to avoid launching too many virtual machine instances unnecessarily without re-evaluating the progress. In our early test, allowing several virtual machine instances to be launched in one scheduling round causes an excess number of virtual machine instances resulting in excessively high cost. The virtual machine type to be launched is decided by first determining the average resource capability unit RCU_{avg}, which is calculated by the total RCU contributed by running virtual machine instances averaged over time as follows;

$$RCU_{avg} = \frac{\sum_{i=1}^{|VM|}\left(RCU_i \times uptime_i\right)}{elapsed\ time} \tag{4}$$

In Eq. (4), *VM* is the set of running virtual machine instances, RCU_i is the resource capability unit of each virtual machine instance in *VM* according to its type, and $uptime_i$ is the duration since each virtual machine instance has started. To determine additional RCU_{req} that is required to meet the deadline, we use the following equation employing *PE*;

$$RCU_{req} = RCU_{avg} \times \left(\frac{1}{PE} - 1\right) \tag{5}$$

Eq. (5) estimates the required RCU_{req} based on the proportion between the progress and the elapsed time subject to the deadline. The adjustment then selects the virtual machine type with the RCU immediately higher than the RCU_{req}. For example, if RCU_{req} is 10 and there are virtual machine types with 12 and 16 RCU, then the one with 12 RCU is selected. The workflow instance control and the cloud resource adjustment work in harmony. By employing *PE*, as workflow instances are generated, virtual machine instances can be launched to handle them.

Scheduling technique

The 3 mechanisms, which are our main contribution, described so far are developed to cope with the task instances and virtual machine resources. Here, we combine them into the proposed technique to schedule parallel execution of the dynamic parallel loop workflow. The whole body of the proposed scheduling technique is shown in **Table 1**, separated into 3 phases. The outermost *while* loop is the *scheduling iteration* control that repeats until the entire execution is completed. The workflow instantiation control is implemented from lines 2 to 10; the task instances instantiated according to the conditions in this phase are added to the set of unscheduled task instances *T*. Then, the task instances in *T* that are ready for execution according to their precedence dependencies are added to the set of ready task instances *RT*. The cloud resource adjustment in lines 12 - 15 is then performed given that the conditions specified in the previous section are met.

A task mapping heuristic is then invoked on the set *RT* (line 16) to assign the task instances within to the virtual machine instances. In this research, we employ the Min-Min [4], the Max-Min [4] and the XSufferage [5] task mapping heuristics for the purpose of testing the scheduling technique because of their simplicity and decent performance [6]. The main reasoning behind the Min-Min heuristic is that it iteratively schedules the shortest task among those being ready to the resource that will complete the task at the earliest. On the other hand, the Max-Min heuristic iteratively schedules the longest task first [4]. The XSufferage heuristic, disregarding the size of tasks, iteratively schedules the task that would suffer the longest delay if not executed first to the resource that will complete it at the earliest.

Table 1 The proposed scheduling technique.

Phase	**Input:** Parallel loop workflow, total number of parallel loop *L*, Cloud provider *P* **Output:** Schedule for executing parallel instances of parallel loop workflow
Scheduling Iteration	1 **while** (($L > 0$) || set of unscheduled task instances *T* is not empty) **do**
Workflow instantiation control	2 Calculate progress evaluation *PE* using Eq.(3)
	3 **if** firstSchedulingIteration **then**
	4 Instantiate the first workflow instance for the first parallel loop, and decrement *L*
	5 **else if** (completed task > 0) **then**
	6 **while** (size of $T <$ threshold) || (($PE < 1.0$) && (progress changes)) **do**
	7 Generate one workflow instance, and decrement *L*
	8 Add task instances from the new workflow instance to set *T*
	9 **end while**
	10 **end if**
	11 Determine the task instances in set *T* that are ready and add them to set *RT*
Resource adjustment	12 **if** (($PE < 1.0$) && (!freeNode) && (completed task > 0) && (progress changes)) **then**
	13 Calculate required RCU_{req} using Eq.(5)
	14 Launch a new VM instance in *P* with RCU immediate higher than RCU_{req}
	15 **end if**
Task mapping	16 Invoke a task mapping heuristic on the set of ready task instances *RT*
	17 $T = T - RT$
Scheduling iteration	18 Wait for the next scheduling iteration
	19 **end while**

Once scheduled, the task instances in *RT* are removed from the set *T* in line 17. The scheduler then waits for the next scheduling iteration, which is triggered either by a completion of a task instance or the conditions for workflow instantiation control being met. This process is repeated by the outermost while loop until the entire execution is completed.

Results and discussion

Workflow simulation is employed in the evaluation. We implement our technique and the 3 task mapping heuristics, namely the Min-Min, the Max-Min and the XSufferage, as the workflow schedulers along with a simulation environment in the Nimrod/K system [2,10]. This system supports parallel execution of parameter sweep workflow, whose execution is in parallel loops, by dynamically running multiple instances of the workflow in parallel.

Because there is no existing technique, to the best of our knowledge, that can schedule multiple parallel cloud workflow instances and estimate the workflow execution cost and time without relying on traversing the workflow structure [3], our evaluation aims to study the behavior of our proposed technique based on the 3 existing task mapping heuristics. We use a workflow scenario as shown in Figure 2 in the experiment. The workflow is composed of 5 tasks arranged in 2 parallel sequences merging at the last task; such structure is analogous to those commonly found in large scientific workflows. The experiment is set to run 100 parallel loops, totaling 500 task instances. As mentioned in our problem definition, the execution takes place within a single cloud provider region thus the data transfer time and cost are negligible.

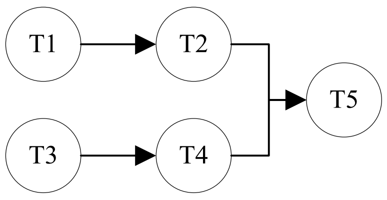

Figure 2 The workflow scenario.

Three general purpose instance types are adopted from the Amazon EC2[1] to define the RCU (referencing Amazon EC2 Compute Unit) and the cost per time unit of our virtual machine types as listed in **Table 2**. Based on the RCUs, we set the Estimated Execution Time (EET) in seconds of each task by each virtual machine type as listed in **Table 3**. One virtual machine of *medium* type is initially running to host the input files for the workflow.

Table 2 Resource capability units and costs of virtual machine types.

VM Type	RCU	Cost per time unit
medium	3	0.067
large	6.5	0.133
xlarge	13	0.266

Table 3 Estimated execution times of each task based on virtual machine types.

EET	medium	large	xlarge
T1	6	2.8	1.4
T2	5	2.3	1.15
T3	9	4.2	2.1
T4	12	5.6	2.8
T5	4	1.8	0.9

[1]https://aws.amazon.com/ec2/pricing.

We define a base time equal to the time taken for one *medium* (slowest) virtual machine to finish one workflow instance, which is 36 (i.e. the sum of the first column in **Table 3** regardless of workflow structures). The deadlines are then set based on the base time to test the proposed technique against deadlines ranging from tighter (equal to the base time) to more relaxed ones (the base time multiplied by half of the number of parallel loops, which is equal to 50 in the experiment). Thus, each test case is characterized by a deadline multiplier and one of the 3 scheduling heuristics. Each test case is repeated 5 times to calculate the average makespan and the average cost with outliers removed.

The average makespans and the standard deviations of the simulation are shown in **Table 4**. The proposed technique can satisfy the workflow deadline when it is set to 1.5 times of the base time or higher. An unexpected result is that the makespans are almost equal at least up to the deadline of 3 times of the base time. This means that such makespans are the minimum makespans that can be obtained by the proposed technique. In the case of the strictest deadline, the technique cannot react fast enough to instantiate additional virtual machine instances at the very beginning of the execution. However, it can react better with other longer deadlines. In fact, it performs much better than we have anticipated before the experiment, as can be seen in the cases of the deadline multipliers from 2.0 and higher. One reason for this observation is that the technique is 'shocked' by the initially slow progress evaluation (low *PE* value) at the start of the execution and thus starts many additional virtual machine instances in succession (but this happens too late to cope with the strictest deadline). These many virtual machine instances yield makespans much lower than the deadlines.

Table 4 Makespans of the workflow execution.

Heuristic	Deadline (multiplier)					
	36 (1.0)	**54 (1.5)**	**72 (2.0)**	**90 (2.5)**	**108 (3.0)**	**1800 (50)**
Min-Min	35.50	34.93	35.99	38.83	40.98	453.81
SD	0.45	0.60	0.26	0.91	1.01	0.43
Max-Min	41.98	42.29	41.82	42.56	43.08	407.03
SD	1.02	1.06	0.66	1.31	0.44	1.05
XSufferage	42.50	42.23	42.10	42.90	43.46	435.57
SD	0.60	0.52	0.46	0.73	0.81	29.82

The average costs of the experiments are detailed in **Table 5** along with the standard deviations. Because virtual machine instances may wait idly for task instances to be scheduled to them, we separate the cost into the computing cost (when running tasks) and the idle cost (when waiting idly). The computing costs in most cases do not differ much ranging from 222 to 232. This is because the resource capability unit per cost is almost similar in the 3 virtual machine types used in our experiment, which reference the on-demand instances in Amazon EC2.

However, the idle costs are much lower with the Min-Min heuristic when the deadline is stricter. The idle cost of the Min-Min heuristic starts to decrease from the deadline of 1.5 times of the base time, while the idle costs of the other 2 heuristics remain higher and start to decrease from the deadline of 2.5 times of the base time. The Min-Min heuristic prioritizes smaller task instances first, leading to a faster progress feedback to the scheduler. Therefore, the workflow instantiation control can react faster and the resource adjustment mechanism can launch a more proper number of virtual machine instances. Nevertheless, when the deadline is relaxed, such as the base line multiplied by half of the number of parallel loops, all the 3 heuristics incur almost the same idle costs.

Table 5 Costs of the workflow execution.

Heuristic		Deadline (multiplier)					
		36 (1.0)	**54 (1.5)**	**72 (2.0)**	**90 (2.5)**	**108 (3.0)**	**1800 (50)**
Min-Min	Computing Cost	223.92	224.65	226.83	228.67	229.66	232.04
	SD	2.05	2.39	1.97	2.68	1.28	0.23
	Idle Cost	74.87	57.71	31.02	13.41	6.55	2.21
	SD	4.73	7.49	2.85	3.52	0.92	0.26
Max-Min	Computing Cost	222.79	226.34	225.49	227.43	228.88	228.73
	SD	2.53	1.17	2.31	0.90	1.81	0.15
	Idle Cost	89.23	73.69	73.82	46.20	38.32	4.96
	SD	9.20	15.14	3.21	5.93	5.05	0.72
XSufferage	Computing Cost	226.53	225.29	226.21	227.95	229.97	230.92
	SD	2.16	0.90	2.22	2.36	0.91	1.94
	Idle Cost	84.55	74.95	78.06	51.76	30.32	2.77
	SD	4.27	8.48	5.84	10.02	6.14	0.50

In summary, our technique is proven to be practical and prefers a greedy approach such as the Min-Min heuristic to try finishing more task instances as soon as possible in order for the progress evaluation to be more accurate. The technique can satisfy most of the deadlines specified in the experiment except for the strictest deadline. The proposed resource adjustment mechanism that launches one new virtual machine instance in each scheduling round cannot react fast enough to handle the strictest deadline. However, the strictest deadline is equal to the time that the slowest virtual machine takes to execute only one parallel loop while the experiment is set to run 100 parallel loops. This is an extreme case designed to test the proposed technique and it is reasonable that the technique cannot meet the deadline. Moreover, such strict deadlines are unrealistic and are not likely to be used in the real world.

Conclusions

This paper presents a novel scheduling technique for dynamic parallel loop workflows based on the objective of reducing the cost of cloud resource usage while preserving a deadline. The contribution of this paper consists of 3 components. Firstly, the technique features a new method to evaluate the workflow execution progress based on the number of completed tasks and elapsed time in order to avoid relying on traversing the workflow graph traditionally used in existing works. The new workflow progress evaluation is employed in the other 2 components, which are the workflow instantiation control and the cloud resource adjustment proposed in order to cope with the dynamic parallel execution of the parallel loop workflows. According to the simulation results, our technique is practical and incurs lower cost when the workflow deadline is more relaxed. It also prefers task mapping heuristics that facilitate faster progress feedback, which leads to a more accurate progress evaluation.

With our finding, a number of issues can be improved. The current progress evaluation treats all tasks equally; this can lead to an inaccurate estimation when task sizes are dispersed. Normalization of the task sizes may prove useful.

Regarding the execution environment, the technique can be extended to cover a hybrid cloud deployment model in which data transfer times and costs must be considered. Also, the technique can be improved to incorporate a limited number of virtual machine instances, which may involve a virtual machine termination mechanism in order to launch faster ones instead. For such cases, the task mapping heuristic may also need to be specifically designed.

Finally, with the growing concern on the environmental impact, the objective of reducing the carbon footprint or energy consumption has begun to receive attention in workflow scheduling research [11]. However, most of the commercial cloud providers do not make available the information regarding the power consumption of their services to the users. This challenge will thus have to be investigated further.

Acknowledgements

This research is funded by A New Researcher Scholarship of CSTS, MOST by the Coordinating Center for Thai Government Science and Technology Scholarship Students (CSTS) of the National Science and Technology Development Agency under project ID SCH-NR2012-212.

References

[1] D Bhatt. A revolution in information technology: Cloud computing. *Walailak J. Sci. & Tech*. 2011; **9**, 108-13.
[2] D Abramson, C Enticott and I Altintas. Nimrod/K: Towards massively parallel dynamic grid workflows. *In*: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing. Texas, USA, 2008, p. 1-11.
[3] S Smanchat and K Viriyapant. Taxonomies of workflow scheduling problem and techniques in the cloud. *Future Generat. Comput. Syst*. 2015; **52**, 1-12.
[4] M Maheswaran, S Ali, H Siegel, D Hensgen and R Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. *In*: Proceedings of the 8th Heterogeneous Computing Workshop. San Juan, Puerto Rico, 1999, p. 30-44.
[5] H Casanova, A Legrand, D Zagorodnov and F Berman. Heuristics for scheduling parameter sweep applications in grid environments. *In*: Proceedings of the 9th Heterogeneous Computing Workshop. Cancun, Mexico, 2000, p. 349-63.
[6] S Smanchat, M Indrawan, S Ling, C Enticott and D Abramson. Scheduling parameter sweep workflow in the Grid based on resource competition. *Future Generat. Comput. Syst*. 2013; **29**, 1164-83.
[7] S Abrishami, M Naghibzadeh and D Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generat. Comput. Syst*. 2013; **29**, 158-69.
[8] L Bittencourt and E Madeira. HCOC: A cost optimization algorithm for workflow scheduling in hybrid clouds. *J. Internet Serv. Appl.* 2011; **2**, 207-27.
[9] E Byun, Y Kee, J Kim and S Maeng. Cost optimized provisioning of elastic resources for application workflows. *Future Generat. Comput. Syst*. 2011; **27**, 1011-26.
[10] C Enticott, T Peachey, D Abramson, E Mashkina, C Lee, A Bond, G Kennedy, D Gavaghan and D Elton. Electrochemical parameter optimization using scientific workflows. *In*: Proceedings of the 6th IEEE International Conference on E-Science. Brisbane, Australia, 2010, p. 324-30.
[11] Y Lee, H Han, A Zomaya and M Yousif. Resource-efficient workflow scheduling in clouds. *Knowl. Based Syst*. 2015; **80**, 153-62.