DOI 10.12694/scpe.v20i3.1548
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 20, Number 3, pp. 527–540. http://www.scpe.org

OPTIMIZED SCHEDULING APPROACH FOR SCIENTIFIC APPLICATIONS BASED ON CLUSTERING IN CLOUD COMPUTING ENVIRONMENT

WALID KADRI* AND BELABBAS YAGOUBI†

Abstract. Cloud Computing becomes an important technology for the supplying of resources that can be used to execute large-scale scientific applications. It also provides lower-cost Computing resources access with personalized configurations. The user does not have to invest much in the acquisition and management of hardware such as storage, computing, databases, and networking, usually issued by the cloud provider. Users typically pay only for cloud services they use. Scientific applications usually represented as Directed Acyclic Graphs (DAGs) are an important class of applications that lead to challenging problems for resource management in distributed computing. With the advent of Cloud Computing, particularly the Infrastructure as a Service (IaaS) offers on-demand virtual machines executing multiple tasks. These tasks consist of a large number of DAGs requiring elaborated scheduling and resource provisioning policies. In goal of optimization and fault tolerance, DAGs applications are generally partitioned into multiple parallel DAGs using clustering algorithm and assigned to Virtual Machine (VM). In this work, we investigate through simulation, the impact of clustering for both provisioning and scheduling policies in the total makespan and financial costs for execution of user's application. We implemented four scheduling policies well-known in distributed computing systems, and adapted clustering algorithm to our resource management policy that leases and destroys dynamically VMs. We show that dynamic resources management policy can achieve better performance in term of makespan and budget cost than static management policies when partitioning workflow applications into grouped tasks before mapping them on virtual machines (VMs). The execution time and budget cost can be considerably reduced by managing efficiently VMs in order to maximise resources utilization and reduce the number of under-loaded VMs.

Key words: Cloud Computing, Dependent Tasks, Scientific Applications, Workflows, Directed Acyclic Graph, Resource Management, Task Scheduling, Virtual Machine, Clustering, CloudSim Simulator.

AMS subject classifications. 68M14, 68M20

1. Introduction.

Cloud Computing plays an increasingly important role in domains that are closely related with the web, offering a wide variety of services (i.e computing, data storage). These services can be accessed at any time and from any location via a high speed internet access. More specifically, in the domain of scientific applications which are very complex due to the nature of their tasks, and very expensive to schedule and execute them in parallel machines.

Due to fact that resource allocation is an NP-complete problem [13], an optimal assignment for tasks is impossible to realize. Finding an optimal scheduling becomes even more complex when tasks are dependent. It is due to the precedence relation between tasks; For example, when $T_1 \rightarrow T_2$, it means that T_2 cannot start until T_1 has been processed.

Scientific applications can be a set of dependent tasks with different granularity and different level. Many known companies in the Information Technology world offer a wide range of services with on demand payment (the user only pays for what he consumes). These services range from the provision of virtual machine instances such as AmazonEC2 [3], to parallel computing services whose main providers are Google and Yahoo.

Due to the importance of work applications, several research projects have been conducted to develop workflow management systems with scheduling algorithms. The projects: Condor Dagman [27], Gridbus toolkit [22], Iceni [26], Pegasus [11], and so forth, are designed for Grids, whereas cloudbus toolkit [23], SwinDeW-C [30], VGrADS [24], and so forth, are developed for Clouds. These systems can be viewed as a type of platform service facilitating the automation of scientific and commercial applications on the Grid and Cloud by masking their orchestrations and executions.

In this paper, we focus on the way how to manage effectively dependent tasks applications on Cloud environments and measure the impact of clustering algorithm in scheduling. We address the interaction between scheduling and resource management combined with clustering of DAGs and their impact on costs and run-time performance. We also make a comparison between generic approaches by selecting those likely to be the best.

*LIO Laboratory, Deptartment of Computer Science, University of Oran 1, Ahmed Ben Bella, Oran, Algeria (kadri.walid@univ-oran1.dz)

†LIO Laboratory, Deptartment of Computer Science, University of Oran 1, Ahmed Ben Bella, Oran, Algeria (b.yagoubi@univ-oran1.dz)

The paper is organized as follows: the first section introduces cloud computing and includes definitions, architectures, deployment and services models. The second section presents the related work to the issue addressed in this paper, namely DAG scheduling and resources management. Through the definition of objective functions of scheduling, we determine the criteria studied and present tasks scheduling algorithms implemented in our work for results comparison. The third section illustrates in detail the type of Scientific Application Models tested in our work with explaining each of them. Section 4 shows the implementation and simulation parameters like hardware configuration and performance metrics. We also define our simulation environment and a Cloud Computing simulator used and implemented algorithms are also detailed in our approach and the different scenarios simulated. We represent the obtained results into graph and analyze them in terms of performance metrics as time execution, complexity and financial costs. We conclude our paper with a conclusion and some future works.

2. Survey of Scheduling Strategies on Cloud Environment. This section survey all developed algorithm in scheduling of applications in cloud environments like scientific workflows and BoT (Bag of Tasks) or independent tasks. In particular, it focused on techniques considering applications modeled as DAGs and the resource model offered by public cloud providers. It presents a taxonomy of the existing scheduling algorithms on Cloud Computing.

The authors on [18] propose a mathematical model that optimizes the cost of scheduling workflows with a time constraint in a multi-cloud environment where each provider proposes a number of heterogeneous virtual machines or a global storage service for intermediate data files. Their method formulate the scheduling problem as Mixed Integer Program (MIP) and propose an overall optimization of placement and data sharing. Two types of workflow granularity are presented, coarse granularity in which tasks must be run for one hour, and the other for fine granularity workflow with shorter tasks and smaller deadline.

In the paper [20], the authors developed a planning service for middleware in the cloud, allowed optimal use of resources in terms of the total number of resources used in a defined interval given by user. They have proved the feasibility of their solution with taking into account dependencies between services.

The authors on [6] propose a dynamic re-configurable framework for the deployment of scientific workflows in the Cloud (called DR-SWDF). The user customize the workflow deployment process with a given set of objectives and constraints. The DR-SWDF framework offers a dynamic clustering of workflows based on K-means algorithm in order to identify the most convenient techniques or algorithms can be applied for their scheduling and deployment.

The work presented on [14] evaluate the impact performance of HPC applications on Microsoft Azure cloud platform using NAS parallel benchmarks. These benchmarks are used to test the communication performance. They have measured the speedup and MOPS for different scheduling allocation strategies and the results show that allocating one process per instance achieves higher scalability in term of the cost. The results are compared with the same configuration in Amazon platform and found that Azure platform has better shared-memory communication performance than Amazon platform. However, their work was designed for HPC Cloud and not for Cloud computing environment.

Authors in [12] developed the Multi-objective Heterogeneous Earliest Finish Time (MOHEFT) algorithm which as is an extension of the well-known DAG scheduling algorithm HEFT [29]. A set of pareto based solutions are computed from which users can select the suited one. The proposed algorithm builds intermediate workflow schedules, or solutions, in parallel in each step, instead of a single one as is done by HEFT. A crowding distance is used as metric between tasks characterized by dominance relationships in goal of finding solutions.

SABA [17] is a scheduling workflows algorithm proposed in a multi-cloud environment. The authors define the concept of immovable datasets which are restricted to a single data center and cannot be migrated or replicated due to security or cost concerns and movable datasets with no security restrictions and can be replicated. Their approach only considers the CPU capacity of VMs to estimate runtimes and features such as I/O, bandwidth, and memory capacity.

The PSO-based algorithm developed in [19] concerns a cost minimization, and a deadline-constrained algorithm that ensures dynamic provisioning and heterogeneity of computing resources. The authors have modeled the resource provisioning and scheduling as a Particle Swarm Optimization (PSO) problem. The output of the algorithm is a near-optimal schedule when determining the number and types of VMs to use, as well as their

leasing periods and the task to resource mapping. The results of the approach show that the computational overhead increases rapidly with the number of tasks in the workflow and the configuration of VMs provided to the user.

The IaaS Cloud Partial Critical Path (IC-PCP) algorithm [1] aims to minimize execution costs with a deadline constraint. The algorithm begins by recursively identifying a set of tasks with partial critical paths (PCP) associated to each exit node (an exit node is a node without child tasks). The tasks in each path are then placed on the same virtual machine with an instance already leased to meet the latest makespan requirements. In the case where the placement fails, the tasks are assigned to a least expensive, newly instantiated virtual machine that can execute the tasks. The process is repeated until the workflow is fully executed.

Authors in [7] adopt the main heuristic of IC-PCP of [29] and [1] by identifying partial critical paths and assigning their tasks to the same VM. They propose the Enhanced IC-PCP with Replication (EIPR) algorithm for scheduling and allocation that uses the idle time of allocated virtual machines and a budget exceeding to replicate tasks in order to minimize performance variations and deadline of applications. The algorithm starts by determining the number and type of virtual machines to use, the order and placement of tasks on these resources, and then determines the start time and end time of virtual machines.

Authors in [28] focus on their work on SLA when scheduling Workflow by implementing a SaaS provider offering a workflow execution service. They consider two types of SLA contracts that can be used to lease VMs from IaaS providers: static and subscription based. Specifically they consider offers configuration of Amazon EC2 [3], namely on-demand and reserved instances. In their proposed model, the SaaS provider has a pool of reserved instances that are used to execute workflows before a user-defined deadline. However, if the reserved instance infrastructure is not enough to satisfy the deadline, then on-demand instances are acquired and used to meet the workflow's requirements. Their algorithm is capable of selecting the best-suited IaaS provider as well as the VMs required to guarantee the QoS parameters.

The authors of [31] propose a scheduling framework that takes into account the dynamic nature of cloud environments in terms of performance and pricing. It is based on the same resource model as Amazon EC2 and takes Spot and on-demand instances into account. The main objective is to minimize the costs of executing workflows by providing a guarantee on probabilistic calculated deadlines based on the variability of resource performance and price of Spot instances. Spot instances aim to reduce infrastructure costs while on-demand instances ensure deadline compliance when spots instances are unable to complete tasks on time. This is achieved by generating a static hybrid instance configuration plan (a combination of spot and on-demand instances) for each task.

The authors of [25] implement various VM allocation and VM selection methods to effectively schedule the user tasks on the fog computing resources and try to find the best combination of task scheduling combination for the effective and optimized user data processing.

In the paper [16], authors propose hybrid balanced task clustering algorithm that uses the parameter of impact factor of workflows along with the structure of workflow. According to this technique, tasks can be considered for clustering either vertically or horizontally based on the value of the impact factor. This minimizes the system overheads and the makespan for execution of a workflow. A simulation based evaluation is performed on real workflows that shows the proposed algorithm is efficient in recommending clusters. It shows improvement of 5-10% in makespan time of workflow depending on the type of workflow used.

In the paper [5], Biswas and *al.* have proposed a Gravitational Search Algorithm (GSA) based workflow scheduling for heterogeneous computing systems. The proposed algorithm considers many objectives such as minimization of makespan, load-balancing, and energy-consumption. Their algorithm is designed to generate a valid execution sequence of tasks recursively with respecting the precedence relationship.

The authors of [2] present a non-dominance sort based Hybrid Particle Swarm Optimization (HPSO) algorithm to handle the workflow scheduling problem on Iaas clouds. The algorithm is a hybridation of their previous algorithm called Budget and Deadline constrained Heterogeneous Earliest Finish Time (BDHEFT) algorithm and multi-objective PSO. The HPSO heuristic tries to optimize makespan and cost under deadline and budget contraints and presents best solutions using pareto, and the final choice is done by user.

Kanagaraj and *Swamynathan* propose on [15] an effective resource provisioning and scheduling mechanism for workflow on cloud computing environment. The main idea is to determine the required number of VMs and

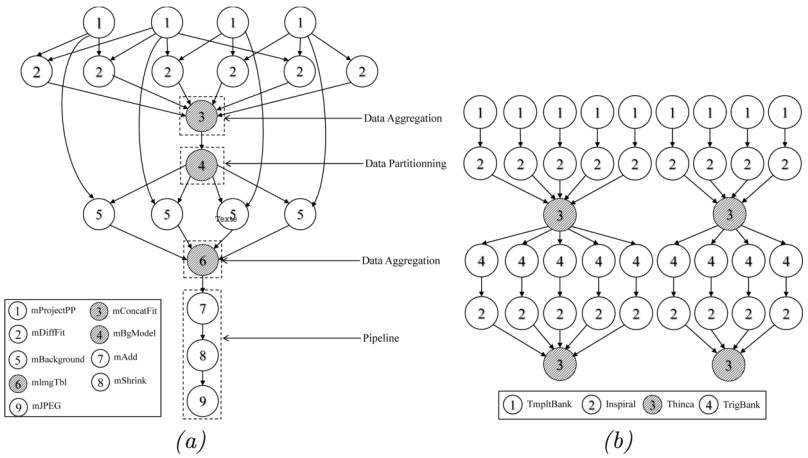

FIG. 3.1. *Montage DAG (a), and LIGO DAG (b).*

their configuration in a goal of optimizing data transfer between workflow tasks.

3. Scientific Applications Model (DAGs). In this section, we enumerate some of scientific applications (DAGs or also called Workflows) given by user and scheduled to VMs for execution using different assignment algorithms. To generate such scientific applications, we used a Pegasus Workflow Generator for DAX (Directed Acyclic Graph in XML) [21, 10] that provides a resource independent workflow description. It captures all the tasks that perform computation, the execution order of these tasks represented as edges in a DAG, and for each task the required inputs, expected outputs, and the arguments with which the task should be invoked. Pegasus provides simple, easy to use programmatic API's in Python, Java, and Perl for the DAX generation. We used the JAVA's API one for our work. The reason why we used a DAX generator is to avoid random generation of DAGs (nodes and links) which cannot be able to prove the effectiveness of our scheduling algorithm. The workflow applications used on our simulation are taken from [10] and described as follow:

3.1. Montage. This type of workflow application shown in Fig. 3.1 has been developed by the NASA/IPAC Infrared Science Archive and used to generate custom mosaics of the sky using images in the Flexible Image Transport System (FITS) format as input.

3.2. LIGO. Laser Interferometer Gravitational Wave Observatory (LIGO) scientific applications are used to search for gravitational wave as shown in Fig. 3.1.

3.3. CyberShake. CyberShake is a scientific application used in geology domain that calculates Probabilistic parameters for geographic sites. It identifies all ruptures and then calculates synthetic seismograms for each rupture variance (Fig. 3.2(a)).

3.4. Epigenomics. Epigenomics is a data-parallel scientific application used in Genetic domain. The Fig. 3.2.(b) shows this type workflow witch consists on Genetic data analyzed in the form of DNA sequence lanes. Each analyze can generate multiple lanes of DNA sequences and converted into a specific format. The mapping software can do one of two major tasks. The scientific application maps DNA sequences to the correct locations in a reference Genome.

3.5. SIPHT. Conducts a wide search for small untranslated RNAs (sRNAs) that regulates several processes such as secretion or virulence in bacteria. The Fig. 3.2.(c) shows a representation of Sipht DAG. The kingdom-wide prediction and annotation of sRNA encoding genes involves a variety of individual programs that are executed in the proper order using Pegasus.

4. Implementation and Simulation. This section presents the performance metrics used, different executed scenarios, the obtained results in the simulations, and their interpretations.

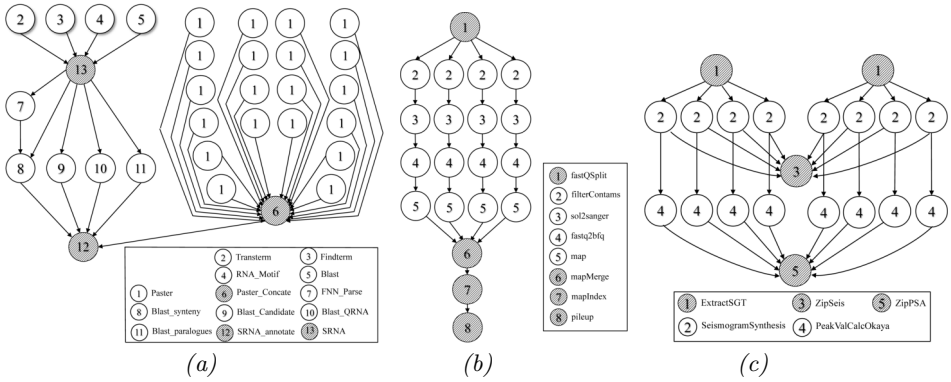

FIG. 3.2. *CyberShake DAG (a), and Epigenomics DAG (b) and Sipht DAG (c)*

4.1. Proposed Resources Management Approach. The algorithm 1 was designed after testing numbers of approaches for creating VMs. It allows to create dynamically virtual machines on demand and power-off the unused ones after a period of time. In fact, it is very difficult to provide a precise number of VMs that will be needed for the application, due to the nature of the jobs (DAGs) and their difference in term of execution time, and user's budget cost.

After testing the number of application, execution time, and finally the depth of each application (dependency levels), we have developed the algorithm described in Alg. 1. The main idea is to browse every level of dependence on the application, from the root to the leaves and increment the number of created VMs at every parallel jobs found, i.e after partitioning the graph into independent jobs as described in section 4.3.

Algorithm 1 VMs Management Algorithm

Require: *jobs*: Set of arrival jobs at t; t: current time;
function VMSCREATION(jobs)
  $VmList \leftarrow \emptyset$ ▷ List of VMs to return
  $Size \leftarrow$ ESTIMATESIZE($jobs$); $VmList \leftarrow$ REQUESTVMS($Size$) ▷ Send request to DataCenter
  return $VmList$
end function
function ESTIMATESIZE(jobs)
  $size \leftarrow 1$; $p_{previous} \leftarrow 0$ ▷ Depth of the previous job
  $depth_{max} \leftarrow jobs.getDepth()$ ▷ the maximal depth of jobs
  for $j \in jobs$ **do**
    $p_{Current} \leftarrow j.getDepth()$
    if $p_{Current} == p_{previous}$ **then** $size++$;
    end if
  end for
  return $size/depth_{max}$
end function

4.2. Creation and Submission of jobs. Workflows are generated from XML files. Generation of applications is as follows:
— A file is given as input. The file is interpreted from its XML structure and tasks are created.
— Jobs are then processed to add dependencies between tasks and creating a dependence matrix.
— Jobs are passed to the main Scheduler, which is responsible for dispatching jobs through the resource.
— Clustering process can be applied to jobs before scheduling.
The job creation and submission shown in Fig. 4.1 illustrate an example of the process from job submission by user until getting results feedback after determining the adequate job scheduling algorithm.

4.3. Clustering Algorithms. Clustering Algorithms (partitioning algorithms) consists on grouping different dependant tasks in the same cluster and executing them on the same virtual machine. A problem becomes complex when the number of clusters is larger than the number of the available virtual machines. It means that

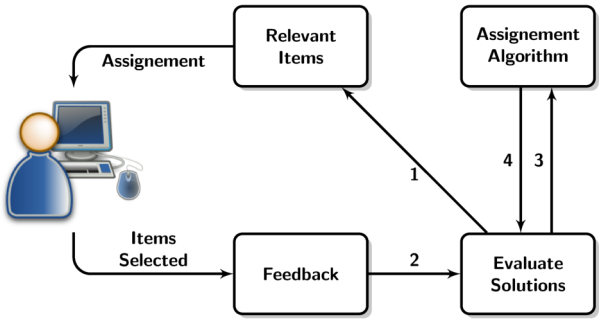

FIG. 4.1. *Creation and DAGs scheduling Process*

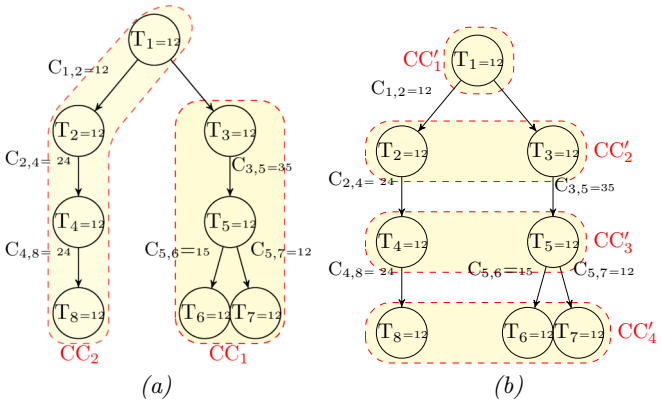

FIG. 4.2. *Vertical Clustering (a), and Level-based clustering (Horizontal Clustering) (b)*

the scheduling of several clusters onto the same virtual machines can increase considerably the overall execution time. We have to find the best mapping for grouped tasks on virtual machine according to its capability and its load. A load balancing algorithm is used in our case to achieve good performance.

In our work, we must determine a certain number of clusters which have a minimum makespan to be assigned into a virtual machine. We have used generic algorithms for tasks partitioning to decompose workflows (DAGs) submitted by user into related component (clusters) in order to schedule them to the same VM in order to minimize time response by reducing intercommunication costs between dependant tasks.

Two approaches have been adapted to our assignment approach: (i) **vertical clustering** described in algorithm 2 and Fig. 4.2 (a), and (ii) **horizontal clustering** described in algorithm 3 and Fig. 4.2 (b).

4.4. Simulation.

4.4.1. Configuration and execution parameters. The simulation was executed on a 64-bit MacOSx 12 work environment. The development environment used is Netbeans ver.7.0.1, all on a MacBook Pro machine with a i7 2.40GHz processor and 8GB of RAM. Simulated infrastructure is a Datacenter with 100 physical machines. Each physical machine can support 10 VMs, details on the configurations of the machines are given in the Table 4.1.

The simulator used is WorkflowSim [9] is an extension of CloudSim Simulator [8].

4.4.2. Performances Metrics. In this paragraph, we present the performance measures on which we have supported to first interpret the results and compare the different approaches. The two main performance measures are the overall implementation time and financial cost. These are conventional steps to test the effectiveness of scheduling algorithms and resource management.

Makespan . T_i being the end date of the job i

$$Makespan = \max T_i \tag{4.1}$$

Algorithm 2 Vertical Clustering Algorithm

Require: :$stack \leftarrow \emptyset$; $JobsList \leftarrow \emptyset$; $List_{temporary} \leftarrow \emptyset$

```
1: function VertiClustering(TasksList, dephtJob)                ▷ Level number of dependency
2:     if dephtJob > 0 then
3:         for t ∈ TasksList do
4:             if t.depth == 0 then Push(stack, t)               ▷ Stacking roots
5:             end if
6:         end for
7:         while stack ≠ ∅ do
8:             node ← Pop(stack)
9:             if !processed(node) then                          ▷ if the node have not been processed yet
10:                nbParent ← node.getNbParent()                 ▷ Number of previous tasks
11:                nbChildren ← node.getNbChildren()             ▷ Number of next tasks
12:                for cNode in node.getChildList() do
13:                    Push(stack,cNode)
14:                end for                                       ▷ Stacking all next tasks
15:                if nbParent! = 0 then                         ▷ if Root, Nothing to do
16:                    if nbParent > 1 then
17:                        if nbChildren > 1 ∨ nbChildren == 0 then
18:                            Listetemporary.add(node)
19:                            addToJobs()
20:                        else                                  ▷ Only one next task
21:                            addToJobs(); Listetemporary.add(node)
22:                        end if
23:                    end if
24:                    if nbChildren > 1 ∨ nbChildren == 0 then
25:                        addToJobs(); Listetemporary.add(node)
26:                    end if
27:                end if
28:            else addToJobs()
29:            end if
30:            tag(node)                                         ▷ Tag the node as processed
31:        end while
32:    end if
33:    return JobList
34: end function
35: function addToJobs( )
36:    if Listetemporary ≠ ∅ then
37:        JobList.add(CreateJob(Listetemporary)); Listetemporary ← ∅
38:    end if
39: end function
```

Table 4.1
Configuration of simulated machines

Configuration	physical Machine	VM
MIPS (Million Instructions per sec.)	2000	1000
RAM	4048 (MB)	2048 (MB)
PE (Processing Element)	4	2
Bandwidth	10000 (MB/S)	1000 (MB/S)
Space (storage)	1 (TB)	6 (GB)

$Makespan$ is simply the date of the last job to be done among all executed jobs.

Budget Cost. For the budget, the calculation model used is similar to a *a1.medium* machine price of *AmazonEC2 platform*[4], i.e periods of an hour for the rental of virtual machines. To simplify the interpretation of results, VMs costs have been reduced to $0.051\$/Hour$. Once a machine is created, an additional cost is added, rounding the cost another time.

Algorithm 3 Horizontal Clustering Algorithm

```
1: function HorizClustering(TasksList, ClusterSize)
2:     depht_max ← 0 ; JobsList, List_temporary ← ∅                ▷ Partitioned Jobs to return
3:     i ← 0; TasksList_temporary ← 0
4:     if ClusterSize > 0 then                                      ▷ Size of partition
5:         for Task t ∈ TasksList do
6:             if t.depth > depht_max then
7:                 depht_max ← t.depth
8:             end if
9:         end for
10:        while i < depht_max do                                   ▷ For every level of tasks graph
11:            for Task t ∈ TasksList do
12:                if t.depth == i then
13:                    Push(Liste_temporary, t)
14:                end if                                           ▷ Temporary list for every level of graph
15:            end for
16:            Shuffle(List_temporary)                              ▷ Random Permutation
17:            while List_temporary ≠ ∅ do                          ▷ All tasks of this level not yet gathered
18:                if ClusterSize < List_temporary.length then
19:                    for j ← 0 ; j < ClusterSize do               ▷ Group tasks as cluster tasks
20:                        Task t ← Pop(List_temporary); Push(TasksList_temporary, t)
21:                    end for
22:                    JobsList.add(CreateJob(TasksList_temporary))
23:                else
24:                    for j ← 0 ; Liste_temporary.length do
25:                        Tache t ← Pop(List_temporary); Push(TasksList_temporary, t)
26:                    end for
27:                    JobsList.add(CreateJob(TasksList_temporary))
28:                end if
29:                TasksList_temporary ← ∅
30:            end while
31:        end while
32:    end if
33:    return JobsList
34: end function
```

Cost/VM. The cost for the ith VM is :

$$Cost_{VM_i} = \frac{Time_{secondes}}{3600} * 0.051\$ \tag{4.2}$$

Overall cost. The overall cost is :

$$Cost = \sum_{i=1}^{NumberVM} Cost_{VM_i} \tag{4.3}$$

4.4.3. Implemented algorithms for comparison.

FCFS (First Come First Served). is the simplest scheduling algorithm that simply queues tasks in the order that they arrive in the ready queue. The tasks that comes first will be executed first and next tasks starts only after the previous gets fully executed. It provides efficient, error-free and simple process for scheduling by saving the VMs or resources in cloud computing.

Minimum Execution Time (MET). Minimum Execution Time (MET) algorithm determines the best predictable completion time for VMs for a given task and decide to assign it to the this resource without regarding to its availability. The MET algorithm can sometimes lead to high load imbalance since the assignment is not dependent on the availability of VMs.

MinMin. MinMin algorithm selects from a set of unscheduled tasks and determines for each task the minimum completion times all virtual machines. The task with generally minimum completion time is chosen and scheduled on the resultant virtual machine. The scheduled task is then removed from task queue and the process is repeated until the all unscheduled tasks are performed.

TABLE 4.2
Specification of simulated scenario and arrival time of of each workflow

Application	Arrival Time
Insipral 1000 Tasks	t=0ms
Epigenomics 460 Tasks	t=100s
LIGO 1000 Tasks	t=200s
CyberShake 3000 Tasks	t=250s
Montage 250 Tasks	t=350s
Sipht 1000 Tasks	t=380s

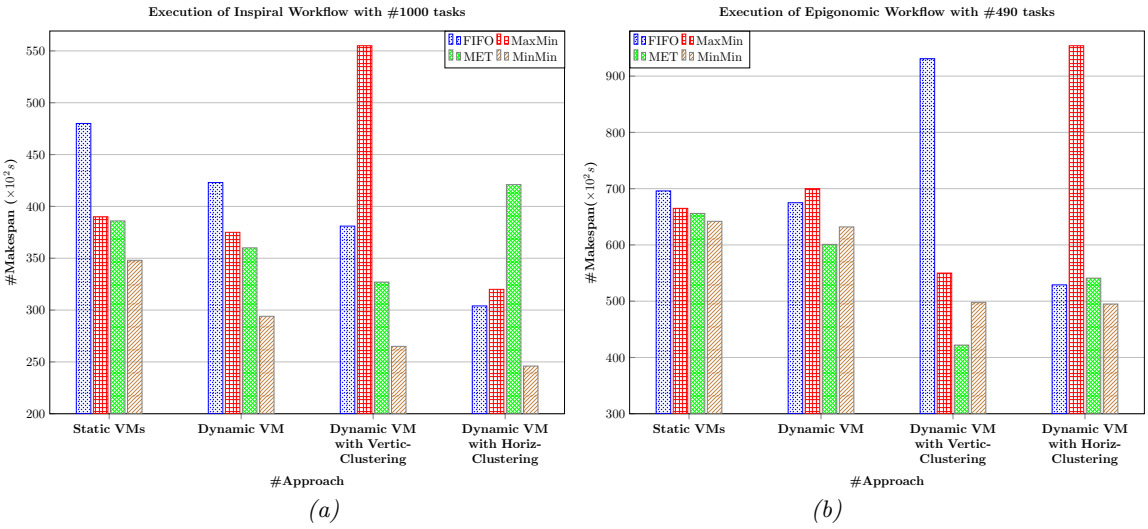

FIG. 5.1. *Makespan for Inspiral (a) and Epigenomic (b)*

MaxMin. Similar to the MinMin algorithm, it determins the completion times for each task on all virtual machines, the task with maximum completion time is scheduled on the consistent virtual machine in the case of MaxMin, and the process is the repeated until all the tasks are scheduled. MaxMin algorithm is usually employed in a situation where there are fewer longer and shorter tasks. Smaller makespan low degree of imbalance among virtual machines is guaranteed if more tasks are scheduled on machines that execute them earliest and fastest due to the predictable makespan and the real time workload evaluation of VMs.

4.4.4. Simulation Scenario. The scenario realized aims to model applications in realistic way, and that, by highlighting several applications with deadlines arrivals between them. The Table 4.2 shows the generated applications in this scenario, with the number of tasks included on each of them. The number of jobs remains the same when the configuration is changed.

5. Simulation Results and Discussion. This section summarizes the findings and contributions made. we will illustrate some experimental results in order to demonstrate the effectiveness of our dynamic management and scheduling resource of the different workflows described on section 3 and measuring the impact of clustering on them. As we can see in Figs. 5.1 (a), 5.1 (b), 5.2 (a), 5.2 (b), 5.3 (a), 5.3 (b) show respectively the makespan time scheduling of Inspiral, Epigenomics, LIGO, Cybershake, Montage, and Sipht scientific workflows into VMs according to implemented scheduling algorithms. We have tested 4 generic algorithms described in section 4.4.3 FIFO, MET MaxMin, and MinMin with an initial number of VMs fixed to 50, and have combined each one with static management VM, our Dynamic Management VM approach, our Dynamic VM management approach combined vertical Clustering, and finally with our Dynamic VM management approach combined horizontal Clustering. We have also measured in Fig. 5.4, the cost of running the user's application described in Table 4.2 and the average resource occupation in terms of CPU and Memory (see Fig. 5.5 (a) and Fig. 5.5(b)).

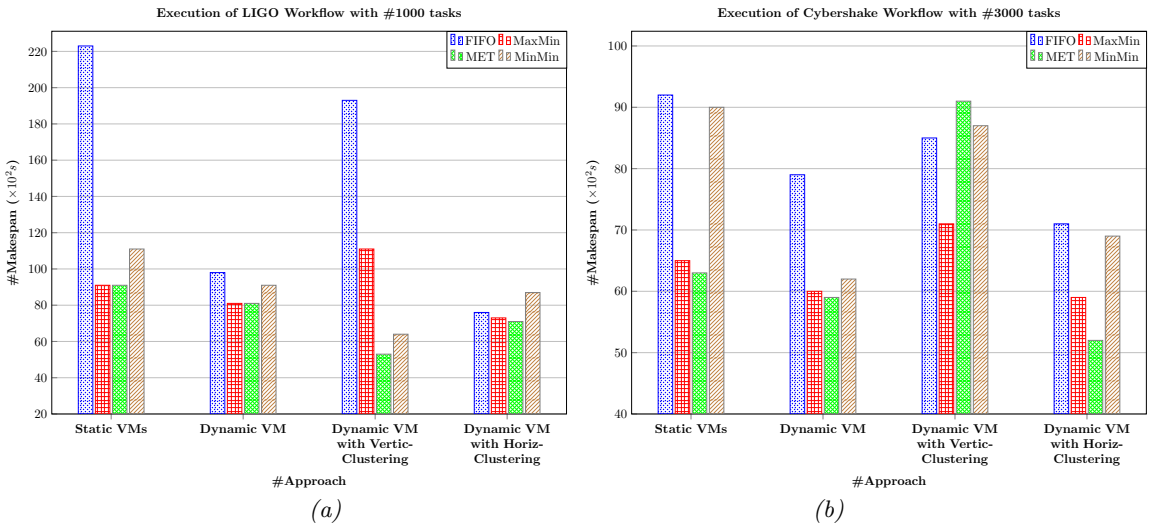

FIG. 5.2. *Makespan for LIGO (a) and Cybershake (b)*

The execution cost is shown in Fig. 5.6.

Figs. 5.1 (a) and 5.1 (b) show the maskespan in second of respectively the inspiral workflow with 1000 tasks and epigenpmics worflows with 460 tasks. From these results it is clear that the dynamic VM management with horizontal clustering gives better results than the other ones expects of MaxMin in vertical clustering (see Fig. 5.1 (a)) and horizontal clustering (see Fig. 5.1(b)) which give the worst makespan. This is due to the nature of Inspiral and Epigenomic worflows. The first one didn't support the vertical clustering due to the number of level of the DAG, and the second didn't support the horizontal clustering due to its number of nodes dependencies at the same level. In the Fig. 5.2 (a), the Ligo worfklow with 1000 tasks is performing very good results using our dynamic VM management approach comparing with the static one. The same analysis is given for Fig. 5.2 (b) which illustrate the cybershake workflow with 3000 tasks. The obtained results in Figs. 5.2 (a) and 5.2 (b) show that the makespan is better and the makespan decreases considerably from $230 \times 10^2(second)$ to $75 \times 10^2(second)$ for FIFO and from 90×10^2 to 72×10^2 for MaxMin when using Horizontal clustering with our management approach (see Fig. 5.2 (a)).

The Fig. 5.2 (b) globally gives good results with dynamic VM management. But we notice that when applying vertical clustering, MET gives the worst makespan. This is due to the vertical partitioning of the cybershake workflow into related jobs, this increase communication cost of the different clusters of this latter and as results, the makespan increase when waiting dependency results from jobs. It is worth discussing these interesting facts revealed by this results and can say that partitioning workflows didn't always give best makespan, it depends on the complexity of the workflow. In Fig. 5.3 (b) our dynamic management VM didn't perform good results with the Fifo algorithm, because of we didn't apply clustering, and when creating VM dynamically, we can disable non-allowed VM, and enable it when needed. This operation take an additional time which is non negligible and increase the makespan of the user's application.

We can also say that the adequate clustering for Sipht is the vertical one for MaxMin, MinMin end MET scheduling algorithm for our dynamic management algorithm, and the horizontal clustering for FIFO as shown in Fig. 5.3 (b). For the makespan of Ligo workflow, better is to use the dynamic VM management with vertical clustering for MET and MinMin, and the dynamic VM management with horizontal for FIFO and MaxMin scheduling algorithm.

Extensive results are illustrated in Fig. 5.4 carried out show that our dynamic management of VMs method reduce the user's budget cost for execution of its application. We can see that the cost of running the application (user's scenario described bellow) decrease considerably from approximately 5.5$ to 1.23$ when using dynamic VM management. Because of shutting down unused VMs and reducing makespan of user application.

However, in some cases, it's better using vertical than horizontal clustering when using complicated workflow

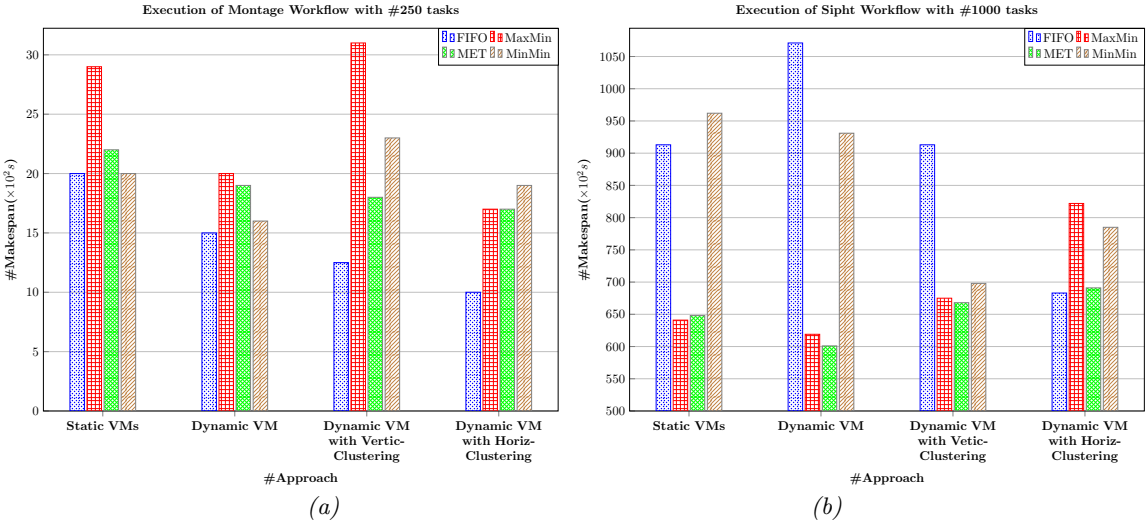

FIG. 5.3. *Makespan for Montage (a) and Sipht (b)*

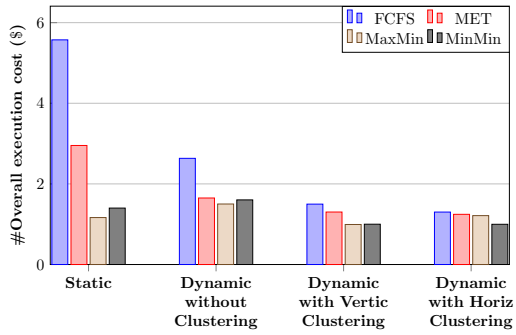

FIG. 5.4. *Execution Costs($) for the simulated scenario*

with a lot of dependencies such as LIGO and Sipht.

However, applying any clustering yields a lower cost as shown in Fig. 5.4. This can be explained by the effect of grouping tasks and mapping them into the same virtual machines. Indeed, the number of VMs created is more concise, and the VMs are used more efficiently. To conclude, we can say that the FCFS gives a higher costs than MET, MaxMin and MinMin algorithms which have slightly different costs. The obtained results in different simulation scenarios let us to think about proposing an adaptation of the vertical and horizontal clustering algorithm as a perspective to further reduce the cost of execution.

The Fig. 5.5 measures according to simulation time the number of VMs allowed for scheduling scenario 1 for both static (see Fig. 5.5 (a)) and dynamic (see Fig. 5.5 (b)) VM allocation. The dynamic VM allocation performs well, giving the minimum number of VM. It leads us to reduce the execution cost without increasing makespan. For the No clustering of DAGs, the number of VM converge to 36 VMs for static approach when this latter decrease to 26 VMs for our dynamic resource management. We can say that best results are obtained with the vertical clustering which is 20 VMs for the static resource management and 6 VMs the dynamic resource management. The peak of VM number (50 VM) in the graph is the started number of VM created initially is fixed in our simulation to 50 VMs.

The Horizontal clustering in Figs. 5.5 (a) and 5.5 (b) give medium results for the overall application, This suggests that the number of tasks of different workflows composing the user's application.

We describe the results of resource occupation graph, which is shown in Fig. 5.6. We understand why the application's execution cost is so high in Fig. 5.3 (a) when using static management approach. It demonstrates

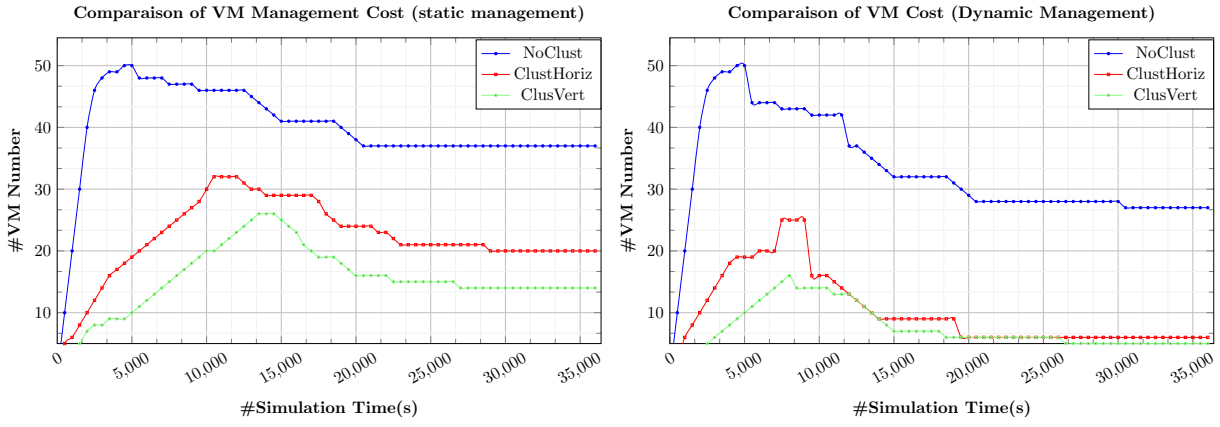

FIG. 5.5. *VMs management costs and clustering impact (Static (a) vs Dynamic (b))*

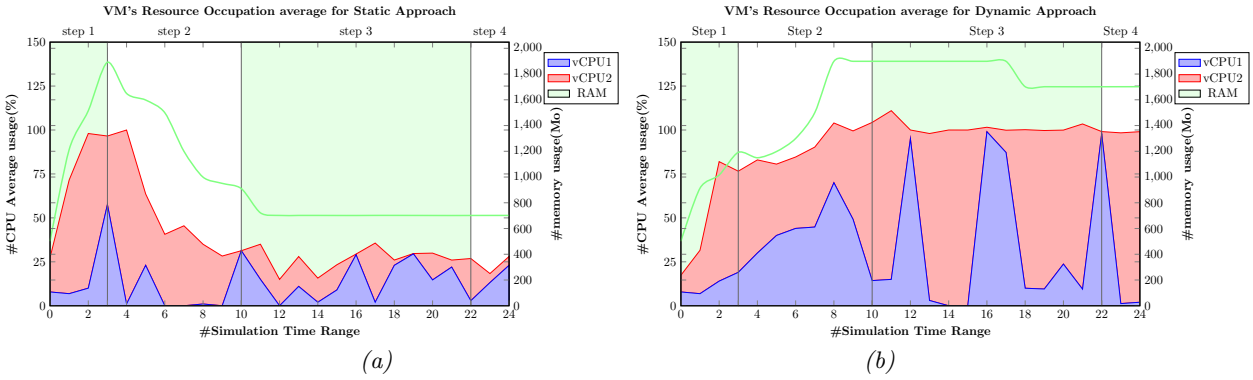

FIG. 5.6. *Resource occupation average of VM (Static (a) vs Dynamic (b))*

two things:
– First, with static management VMs, the resource utilization is not maximized and the VMs are running jobs but not effectively, and we notice that CPU average not exceed 35% the most of time. The same thing for the memory, it not exceeds 600MB from the hole 2048MB allowed for one Virtual Machine (see Fig. 5.6 (a)).
– Second, by reducing the number of VMs and maximizing the utilization of resource using our dynamic approach like showed in Fig. 5.6 where CPU average occupation is globally 100% and memory approaching the max capacity allowed (1400MB), we can reduce considerably the budget cost of running complex scientific application submitted by user.

From these results it is clearly proved that dynamic management approach reduce considerably the makespan and performs better results on most of cases. It depends on the complexity of scientific workflow submitted by the user, clustering can impact positively on performance metrics. But our approach has some limits and depends on partitioning the graph which can lead to higher makespan if the scientific workflow is very complex.

But globally, our approach leads to good results, even if the improvement is negligible in some cases.

6. Conclusion and Future Work. As big are the opportunities that cloud can offer, scheduling complex scientific application (workflows) and resource allocation plays a critical role on the Cloud. In fact give the user the ability to allocate the resources it needs is to give the control over the resources, so that he can manage by himself.

A proper dynamic scheduling mechanism will aid the user to reduce the cost and time of workflow execution. An optimized dynamic scheduling workflow proposed in this paper, analyses the structure of workflows and suggests an optimized resource provisioning approach after partitioning dependent tasks on related com-

ponent. This helps the users to allocate optimum number of resources with the required configuration reducing considerably execution cost and makespan.

In this case, we have addressed the problem of an optimized resource management and scheduling approach of tasks and their influence on the performance and cost execution of complex scientific applications running on IaaS Cloud. The initial objectives were:
— Propose a optimized scheduling approach and resource management policy for an IaaS cloud.
— Implement and study the impact on performance metrics of our approach with other scheduling policies and workflows clustering.
— Finding best combinations of scheduling and clustering depending on nature of workflows applications.

Four generic task scheduling algorithms were implemented and combined with two partitioning approaches for workflows. These algorithms were compared with our dynamic VM management policy that manage dynamically virtual machines according to their workload balance, to the nature and size of workflows.

Clustering the workflows before scheduling them with our dynamic management policy of VMs seems to be very effective for complex DAGs applications we tested. The MaxMin and MinMin algorithms which start with the largest (respectively, smaller) jobs and scheduling them on best resources, were slightly better on the FCFS (first come first served) and MET approaches, the first placing the jobs according to their arrival order by relegating the others in the queue, while the second takes the logic of the first without queuing them.

The simulation outcomes express that dynamic scheduling approach algorithm increases the utilization of resource and reduces the response time for workflow scheduling. The obtained results are quite encouraging and many perspective still open for our future work. We can mention some of these future directions:
— Integrating new tolerance failure mechanism of computing resources and migration techniques of VMs them into another VMs with saving its instance.
— Considering data storage for scientific applications that can affect considerably performance and execution cost.

REFERENCES

[1] S. Abrishami, M. Naghibzadeh, and D. H. Epema, *Deadline-constrained Workflow Scheduling Algorithms for Infrastructure As a Service Clouds*, Future Gener. Comput. Syst., 29 (2013), pp. 158–169, 10.1016/j.future.2012.05.004.

[2] V. Amandeep and K. Sakshi, *A Hybrid Multi-Objective Particle Swarm Optimization For Scientific Workflow Scheduling*, Parallel Computing, 62:C (2017), pp. 1–19, 0.1016/j.parco.2017.01.002.

[3] *Amazon EC2*, September (2017), https://aws.amazon.com/ec2.

[4] *Amazon EC2 Pricing*, March (2019), https://aws.amazon.com/fr/ec2/pricing/on-demand/.

[5] T. Biswas, P. Kuila, A. K. Ray, and M. Sarkar, *Gravitational Search Algorithm Based Novel Workflow Scheduling for Heterogeneous Computing Systems*, journal of Simulation Modelling Practice and Theory, 96 (2019), pp. 101932, 10.1016/j.simpat.2019.10193.

[6] K. Bousselmi, Z. Brahmi, and M. Mohsen Gammoudi, *DR-SWDF: A Dynamically Reconfigurable Framework for Scientific Workflows Deployment in the Cloud*, j-SCPE., 18:2(2017), pp. 177–193.

[7] R. N. Calheiros and R. Buyya, *Meeting Deadlines of Scientific Workflows in Public Clouds with Tasks Replication*, IEEE Trans. Parallel Distrib. Syst., 25 (2014), pp. 1787–1796, 10.1109/TPDS.2013.238.

[8] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, *Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*, Softw. Pract. Exper., 42 (2011), pp. 23–50.

[9] W. Chen and E. Deelman, *WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments*, in Proceedings of the 2012 IEEE 8th International Conference on E-Science (e-Science), Washington, DC, USA, 2012, IEEE Computer Society, pp. 1–8, 10.1109/eScience.2012.6404430.

[10] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny, and K. Wenger, *Pegasus, a Workflow Management System for Science Automation*, Future Gener. Comput. Syst., 46 (2015), pp. 17–35, 10.1016/j.future.2014.10.008.

[11] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, M. Livny, *Pegasus: Mapping Scientific Workflows onto the Grid*, in Grid Computing, Second European Across Grids Conference, AxGrids 2004, Nicosia, Cyprus, January 28-30, 2004, Revised Papers, 2004, pp. 11–20, 10.1007/978-3-540-28642-4_2.

[12] H. M. Fard, R. Prodan, J. J. D. Barrionuevo, and T. Fahringer, *A Multi-objective Approach for Workflow Scheduling in Heterogeneous Environments*, in Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012), Washington, DC, USA, 2012, IEEE Computer Society, pp. 300–309, 10.1109/CCGrid.2012.114.

[13] M. R. Garey,and D. S. Johnson, *Using NP-Completeness to Analyze Problems*, in Computers and Intractability A guide to the theory of NP-completeness., 29 (2002), pp. 34–48.

[14] H. A. Hassan, A. I. Maiyza , and W. M. Sheta,*Impact of Process Allocation Strategies in High Performance Cloud Computing on Azure Platform*, j-SCPE., 18:2(2017), pp. 161–176, https://www.scpe.org/index.php/scpe/article/view/1288.
[15] K. Kanagaraj and S. Swamynathan, *Structure aware resource estimation for effective scheduling and execution of data intensive workflows in cloud*, Future Gener. Comput. Syst, 79 (2018), pp. 878–891.
[16] A. Kaur, P. Gupta, and M. Singh, *Hybrid Balanced Task Clustering Algorithm For Scientific Workflows in Cloud Cmputing*, j-SCPE, 20:2 (2019), pp. 237-258, 10.12694/scpe.v20i2.1515.
[17] Z. Lingfang, V. Bharadwaj, and L. Xiaorong, *SABA: A security-aware and budget-aware workflow scheduling strategy in clouds*, J. Parallel Distrib. Comput., 75 (2015), pp. 141–151, 10.1016/j.jpdc.2014.09.002.
[18] M. Malawski, K. Figiela, M. Bubak, E. Deelman, and J. Nabrzyski, *Scheduling Multilevel Deadline-constrained Scientific Workflows on Clouds Based on Cost Optimization*, Sci. Program., 2015 (2015), pp. 5:5–5:5, 10.1155/2015/680271.
[19] M. Masdari, F. Salehi, M. Jalali, and M. Bidaki, *A Survey of PSO-Based Scheduling Algorithms in Cloud Computing*, J. Netw. Syst. Manage., 25 (2017), pp. 122–158, 10.1007/s10922-016-9385-9.
[20] F. Nizamic, V. Degeler, and R. Groenboom, *Policy-Based Scheduling of Cloud Services*, j-SCPE., 13:3(2012), pp. 187–199.
[21] Pegasus, *Workflow Generator*, 2018.
[22] B. Rajkumar and V. Srikumar, *The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report*, CoRR, cs.DC/0404027 (2004).
[23] B. Rajkumar, P. Suraj, and V. Christian, *Cloudbus Toolkit for Market-Oriented Cloud Computing*, CoRR, abs/0910.1974 (2009).
[24] L. Ramakrishnan, C. Koelbel, Y.-S. Kee, R. Wolski, D. Nurmi, D. Gannon, G. Obertelli, A. YarKhan, A. Mandal, T. M. Huang, K. Thyagaraja, and D. Zagorodnov, *VGrADS: Enabling e-Science Workflows on Grids and Clouds with Fault Tolerance*, in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, New York, NY, USA, 2009, ACM, pp. 47:1–47:12, 10.1145/1654059.1654107.
[25] S. P. Singh, A. Nayyar, H. Kaur, and A. Singla, *Dynamic Task Scheduling Using Balanced VM Allocation Policy For FOG Computing Platforms*, j-SCPE, 20:2 (2019), pp. 433-456.
[26] M. Stephen, Y. Laurie, A. Ali, N. Steven, and D. John, *Workflow Enactment in ICENI*, in In UK e-Science All Hands Meeting, Publishing Ltd, 2004, pp. 894–900.
[27] D. Thain, T. Tannenbaum, and M. Livny, *Condor and the Grid*, 2003, 10.1002/0470867167.ch11.
[28] A. L. G. Thiago, F. B. Luiz, and R. M. M. Edmundo, *Workflow scheduling for SaaS / PaaS cloud providers considering two SLA levels*, in 2012 IEEE Network Operations and Management Symposium, NOMS 2012, Maui, HI, USA, April 16-20, 2012, 2012, pp. 906–912, 10.1109/NOMS.2012.6212007.
[29] H. Topcuouglu, S. Hariri, and M. you Wu, *Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing*, IEEE Trans. Parallel Distrib. Syst., 13 (2002), pp. 260–274, 10.1109/71.993206.
[30] Y. Yun, L. Ke, C. Jinjun, L. Xiao, Y. Dong, and J. Hai, *An Algorithm in SwinDeW-C for Scheduling Transaction-Intensive Cost-Constrained Cloud Workflows*, in Fourth International Conference on e-Science, e-Science 2008, 7-12 December 2008, Indianapolis, IN, USA, 2008, pp. 374–375, 10.1109/eScience.2008.93.
[31] A. C. Zhou, B. He, and C. Liu, *Monetary Cost Optimizations for Hosting Workflow-as-a-Service in IaaS Clouds*, IEEE Trans. Cloud Comput., 4 (2016), pp. 34–48, 10.1109/TCC.2015.2404807.

Edited by: Dana Petcu
Received: April 8, 2019
Accepted: July 26, 2019