SN Computer Science (2021) 2:456
https://doi.org/10.1007/s42979-021-00852-w

ORIGINAL RESEARCH

Adaptive Resource Provisioning and Scheduling Algorithm for Scientific Workflows on IaaS Cloud

P. Rajasekar[1] · Yogesh Palanichamy[1]

Received: 2 March 2021 / Accepted: 6 September 2021 / Published online: 15 September 2021

Abstract
Scientific workflow applications are deployed to run extensive volumes of data and to manage comprehensive observations and simulations. They are resource-intensive and time-utilizing applications that profit from processing in distributed environments. Especially, workflow applications can highly support the simple access, scalability, and affordability provided by cloud computing. To attain this, disruptive and well-planned operation of managing the workflow tasks and running the compute pool in a cost-effective mode essential to be evolved. We present an adaptive resource provisioning and scheduling algorithm for scientific workflows on Infrastructure as a Service (IaaS) clouds. Our approach was planned to deal challenges especially to clouds, such as unlimited on-demand access, heterogeneity, performance variation and pay-per-use type (i.e., per minute billing). To correspondingly efficient to these uncertainties of cloud, therefore, our approach was developed with consider these necessary problems to run, and is achievable in making effective solutions that satisfy a user-described deadline and reduce the spending cost of the utilized environment using the service of resource provisioning and scheduling. Finally, experimental results show that it executes a workflow effectively with regard to achieving deadline and minimizing cost than other advanced algorithms.

Keywords Resource provisioning · Scheduling · Scientific workflows · And IaaS cloud

Introduction

Scientific workflows are described as computational tasks set and control or data-dependencies set among them. They are extensively utilized by the scientific groups to examine and run extensive volumes of data effectively. These huge scientific workflows are resource comprehensive applications and therefore are generally used on distributed environments. Scheduling approaches perform a significant role in processing workflows adequately because they are important for the management of the tasks on the distributed resource pool. Their findings are led by a set of Quality of Service (QoS) specifications described by the users' of application, such as reducing the makespan (complete execution time) and total cost, or satisfying a user-defined deadline or budget. This non-trivial problem of scheduling, in actually, is a recognized NP-Complete problem [1] and hence, algorithms should prioritize on discovering optimal solution in an acceptable period of time.

Infrastructure as a Service (IaaS) clouds provides a simply accessible, scalable and adaptable infrastructure for the utilization of wide-ranging scientific workflows. They admit users' opportunity to use a shared resource infrastructure on demand as charging just for what they utilize. This is achieved by renting Virtual Machines (VMs) or virtualized system resources, with a pre-described amount of CPU, storage, memory, and network bandwidth capacity. Various compute resource sets (i.e., VM sets) are accessible to consumers at various costs to fit a large-scale of application demands. Apart from VMs, IaaS vendors also provide storage and network systems to transfer the data in/out from/to the storage system. To completely take benefit of these opportunities and services, scheduling algorithms should examine some significant attributes of clouds.

The primary attribute is the on-demand, adaptable resource framework. This attribute recommends a

✉ P. Rajasekar
rajasekar@auist.net

Yogesh Palanichamy
yogesh@annauniv.edu

1 Department of Information Science and Technology, College of Engineering, Anna University, Guindy, Chennai, Tamil Nadu, India

re-development of the scheduling problem as usually described for existing distributed environments, such as cluster and grids. Clouds do not provide a limited range of computing resources, alternatively, they provide a virtually infinite set of resources with different configurations to be rented and utilized just for if they are demanded. This framework generates the requirement for a resource provisioning heuristic that runs concurrently with scheduling algorithm; a strategy that finds not only the number and the type of VMs to demand from the cloud but also determines when is the right time to rent and deliver them. Since this objective is customized for cloud infrastructure, from now on, the term scheduling will be taken to mention to an algorithm efficient of integrating both scheduling and resource provisioning findings (solution).

Another attribute to examine is the profitability-based pricing type employed by cloud vendors. The fee of utilizing the infrastructure demands to be examined or else, users probably paying unreasonable and excessive cost. For instance, the number of VMs rented, their type and total amount of unit they are utilized for, all contain a significant impact on the overall cost of executing the workflow in the cloud. As a result, schedulers demand to discover a trade-off among total execution cost and total execution time (makespan).

A third attribute of clouds is their dynamic condition and the environmental uncertainties this makes with it. An instance is the unpredictability in performance demonstrated by VMs with regard to processing times [2]. This unpredictability represents that in spite of a VM prototype being announced officially to have a particular CPU amount, it will probably execute at a less utilized amount that will vary overtime. It also represents that two homogenous VMs (same type) may give entirely various performances. Moreover, having many one or more users dividing a network represents that performance variation is also noticed in networking systems [2]. Now another origin of unpredictability are the delay of resource provisioning and de-provisioning; at are no promises on these values of provisioning and de-provisioning delay and they could be extremely unpredictable and variable [3]. Knowing performance unpredictability is crucial for schedulers so that they can execute a workflow efficiently from unpredicted delays and satisfy the requirements of QoS. Finally, we taken pay per minute billing for reducing the overall execution cost.

The contribution of this paper can be explained as follows:

- Consequently of these attributes, we present an Adaptive Resource Provisioning and Scheduling (ARPS) algorithm for scientific workflows on IaaS cloud.
- Our output discovers a balance among generating dynamic findings to respond to variations in the environment and focusing forever to make high-standard schedules.
- It goals to reduce the complete utilized infrastructure spending cost as satisfying a user-described deadline.
- It is efficient in determining what compute systems to utilize examining heterogeneous VM categories, when the appropriate time to rent them and when they must be terminated to prevent unnecessary expenses.
- Finally, our experimental results show it is scalable with respect to number of tasks in the scientific workflow, it is fit and flexible to the cloud performance unpredictability and it is efficient of making effective solutions than the other advanced algorithms.

Related Work

There have been some assignments as the emergence of cloud computing that targets to systematically organize scientific workflows. Many are powerful and efficient in adjusting to changes in the framework. A case is the algorithm of Dynamic Provisioning Dynamic Scheduling (DPDS) [4] in which the set of VMs is adapted requiring on how appropriately they are being utilized by the application. This study [5] also offers a dynamic process developed to represent the dynamic characteristic of cloud frameworks from the implementation and billing point of perspective. Planned a fault-tolerant effective algorithm formed on the partial critical paths of workflows [6]. The Partitioned Balanced Time Scheduling algorithm [7] calculates the suitable number of resources requested per billing slot so that the cost is reduced and a deadline is satisfied. Other dynamic approaches incorporate those presented by [8–21]. The main limitation of these techniques is their task-level optimal scenario, which is a trade-off for their adjustability to unpredicted delays.

Other characteristic of the scope are static approaches. An instance is the algorithm of Static Provisioning Static Scheduling (SPSS) [4] and Co-evolutionary Genetic Algorithm (CGA) [22]. Planned to schedule a set of interlinked workflows (i.e., ensembles), it generates a scheduling and provisioning strategy prior to processing any task. An additional case is the IaaS Cloud Partial Critical Path (IC-PCP) algorithm [23]. It is formed on the partial critical paths of workflows and attempts to reduce the total execution cost as satisfying a time constraint. Other specifications consider [24–28]. Usually, these strategies are extremely sensitive to performance delays and processing time calculation of tasks, which is a trade-off for their potential to execute workflow-level optimization and consider different solutions prior to deciding the most effective one. This work offers a dynamic approach developed to schedule a workflow on on-demand and spot instances [29]. To attain this, they have techniques in place to adopt spot and on-demand instance. However, most of the time spot price instances are not reliable and switch to on-demand instance latter to run the workflow for satisfying the deadline constraint.

Irrelevant to completely static or dynamic approaches, our objective integrates both with the purpose of discover an effective solution between changeability and the advantages of global optimization. Dyna [5] and SCS [30] are an instance of an algorithm trying to succeed this. It possesses a global optimization strategy that permits it to discover the suitable mapping of task to VM category. This mapping is then taken at processing time to scale the computing systems in or out and to organize tasks while they turn proceed for execution. Our plan is varied to Dyna and SCS because the static module does not study the whole workflow form and alternatively optimizes the organization of a subgroup of the workflow tasks. Furthermore, our static plan creates a usual schedule for these tasks instead of only choosing VM instance.

Application and its Resource model

We examine workflows represented as Directed Acyclic Graph (DAGs); that is, DAG with leading edges and no conditional dependencies. Generally, a workflow W is made up of a group of tasks $T=\{t_1, t_2,\ldots,t_n\}$ plus a group of leading edges E. A leading edge $e_{ij}=(t_i,t_j)$ is present if it is a dependency of data among task t_i and t_j, instance in which task t_i is implied to be parent of t_j and task t_j is implied to be child of t_i. As a result, a child task that cannot process up to all of its parents tasks has finished and its input file is accessible in the relative compute system. In addition to, a workflow is connected with a deadline D_w, described as a time restriction. Moreover, we believe that the scale of a task S_t (task size) is computable in Million of Instructions (MIs) and that, for each task, this fact is offered as information to scheduler.

A pay-per-use framework where VMs are rented whenever required and are billed per billing slot is tested. Some incomplete utilization happens in the machine use being charged as full-time utilization. We represent VM category, VMT, with respect to its processing capability PC_{vmt} and its charge per billing slot C_{vmt}. We describe PC_{vmt} with respect to the unit of instructions the CPU can run per second, Million of Instructions Per Second (MIPS). It is believed that for every type of VM, its processing capability in MIPS can be evaluated formed on the facts provided by vendors.

Scientific workflows run data in the structure of files. A general method taken to divide these files between tasks is to deploy peer-2-peer (P2P) framework in which files are transported immediately from the VM processing the parent task to the VM processing the children task. An additional approach is to take a general storage in sharing mode, such as Amazon s3 as a file cache. In this instance, tasks keep their output in the general storage and recover their inputs from the general storage as similar. We investigate the latter framework developed on the benefits it provides. Initially, the stored data are kept at it and therefore, can be taken for retrieval in the event of failure. Second, it enables for computation in asynchronous. In the P2P fashion, synchronous communication among tasks represent that VMs should be continued to be processing up to all of the child tasks have obtained the relative data. With a storage in shared on the opposite, the VM processing the parent task can be transferred immediately as the data are kept in the storage resource. This could possibly not just maximize the resource utilization, but also minimize the charge of VM renting.

We believe data transfer from/to the general storage system is in the house, as is the instance for commodity like Rackspace Block Storage, Amazon S3, and Google Cloud Storage. Like for the usual data storage, many cloud vendors amount formed on the volume of data being saved. We do not consider this fee in the overall implementation cost, neither our performance and nor the performance of the algorithms deployed for differentiated in the evaluations. The motivation for this can consider our technique with others developed to transport files in a P2P form. Moreover, despite the procedure, the stored amount of data for a considered workflow is most similarly correspondent in every condition that it does not outcome in a variability in cost.

We respect the fact of VM de-provisioning and provisioning delay and believe that CPU capacity of VMs is not consistent [2]. As opposed to, it differs over time with its most obtainable value being the CPU power promoted by vendor. Also, we believe network congestion makes a variance in data transport times [31]. The bandwidth allocated to a transportation relies on the current variance for the network connection being utilized. Also, we believe a general storage with an infinite storage room. The speed at which it is efficient of writing and reading data differs according to the number of processes presently reading or writing data from the storage system. Lastly, the running time of task t on a VM type VMT, PT_t^{vmt} is described as the total of its processing time and the time it needs to scan the input facts from the general storage and write the created output to the general storage. Account that at whatever time a parent and a children's task are performing in the corresponding machine, there is no essential to retrieve the child's input data from the storage system.

According to the aforementioned explanations, the problem would be officially described as proceeds: discover a schedule S with less EC (Execution cost) while ensure its associated M (Makespan) does not run over the deadline is shown in Eq. (1).

$$\begin{aligned}&\text{Minimize EC}\\&\text{Constraint to } M \le D_{\mathrm{w}}.\end{aligned} \tag{1}$$

ARPS Algorithm

Motivation

ARPS has dynamic and static characteristics. Its dynamicity exists in the factor that the scheduling findings are generated at processing time, whenever tasks are delivered into a processing queue. This admits it to adjust to unpredicted delays made by faulty estimates or by environmental uncertainties, such as delays in VM provisioning, network congestion and performance variation. The static module extends the potentiality of the algorithm from generating decisions formed on a single task to generating decisions formed on a set of tasks. The aim is to discover a trade-off among the local fact of dynamic procedures and global fact of static procedures. This is done by introducing the idea of pipeline and by consistently scheduling entire tasks in the processing queue at only one time. In this manner, APRS is apt to generate suitable optimization findings and discover high-standard schedules.

A pipeline is a general topological form in scientific workflows and is commonly a set of tasks with 1-to-1 sequential correspondence among them. Generally, a pipeline P is described as a group of tasks $T_{\mathrm{p}} = \{t_1, t_2, \ldots, t_{\mathrm{m}}\}$ where $m \ge 2$ and at is an edge $e_{c}, _{c+1}$ among task t_c and task t_{c+1}. In different words, t_1 is the parent task of t_2, t_2 is the parent task of t_3 and so forth. The initial task in a pipeline probably has other than one parent but it should only have one-child task. All upcoming tasks should have one parent (former task of pipeline) and one child (later task of pipeline). A pipeline is connected with a deadline D_{p} which is similar to the deadline of the final task in the series. For example, is depicted in Fig. 1a.

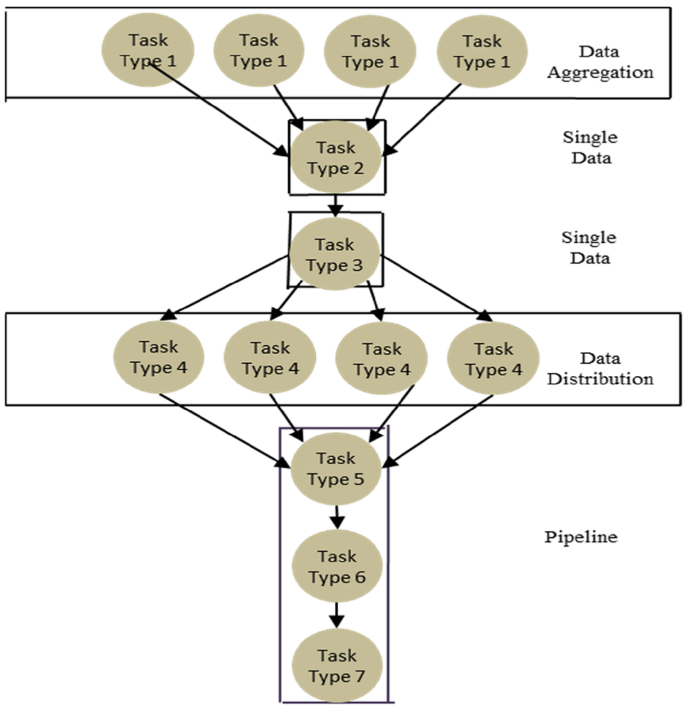
By finding pipelines in a scientific workflow, we can simply extend the perspective from one task to a group of tasks that could be scheduled more effectively as a set rather than on their characteristic. To ignore processing and communication overheads and also the VM provisioning and shutdown delays, tasks in a pipeline are integrated simultaneously and are consistently allocated to execute on the same VM. The explanations are twofold. First one are tasks which are sequential and in demand to execute one by one. There is no advantage with respect to parallelization on allocating them various VMs. Second one is parent task output file becomes the input file of the child task by processing on the same VM, we ignore the cost and the transferring time of data files out/in from/to the general storage.

The heuristics taken to schedule organized tasks are originated from the topological structures of workflows. Apart from pipelines, a scientific workflow also has concurrent forms made up of tasks without dependencies among them. These tasks can process concurrently and are commonly based whenever data aggregation or distribution occurs. In the strategy of data distribution [32], the output of a task is assigned to one or more tasks for running. In the strategy of data aggregation [32], the output of various tasks is combined, or organized, by only one task. Figure 1a depicts these structures for an example.

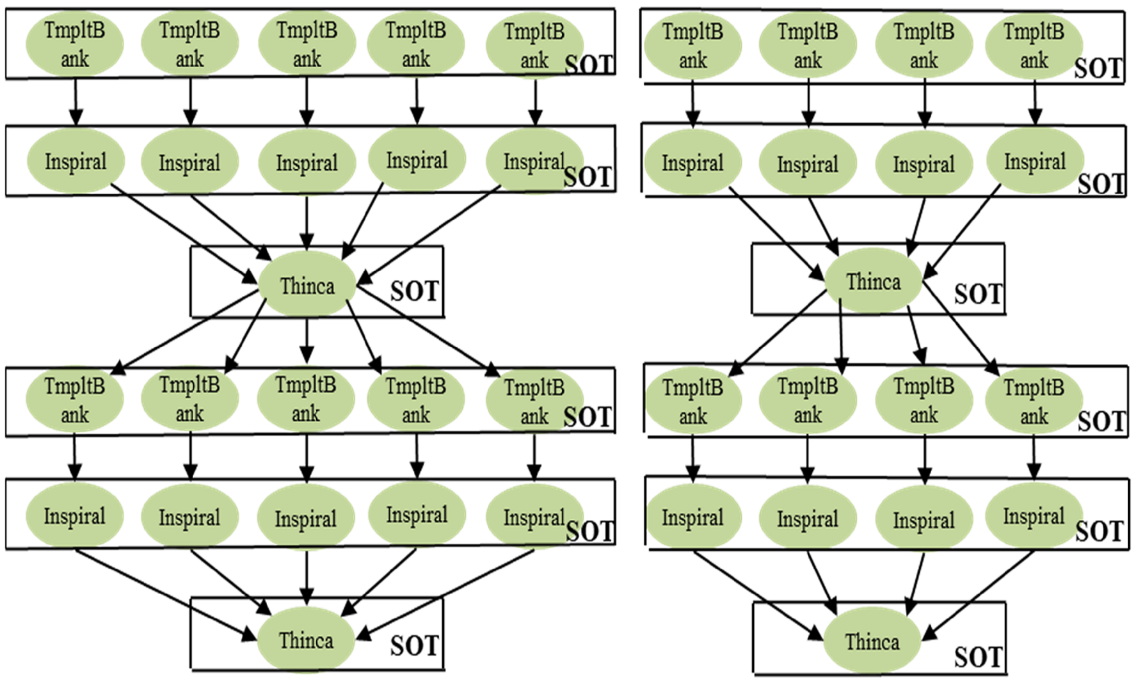
The concurrent tasks in these workflow framework can be homogeneous tasks. The situation in which the tasks are uniform (homogeneous) is usual in workflows; examples of familiar applications with this features are LIGO, Montage, Epigenomics and CyberShake. Formed on this, we conceive a plan to effectively schedule homogeneous concurrent tasks that are of the equivalent size (MIs) and are at the concurrent level in the structure. Once taking a level-based deadline allocation strategy, these concurrent tasks will also admit the very same deadline. As an instance, investigate the data aggregation situation, all concurrent tasks have to complete processing prior to the aggregation task can begin; hence, they would be allocated the very same time constraint which would be same to the time the aggregation task is expected to start. In addition, there are other facts apart from aggregation and distribution where the concurrent tasks with the similar attributes can be discovered, anyway, we prioritize on these as representatives for demonstrating the reason behind our scheduling scheme.

The significant static scheduling approach of APRS comprises then on clustering queued tasks of homogeneous and with the equivalent deadline into the sets. Two typical sets can be noticed in Fig. 1a, the first set is composed of entire tasks of Type 1 and second set is composed of entire tasks of Type 4. Scheduling these sets of tasks is very simpler than scheduling entire workflow. There are no inter-relationships, the tasks are same type and have to complete at the equivalent time. We represent the problem of processing these tasks within their deadline and with lesser cost as an unrestrained knapsack problem variation and discover a better solution taking dynamic programming. The similar idea is followed to pipelines, they are clustered into sets and planned in the similar way as sets of tasks are. An instance of a set of pipelines is shown in Fig. 1c, d.

We have thus developed ARPS which is dynamic to a particular extent with the aim of adjusting to unpredicted delays consequence of the uncertainties of cloud frameworks, but that also keeps static features that make it to produce high-standard schedules and satisfy deadline at lesser costs. Furthermore, it integrates a strategy-based technique with dynamic programming as to be able to run extensive workflows in a scalable and efficient manner.



(a)



(b)

Fig. 1 Examples of scientific workflows. **a** Examples of set of tasks and three different topological structures discovered in workflows: data aggregation, data distribution and pipelines. **b** LIGO workflow. **c** Montage workflow. **d** Epigenomics workflow. **e** CyberShake workflow

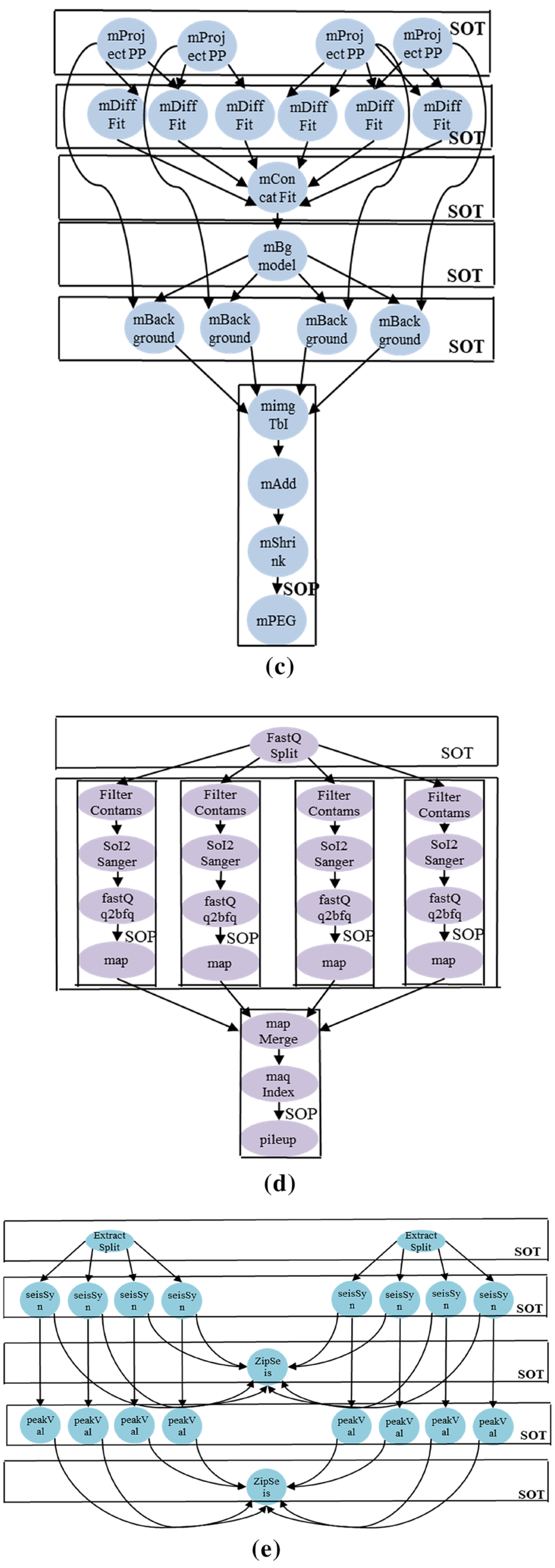


Fig. 1 (continued)

Unrestrained Knapsack Problem

The unrestrained knapsack problem is a combinatorial optimization of NP-hard problem that obtains from the problem of choosing the most profitable objects to load into limited weight knapsack. Considered n objects of various types, each type of object $1 \le i \le n$ with an equivalent weight w_i and value v_i, the aim is to find the type and number of objects to load; hence, that the knapsack weight range W is not overreached and the entire value of the objects is maximized. Unbound number of each object type is believed.

Let $x_i \ge 0$ be the group of objects of type i to be filled in the knapsack set. Then, UKP would be described as (2).

$$\text{Maximize} \sum_{i=1}^{n} v_i x_i \quad \text{Subject to} \sum_{i=1}^{n} w_i x_i \le W. \tag{2}$$

This problem could be efficiently solved employing dynamic programming by examining knapsack of lesser spaces as subproblems and saving the optimal value for each space. Let $w_i > 0$, therefore a vector could be described, where $m[w_i]$ is the value of maximum that could be acquired with a weight equal or less than to weight w_i. In this manner, $m[0] = 0$ and $(m[w_i] = \max_{w_j \le w_i} (v_J + \mathrm{m}[w_i - w_j])$. The solution time complexity is $O(nW)$ as estimating every $m[w_i]$ comprises considering n objects and there are W values of $m[w_i]$ to estimate. This processing time is pseudo-polynomial as it evolves exponentially with the range of W. Still, there are some algorithms that can effectively work out UKP. For example, EDUK [33] algorithm which integrates the ideas of monotonic recurrence [34], periodicity [35] and dominance [36]. Evaluations conducted by the correspondents show its ability to be changed in size. For example, for $W > 2 \times 10^8$, $n = 10^5$, and the objects with weight in the [1, 10^5] limit, the processing time average was discovered to be 0.150 s.

Algorithm

APRS initially pre-processes the workflow by finding the pipelines and by allocating an amount of the deadline D_w to every individual task. To identify the pipelines, tasks are initially clustered in topological structure, in this manner, we assure data-dependencies are secured. Then, pipelines are created according to the following plan. For each clustered tasks that has not been executed, the ARPS recursively attempts to create a pipeline that begins with that task. The fundamental logic of the recursion occurs when the running task has no child, or when it has beyond one child, or when it has a child with beyond one parent. The recursive process happens when the running task has precisely one child which all together has exactly one parent task. In this condition, task is attached to the pipeline and the recursion keeps running with its child task. When a pipeline was found and the recursion completes, the approach is continued for the following unexecuted clustered task, this runs on up to there is no clustered task remaining to be processed. A further thorough explanation of the recursive stage of the process is shown in Algorithm 1.

Algorithm 1
Identify a Pipeline Repeatedly

```
1.  Process IDENTIFYPIPELINE(Task t, Pipeline P)
2.      if t.child.size > 1 OR t.child.size = 0 OR
3.         t.child[0].parentssize > 1 then
4.         if p.tasks.size > 0 then
5.                p.addTask(t)
6.         end if
7.         return
8.      end if
9.      p.addTask(t)
10.        identifyPipeline(t.child[0],p)
11. end process
```

For the distribution of deadline, the ARPS initially estimates the earliest finish time of each task described as $\text{eft}_t = \max_{p \in t.\text{parents}}\{\text{eft}_p\} + \text{PT}_t^{\text{vmt}}$. The slowest type of VMT is utilized to estimate the running time of a task. In this manner, they can just enhance if various types of VM are taken. Anyway, if taking the slowest type of VM represents not being capable to satisfy the deadline, subsequently the next speediest VM model is taken to calculate processing times and so forth. Next, the extra time, described as the gap among the deadline and earliest finish time of the scientific workflow ($D_w = \max_{t \in W}\{\text{eft}_p\}$) is estimated and shared among the workflow group formed on the number of tasks they keep. At last, each task is allocated its deadline $D_t = \max_{p \in t.\text{parents}}\{D_p\} + \text{PT}_t^{\text{vmt}} + t.\text{level.extra}$.

When a workflow is pre-processed, the task scheduling happens. In the initial iteration, all the arrival tasks (those without parent tasks) turn prepared for processing and are put in a scheduling order. These tasks are scheduled and in a while they complete their processing, their child tasks are delivered onto the scheduling order. This approach is continued until whole workflow tasks have been executed successfully. To schedule the tasks in the order, tasks are initially clustered into sets of tasks and sets of pipelines. A set of tasks *sot* is described as a set of one or multiple tasks T_{sot} that can execute in concurrent. The total number of tasks in a set divides the same deadline D_{sot}, is of the homogeneous T_{sot} and is not role of a pipeline. Generally, $\text{sot} = (T_{\text{sot}}, D_{\text{sot}}, T_{\text{sot}})$. The description of set of pipelines *sop* is same but in place of a set of tasks, the set holds one or multiple pipelines P_{sop} that are in parallel. The element $\text{sop} = (P_{\text{sop}}, D_{sop})$, where ($D_{\text{sop}}$) is the deadline limit the pipelines in the set keep in general, usually defines the plan.

To discover the bags of sets of tasks $\text{SoT} = \{\text{sot}_1,\ldots,\text{sot}_n\}$ and sets of pipelines $\text{SoP} = \{\text{sop}_1,\ldots,\text{sop}_n\}$, every task in the order is executed in the following manner. If a task does not associate to a pipeline, then it is put into sot_i that includes tasks of the homogeneous and with the equivalent deadline. If no correspondent sot_i is present, a fresh set is generated and the task allocated to it. If, on the other side, the task associates to a pipeline, the parallel pipeline is put in the sop_i which includes pipelines with the same types of tasks and deadline. If there is no sop_i with this attributes, a fresh set is generated with its only component being the considered deadline.

When the bags of SoP and SoT are generated, we proceed to map them. Both categories of sets are scheduled taking the related plan, with the slight variation being the pipeline tasks should be handled as a component. We define the strategy taking SoT, consider anyway that the corresponding rules follow when scheduling SoP. To schedule SoT, we rerun the following strategy for every set $\text{sot}_i \in \text{SoT}$ that has only one task (sets with a single task are managed as specific type and scheduled correspondingly). Initially, APRS attempts to minimize the size of the set and reutilize earlier rented VMs by allocating a bunch of tasks as achievable to non-busy VMs. The number of tasks scheduled to an accessible VMs is decided by the number of tasks that can complete prior to their deadline and prior to the next billing slot of the VM. In this manner, wastage of earlier charged CPU cycles is minimized without disturbing the workflow makespan. Then, a plan of resource provisioning is generated for the rest of the tasks in the set.

To create a cost-effective resource provisioning strategy, APRS should search various solutions taking various VM models and consider their related costs. We attain this and discover the suitable consolidation of VMs that can complete the tasks in the set within the time with less cost by stating the problem as UKP variation and work it out taking dynamic programming. An object of knapsack is described by its category, load, and value. For our job, we describe a scheduling object of knapsack $\mathrm{SKI}_J = (\mathrm{VMT}_J, \mathrm{NT}_J, C_J)$ where the type of object relates to a type of VM VMT_j, the load is the maximal number of tasks NT_j that can process in a type of VM NT_j prior to their deadline, and the associated price C_j of processing NT_j tasks in a type of VM VMT_j. Moreover, we believe there is an infinite number of VMs of every type that could be possibly rented and describe the weight limit of knapsack as the number of tasks in the set, that is, $W = |T_{\mathrm{sot}}|$. The objective is to discover a bag of objects *SKI*; hence, that their integrated weight (the total number of tasks) is at minimum as huge as the weight limit of knapsack (the number of tasks in the set) and whose integrated value is lesser (the cost of executing the tasks is lesser). Officially, a set of tasks scheduling problem is written as (3).

$$\text{Minimize} \sum_{i=1}^{n} C_i \times x_i \quad \text{Subject to} \sum_{i=1}^{n} \mathrm{NT}_i \times x_i \geq |T_{\mathrm{sot}}|. \quad (3)$$

After sorting out the problem of UKP, resource provisioning $\mathrm{RP}_i^{\mathrm{sot}} = (\mathrm{VMT}_i, \mathrm{numVM}_i, \mathrm{NT}_i)$ is acquired for each type of VM. It represents the amount of VMs of category VMT_i to utilize numVM_i and determined number of tasks to process on every VM (NT_i). In special condition in which no VM categories that can complete the tasks within the time, a provisioning scheme of the pattern $\mathrm{RP}_{\mathrm{sot}}^{\mathrm{fastest}} = (\mathrm{VMT}^{\mathrm{fastest}}, W, 1)$ is generated. This represents that VM of the quickest instance should be rented for every task in the set; hence, they can process in concurrent and complete as soon as achievable. Afterwards, for every $\mathrm{RP}_i^{\mathrm{sot}}$ for which $\mathrm{numVM}_i > 0$, APRS attempts to identify a type of VM VMT_i, which has been rented early and is available to utilize. If it presents, then NT_i tasks from the set are mapped on to it. In this manner, we minimize cost utilizing already billed for time periods and ignore newly rented VMs high provisioning delays. If it is not accessible VM of the demanded type, then a new VM is provided and NT_i tasks from the set are mapped on to it.

We investigate the condition of a set with a single or only one task as a specific case. Scheduled single tasks on available VMs if they can complete the task within the time and prior to their present billing slot completes. If there is no available VM that can attain this, then a new type of VM that is efficient in completing the task by its time limit at the lowest cost is provided and the task mapped to it. If no such type of VM can complete by its time limit, the accessible type of fastest VM is taken. The SoT scheduling pseudocode is depicted in Algorithm 2.

Algorithm 2
SoT Scheduling

```
1.  Process ScheduleSoT(SoT)
2.      for all sot ∈ SoT do
3.        if all sot.tasks.size > 1 then
4.          RP^sot = UKPFormedProvisioningStrategy(sot)
5.          for all $RP_i^{sot} = (VMT_i, num_i, Nt_i) \in RP^{sot}$ do
6.            for k=0; k<numVM_i; K++ do
7.              n tasks = min{Nt_i, sot.size}
8.              tasks = {t_1,...t_nTasks|t_i ∈ sot.tasks}
9.              remove tasks from sot.tasks
10.             vm = identifyFreeVM(VMT_i)
11.             if vm == null then
12.             vm = leaseNewVM(VMT)
13.             end if
14.                scheduleTasks(task, vm)
15.           end for
16.         end for
17.       else
18.         task = sot.tasks[0]
19.         vm = identifyfreeVMForTask(task, deadline),
20.         if vm ==null then
21.            vm = leaseNEWVM(VMT_i)
22.         end if
23.            scheduleTasks(task, vm)
24.       end if
25.     end for
26. end process
```

ARPS repeatedly adapts the deadline allocation to reflect the usual complete time of tasks. If a task completes before than predicted, all of the leftover tasks will have longer time to process and cost can be possibly minimized. Once a task completes longer than predicted, the deadline of all leftover tasks is improvised to ignore delays that would guide to the complete deadline being violated. ARPS also has a mechanism of rescheduling that admits it to manage with unpredicted delays occurred while processing a set of tasks (or pipelines). As many tasks are consistently allocated to a VM, a delay in the performance of a single task will have an influence on the expected completing time of the upcoming tasks. To reduce this influence, once a task associating to a set completes after its time limit on VM_i, then the tasks in the processing order of VM_i are examined in the following manner. If rest of the tasks in the order of VM_i can complete by their improvised deadline, then no activity is considered. If VM_i cannot complete its assigned tasks within their time limit, then the tasks are delivered return into the scheduling order; hence, they can be re-mapped again according to their improvised deadline.

As stated before, the bag of sets of *pipelines SoP is scheduled taking the same procedure taken for SoT*. Simply as tasks, pipelines contain a deadline and a size. Hence, we can follow the same scheduling procedure and represent the challenge as a UKP variation with a little bit difference in the description of knapsack object. For pipelines, $\mathrm{SKI}_j = (\mathrm{VMT}_j, \mathrm{NP}_j, \mathrm{C}_j)$, where an object's weight NP_j, is similar to the determined number of pipelines a VM model can complete prior to their deadline. The remapping heuristic is also same to that of tasks, other than that whenever a task in a pipeline is wait longer, the rest of the pipeline tasks are remained to complete in the VM as time-ahead pipelines are remapped form on their improvised deadline. When again, sets with only one pipeline are handled as a specific type and are allocated to available VMs if they can complete them within their time limit or rented new type of VM that can complete them within their time limit at less cost.

Lastly, ARPS switches off VM if its utilization is nearing the next charging slot and if no tasks allocated to it. A calculation of the delay of VM de-provisioning is taken to secure the VM switch-off request is forwarded before enough, hence that it ends being charged prior to the ends of current billing slot.

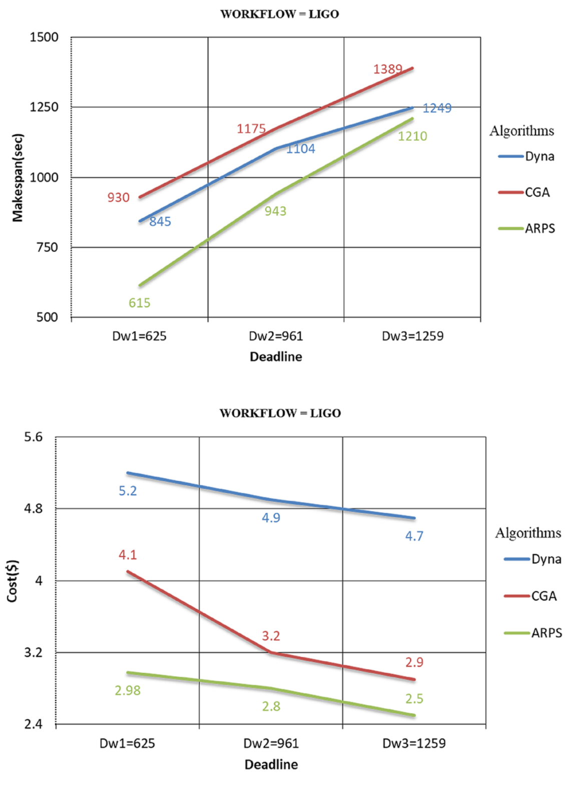
Experimental Results

We used the CloudSim framework [37] to set up the cloud platform in which APRS was estimated by four well-known scientific workflows from many scientific fields: LIGO associating to astrophysics field, Montage associating to astronomy field, Epigenomics associating to bioinformatics field, and Cybershake associating to physics field. The LIGO, Montage, Epigenomics and CyberShake are seen in Fig. 1b–e. Each workflow possesses various topological frameworks, various data and computational features. Their characterization and description is given by [32].

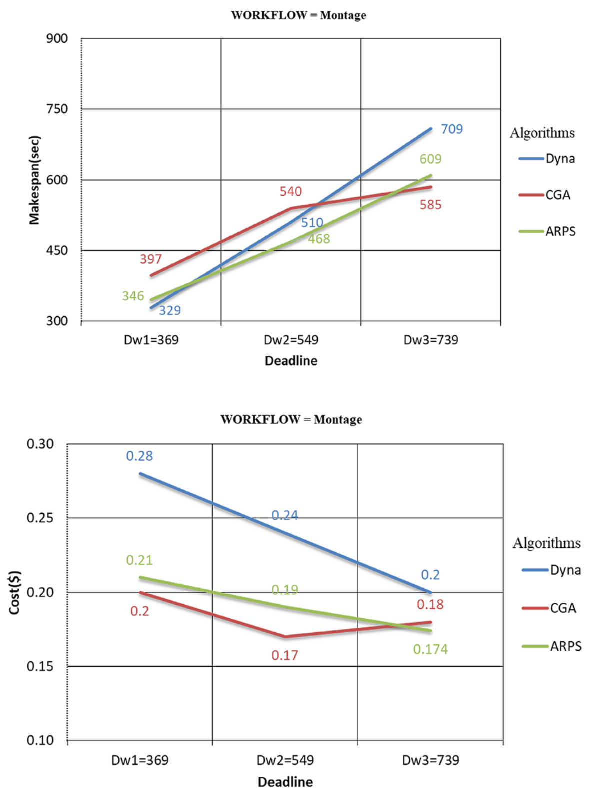
Two approaches were utilized to estimate the standard of the schedules made by ARPS. The first algorithm is Co-evolutionary Genetic Algorithm (CGA) [22] which allots sub-deadlines to each task and executes them on to currently rented or existing VMs so that overall cost is reduced. It was selected because it is a static approach potential of making optimal solution. Its limitation is its insufficiency to satisfy deadlines when unpredicted delays happen. Anyway, we are more fascinated in comparing ARPS to CGA when both are capable of satisfying the deadline constraints. Moreover, by considering them, we are capable of verifying our result's adjustability and show how at what time other algorithms did not succeed to improve from unpredicted delays ARPS achieves while doing so.

Next algorithm is Dyna [5], an advanced dynamic approach that possesses an autoscaling feature that allots and de-allots VMs formed on the present status of tasks. It was developed to schedule multiple workflows even but it is able to be adjusted to schedule only one workflow at a time. It identifies the suitable cost-effective type of VM for each task by A star search. This algorithm is improvised through every interval of scheduling and represents the number of VMs of each category demanded with the aim of the tasks to complete by their time limit with less cost. The aim is to show how the ARPS static component admits it to make high-standard schedules than the algorithm of Dyna.

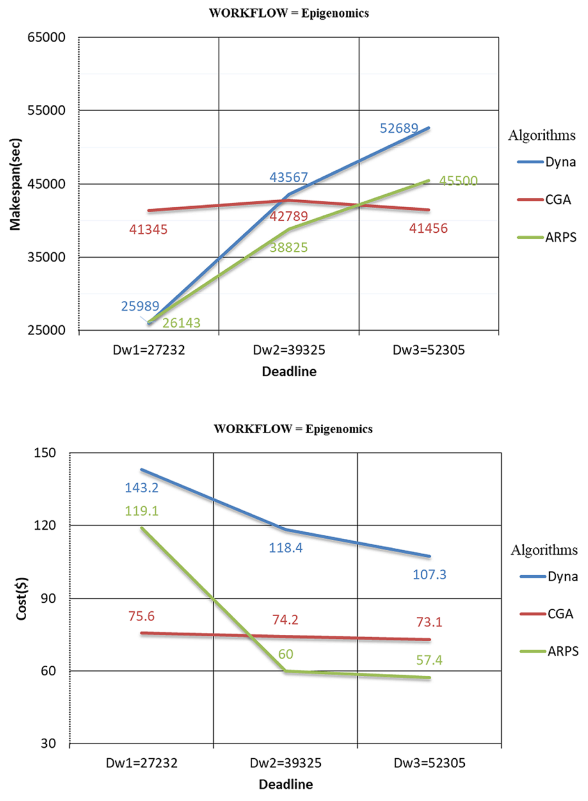
An IaaS vendor giving a solitary data region and four categories of VM was deployed. The configuration type of VM is formed on the offerings of Google Compute Engine and is presented in Table 1. A 60 s's billing slot was modeled, as given by vendors, such as Microsoft Azure and Google Compute Engine. For all types of VM, the delay of provisioning was given to thirty seconds [38] and shut down delay was assigned to three seconds [3]. Performance variation of CPU was used after the judgings by [2]. The VM performance is reduced at most 24% formed on the distribution of normal with a mean and standard deviation of 10% and 12%, respectively. A total available bandwidth of network connections is shared among entire transfers utilizing the network connection at a considerable particular moment. This allocation of bandwidth was achieved utilizing the algorithm of progressive filling [39] to set data transfer time and congestion degradation. A maximum speed of reading and writing of global shared storage was also set. The speed of reading possible by a considered transfer is found by the number of processes reading presently from the storage and the similar guideline using for the writing speed. In this manner, storage system congestion is simulated.


WORKFLOW = LIGO
Makespan(sec)
1500
1250
1000
750
500
930
1175
1389
845
1104
1249
615
943
1210
Dw1=625
Dw2=961
Dw3=1259
Deadline
Algorithms
Dyna
CGA
ARPS
WORKFLOW = LIGO
Cost($)
5.6
4.8
4
3.2
2.4
5.2
4.9
4.7
4.1
3.2
2.9
2.98
2.8
2.5
Dw1=625
Dw2=961
Dw3=1259
Deadline
Algorithms
Dyna
CGA
ARPS

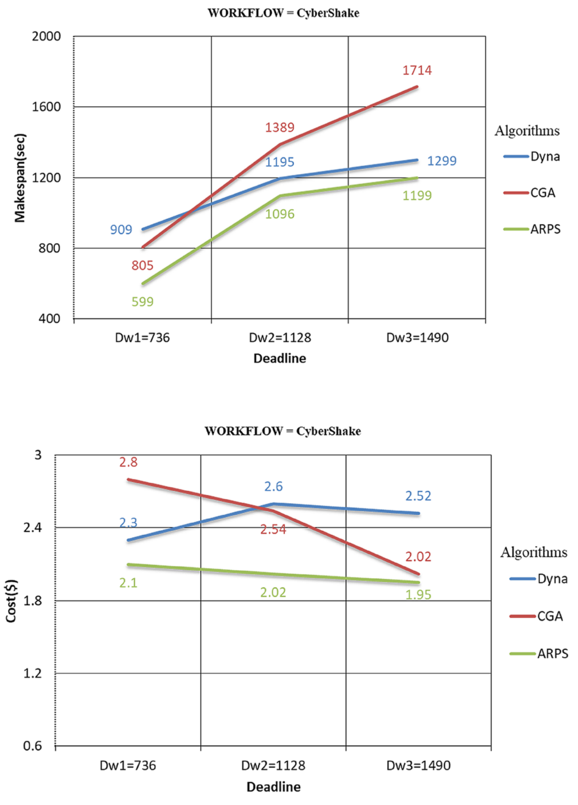
(a)


WORKFLOW = Montage
Makespan(sec)
900
750
600
450
300
397
540
585
329
510
709
346
468
609
Dw1=369
Dw2=549
Dw3=739
Deadline
Algorithms
Dyna
CGA
ARPS
WORKFLOW = Montage
Cost($)
0.30
0.25
0.20
0.15
0.10
0.28
0.24
0.2
0.21
0.19
0.174
0.2
0.17
0.18
Dw1=369
Dw2=549
Dw3=739
Deadline
Algorithms
Dyna
CGA
ARPS

(b)


WORKFLOW = Epigenomics
Makespan(sec)
65000
55000
45000
35000
25000
52689
43567
25989
45500
38825
26143
41345
42789
41456
Dw1=27232
Dw2=39325
Dw3=52305
Deadline
Algorithms
Dyna
CGA
ARPS
WORKFLOW = Epigenomics
Cost($)
150
120
90
60
30
143.2
118.4
107.3
119.1
60
57.4
75.6
74.2
73.1
Dw1=27232
Dw2=39325
Dw3=52305
Deadline
Algorithms
Dyna
CGA
ARPS

(c)


WORKFLOW = CyberShake
Makespan(sec)
2000
1600
1200
800
400
1714
1389
805
909
1195
1299
599
1096
1199
Dw1=736
Dw2=1128
Dw3=1490
Deadline
Algorithms
Dyna
CGA
ARPS
WORKFLOW = CyberShake
Cost($)
3
2.4
1.8
1.2
0.6
2.8
2.54
2.02
2.3
2.6
2.52
2.1
2.02
1.95
Dw1=736
Dw2=1128
Dw3=1490
Deadline
Algorithms
Dyna
CGA
ARPS

(d)

◂**Fig. 2** Evaluation of makespan and cost. The three column in the makespan graph represents the three deadline values and depicted its cost below to makespan graph. **a** LIGO workflow makespan and cost. **b** Montage workflow makespan and cost. **c** Epigenomics workflow makespan and cost. **d** CyberShake workflow makespan and cost

Workflows with about 1000 tasks were utilized for the estimation. We admit the fact that the task sizes estimation would not be 100% perfect and therefore, launching in our experiment a variation of ± 10% formed on uniform distribution to each task size. The experiments were evaluated using three different deadlines, D_{w_1} is the strictest one, D_{w_2} and D_{w_3} are the medium and relaxed one. For every workflow, D_{w_1} is same to the time it needs to run the tasks in the workflow path and the time it needs to transfer all I/O files from/to the storage. The leftover deadlines are formed on D_{w_1} and a size of interval $D_{\text{int}} = D_{w_1}/2 : D_{w_2} = D_{w_1} + D_{\text{int}}$ and $D_{w_3} = D_{w_2} + D_{\text{int}}$. The results shown are the mean acquired after executing every experiment 25 times.

Results and Analysis

Evaluation of Makespan and Cost: The obtained average makespan (total execution time) and cost for each workflow is shown in Fig. 2. The source lines in the line plots of makespan linking to the values of three deadlines utilized for every workflow. Estimating the makespan and cost in terms of this value is significant as the important goal of entire algorithms is to complete within the deadline.

In the case of LIGO, D_{w1} demonstrates to be too strict for each algorithm to complete within the deadline. Anyway, the comparison between the obtained makespan of ARPS and the deadline is very little. Moreover, ARPS makes the lesser cost schedules in this situation. The D_{w_2} deadline interval is yet not eased sufficient for the algorithm of Dyna or CGA to reach their target, anyway, ARPS shows its adjust-ability and potentiality to make less cost schedules by being an algorithm to complete its execution prior to the deadline and with the cheapest cost. For the leftover deadline, D_{w_3}, both Dyna and APRS are having the ability of satisfying the constraint, in case Dyna possesses a somewhat lesser makespan but ARPS possesses the cheapest cost. CGA is just potential of satisfying the deadline in relaxed deadline interval and in this situation, ARPS performs better than CGA with respect to overall cost. Generally, ARPS satisfies all deadlines and in all situations, succeeds high-standard schedules with the lesser cost.

For Montage, Dyna and ARPS satisfy each deadline with ARPS constantly making less cost schedules. CGA does not succeed to satisfy the strictest deadline but achieves in satisfying D_{w_2} and D_{w_3}. Its victory in satisfying two out of three deadlines would be defined in the information that the majority of the task in the application of montage is examined small and needs less utilization of CPU, guiding to a possible less amount of performance variation effect on the consistent schedule. In the instance of D_{w_2}, CGA demonstrates its capability to make less cost schedules and outperforms than CGA and ARPS; even though ARPS possesses a less makespan in this instance, its cost is somewhat larger than CGA. For D_{w_3}, anyway, ARPS succeeds in making less cost outputs than Dyna and CGA.

In the case of Epigenomics, CGA does not succeed to meet deadline in D_{w_1}, whereas deadline succeeded by ARPS, followed by Dyna and lastly, ARPS algorithm is only capable of satisfying D_{w_2} with less cost. The constraint of third deadline is achieved by CGA and ARPS, with ARPS perform better than CGA with respect to cost. Lastly, as the deadline turn relaxed sufficient, all three approaches achieved in satisfying the deadline and ARPS achieves it with less cost. The huge deadline fail percentage of CGA and Dyna in D_{w_2} is caused by high CPU consumption characteristics of the tasks of Epigenomics, representing that performance degradation of CPU will possess a crucial effect on the processing time of tasks making unpredicted delays.

In the case of Cybershake, ARPS achieves in satisfying all user-defined deadlines with less makespan and less cost than Dyna and CGA. The result show that even in situations like this, ARPS stays efficient and responsive. It achieves in satisfying all three deadlines with less makespan and cost amidst the algorithms that satisfies the constraint. The huge number of data that demand to be shifted when executing this workflow guides to CGA battling to improve from the rates of less data transfer because of congestion in network and hence not succeeding to satisfying the three deadlines. And also Dyna does not succeed to adjust to these delays within the deadline and not achieves to satisfy the three toughest deadlines.

Finally, ARPS algorithm is the most efficient in satisfying deadlines. In terms of mean, it achieves in satisfying the deadline constraint in cent percent of the instances, while Dyna achieves 45.833% and CGA achieves 12.5%. These outputs are in line with what was predicted of every algorithm. The static heuristic is not more effective in satisfying the deadlines, whereas adjust-ability in ARPS and Dyna permits them to achieve their goal frequently. The experiment results also show the ARPS efficiency with respect to its potentiality to make less cost schedules. It performs better than CGA and Dyna as in all cases, ARPS succeeds the cheapest cost when examined the difference to the algorithms of Dyna and CGA that satisfied the deadline. Another advantage of ARPS that would be noticed from the outputs is its capability to constantly increase the time it needs to execute the workflow. The significance of these dependencies in the information that multiple users are eager to trade-off execution time for lesser costs as others is eager to pay more cost for fastest execution. The algorithm demands to

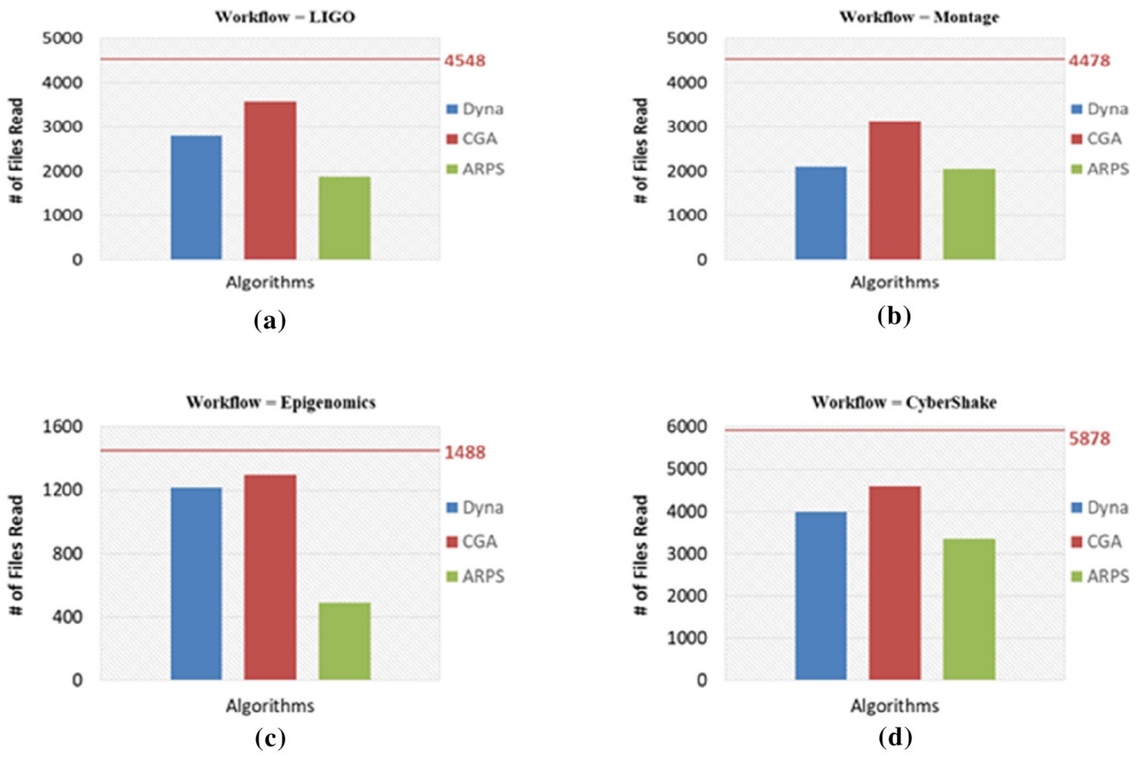


Fig. 3 Average number of files read from the global storage by each algorithm. The red line represents the total number of files required as input by the given workflow. **a** LIGO. **b** Montage. **c** Epigenomics. **d** CyberShake

act not beyond this logic with the aim of the value of deadline considered by users to be reasonable.

Network Use Estimation

Network connections are notable bottlenecks in IaaS cloud platforms. For example, address the variation of data transfer time up to sixty five percent in Ec2 cloud [31]. Therefore, as representatives of minimizing the origins of uncertainty and enhancing the execution of scientific workflow applications, it is crucial for the algorithms of scheduling to attempt to minimize the data transferred amount by the cloud network platform. In this portion, we estimate the unit of files read from the storage by each algorithm. Remember that a task does not demand to read from the general storage at whatever time the input data files, its needs are early accessible in the VM where it is executing.

The column chart in Fig. 3 demonstrate average amount of input data files reading over three deadlines for each scientific workflow and each algorithm. The source line represents the complete number of input data files that are demanded by the tasks in the workflow. By planning pipelines in only one VM and by executing as multiple tasks or pipelines from the identical set in only one VM, ARPS is achievable in minimizing 50% or further the amount of input data files read from the global storage. In particular, ARPS reads the minimum number of input files when differentiated to Dyna and CGA in the situations of the workflow of LIGO and Epigenomics. The number of input data files read from the global storage are minimized by 59% in LIGO workflow and 76% in Epigenomics workflow. For the application of montage workflow, Dyna and ARPS succeed with slight performance variation and minimize the amount of input files reading by more than 53%. Lastly, ARPS is achievable in minimizing the amount of input files reading than the algorithm of CGA and Dyna in CyberShake and the amount of files read in CyberShake from the global storage are minimized by around 47%.

Sensitivity of Provisioning Delay

The ARPS execution time is determined by the ability to satisfy the defined strict constraint of deadline under various provisioning delay; this is estimated by considering the workflows' deadline ratio. In this manner, ratio value less than one means ARPS execution time is within a deadline, ratio value is equal means ARPS execution time and deadline are equal and the ratio value is larger than one means a deadline violation. These procedures are followed to next section also with different values of performance variation.

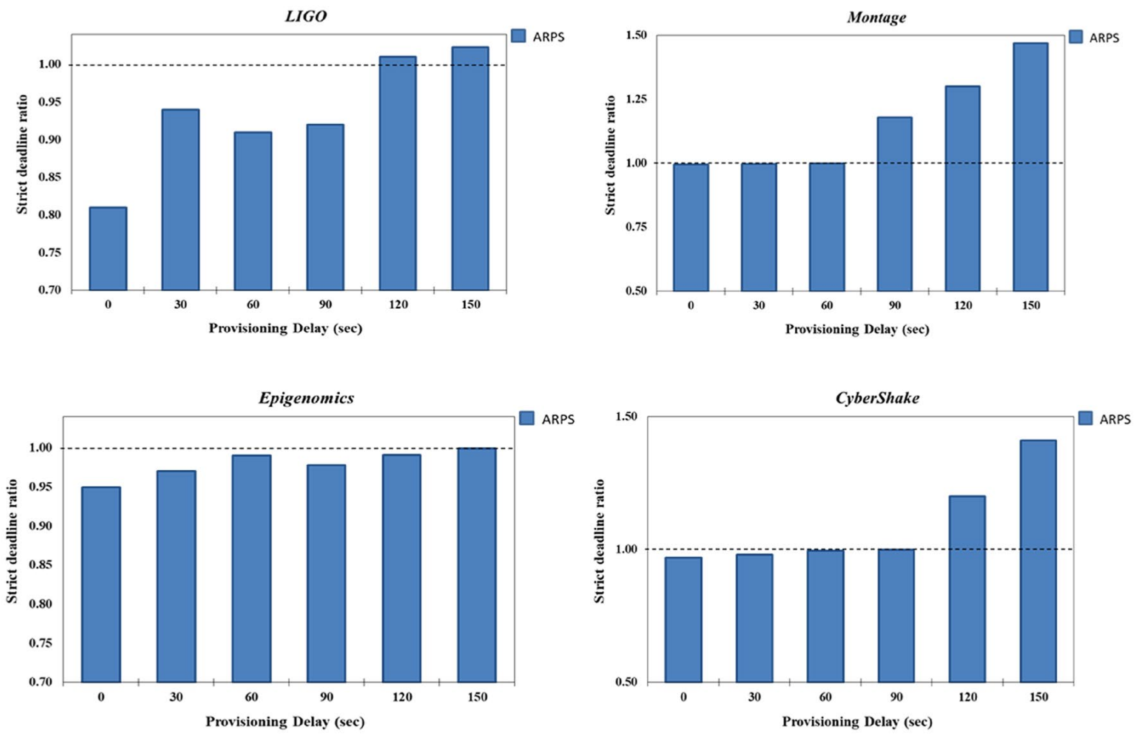


Fig. 4 Strict deadline ratio obtained for each workflow with different provisioning delay

The pricing model contributes frequent operations of VM provisioning and hence, it is significant to estimate the capability of ARPS ability to complete the execution of workflow no higher than the deadline given within different delays of VM provisioning. The delays were ranged from 0 to 150 s. Figure 4 depicts obtained strict deadline ratios for each workflow application considers across various provisioning delay.

For the application of Ligo, most of the ratio values under one for all delays of provisioning except last two values of 120 and 150. The obtained ratio values of these provisioning delay are slightly larger than one. This is because of the algorithm failed to satisfy the deadline, as it is too strict for it to be succeeded with such larger delays of provisioning. Finally, the Montage application shows maximum ratio values higher than one in last three scenarios except first three. This is made by too strict deadline for ARPS algorithm to complete on time despite the delays of provisioning. In the case of Epigenomics application, all data points of ratio are less than one and clearly showing the ability of ARPS algorithm to adapt in improving provisioning delays if the budget admits for it. For the application of CyberShake, last two ratio values are larger than one. This happening by the algorithm budget corresponds to be too tight and as the delay of provisioning increases the ratio values are also increased.

Sensitivity of Performance Variation

Knowing performance variation is significant for schedulers, hence they can improve from unpredicted delays and satisfy the requirements of QoS. The algorithm sensitivity to VM performance variation was analyzed by studying the strict deadline ratio within different values of variation in performance. It was done using normal distribution with different average values. The average values were specified as part of the maximum performance degradation of CPU ranges from 0 to 60%.

The obtained results are shown in Fig. 5. For the applications of Ligo and Epigenomics, all values are lesser than one except the last one of 60% degradation due to the tight budget of algorithm. For the Montage application, achieving the ratio values below one are 0%, 10%, 30% performance variation and violating the ratio values are 20%, 40% 50% and 60% degradation due to the strict deadline. Finally, the application of CyberShake, most ratio values are nearer to the one and exceeds the values of 50% and 60% performance degradation due to the deadline interval is too strict.

Another possible cause for violating the deadline constraint is the reason that ARPS generates a static provisioning method for SoTs and SoPs. Although this allows the algorithm to generate better decision of optimization to minimize the workflow's makespan, it also reduces its responsiveness to environment uncertainties.

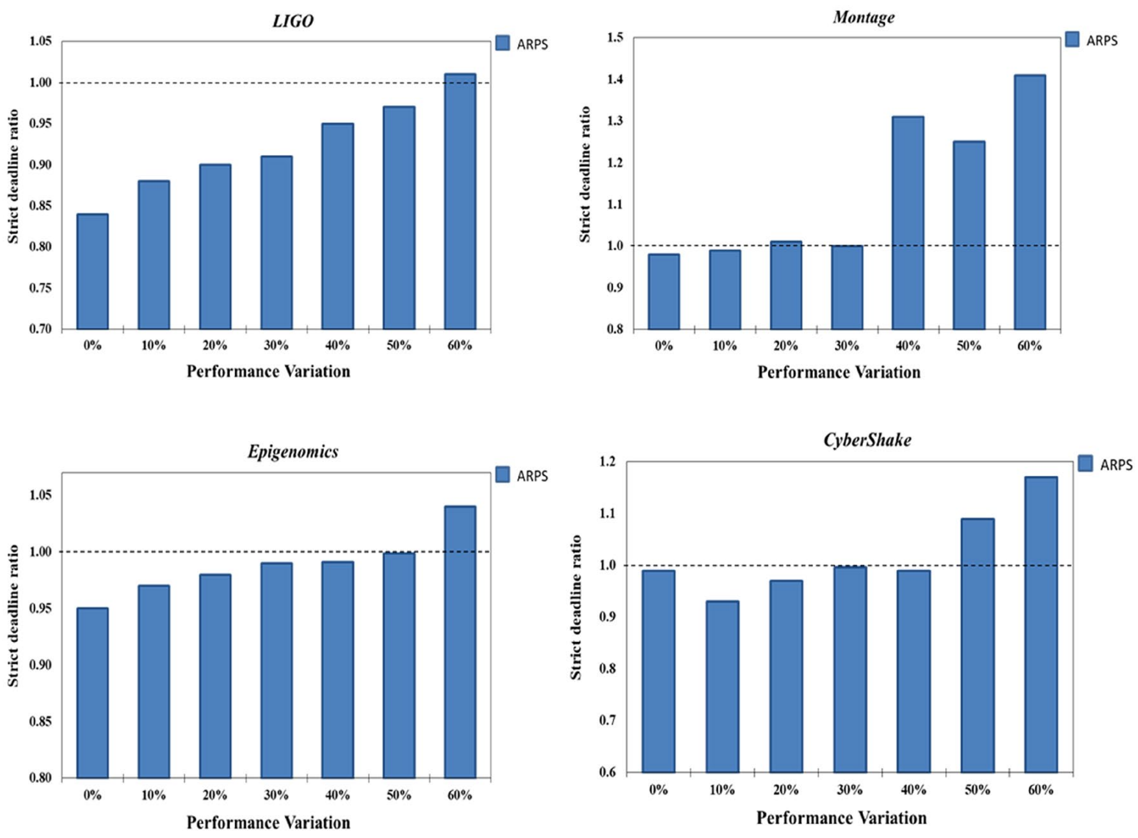


Fig. 5 Strict deadline ratio obtained for each workflow with different performance variation

Table 1 Types of VM based on Google Compute Engine offering

Name	Memory (GB)	Google compute engine units	Price per minute
n1-standard-8	30	22	\$0.0084
n1-standard-4	15	11	\$0.0042
n1-standard-2	7.5	5.50	\$0.0021
n1-standard-1	3.75	2.75	\$0.00105

ARPS Time Complexity Evaluation

Theorem 1 *The complexity time of ARPS is O (m*n), which denotes the number of tasks as m and the number of VM as n.*

Confirmation Choosing VM for every clustered topological tasks has complexity time $O(n)$. Hence, for entire tasks, the complexity time is $O(m*n)$. Therefore, the overall time complexity of ARPS is $O(m*n)$.

Conclusion and Future Work

APRS is an Adaptive Resource Provisioning and Scheduling algorithm for scientific workflows on IaaS cloud efficient of making good standard schedules was depicted. It has as goals reducing the total cost of utilizing the infrastructure of cloud as satisfying a user-described deadline. The procedure is dynamic to a particular range to reciprocate to the unpredicted delays and environmental uncertainties usual in cloud computing. Moreover, it has a static module that admits it to discover the better schedule for a set of workflow tasks, as a result enhancing the standard of the schedules it produces. By minimizing the workflow into sets of pipelines and homogeneous tasks that split a deadline, we can framework their scheduling as a UKP variation and solve it using dynamic programming in pseudo-polynomial time.

The experiment results demonstrate that our key has a more effective performance than other algorithms. It is achievable in satisfying deadlines in a state of unpredicted delays including VM performance variation, VM provisioning delay, network congestion and inaccurate estimations of task size. It attains this at less cost, even lesser than entirely static techniques which have the capability of utilizing the whole workflow form and comparing different outputs prior to the execution of workflow.

As future assignment, we would like to consider the workflow contains heterogeneous tasks at each level, such as SIPHT and observation with different optimization techniques, such as PSO algorithm, and compare their execution with ARPS. Moreover, we would like to enhance the resource model to examine the cost of data transfer between data regions, hence, that VMs would be used on different data centers. We would like to execute the ARPS with real-time workflows.

Acknowledgements The first author would like to thank "Anna Centenary Research Fellowship, Anna University" for supporting the proposed research work financially in the form of scholarship.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval All procedures performed in studies involving human participants were in accordance with the ethical standards of its institutional and /or national research committee and with the 1964 Helsinki Declaration and its later amendment or comparable ethical standards.

Informed consent Informed consent was obtained from all individual participants involved in the study.

References

1. Ullman JD. NP-complete scheduling problems. J Comput Syst Sci. 1975;10(3):384–93.
2. Schad J, Dittrich J, Quiané-Ruiz JA. Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proc VLDB Endow. 2010;3(1–2):460–71.
3. Mao M, Humphrey M. A performance study on the vm startup time in the cloud. In: 2012 IEEE Fifth International Conference on cloud computing. IEEE. 2012; p. 423–30. https://doi.org/10.1109/CLOUD.2012.103.
4. Malawski M, Juve G, Deelman E, Nabrzyski J. Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds In: Proceedings of the International Conference on high performance computing, networking, storage and analysis, 2012; pp. 1–11. https://doi.org/10.1109/SC.2012.38.
5. Zhou AC, He B, Liu C. Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds. IEEE Trans Cloud Comput. 2015;4(1):34–48.
6. Poola D, Ramamohanarao K, Buyya R. Fault-tolerant workflow scheduling using spot instances on clouds. In: ICCS. 2014; p. 523–533.
7. Byun EK, Kee YS, Kim JS, Maeng S. Cost optimized provisioning of elastic resources for application workflows. Futur Gener Comput Syst. 2011;27(8):1011–26.
8. Arabnejad V, Bubendorfer K, Ng B. Budget and deadline aware e-science workflow scheduling in clouds. IEEE Trans Parallel Distrib Syst. 2018;30(1):29–44.
9. Bhatti MK, Oz I, Amin S, Mushtaq M, Farooq U, Popov K, Brorsson M. Locality-aware task scheduling for homogeneous parallel computing systems. Computing. 2018;100(6):557–95.
10. de Oliveira D, Ocaña KA, Baião F, Mattoso M. A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. J Grid Comput. 2012;10(3):521–52.
11. Huu TT, Montagnat J. Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure. In: 2010 10th IEEE/ACM International Conference on cluster, cloud and grid computing. IEEE. 2010; p. 612–17. https://doi.org/10.1109/CCGRID.2010.23.
12. Ghose M, Kaur S, Sahu A. Scheduling real time tasks in an energy-efficient way using VMs with discrete compute capacities. Computing. 2020;102(1):263–94.
13. Nik SSM, Naghibzadeh M, Sedaghat Y. Cost-driven workflow scheduling on the cloud with deadline and reliability constraints. Computing. 2020;102(2):477–500.
14. Xu M, Cui L, Wang H, Bi Y. A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing. In: 2009 IEEE International Symposium on parallel and distributed processing with applications. IEEE. 2009; p. 629–34. https://doi.org/10.1109/ISPA.2009.95.
15. Deldari A, Naghibzadeh M, Abrishami S. CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud. J Supercomput. 2017;73(2):756–81.
16. Das D, Banerjee S, Kundu A, Chandra S, Pal S, Biswas U. An approach towards development of a migration enabled improved datacenter broker policy. Aptikom J Comput Sci Inf Technol. 2019;4(3):112–24.
17. Mandal R, Mondal MK, Banerjee S, Biswas U. An approach toward design and development of an energy-aware VM selection policy with improved SLA violation in the domain of green cloud computing. J Supercomput. 2020; 76:7374–93. https://doi.org/10.1007/s11227-020-03165-6.
18. Banerjee S, Adhikary M, Mandal D, Biswas U. Service delivery improvement for the cloud service providers and customers. Int J Comput Appl. 2012; 51(5):20–3. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.734.7497&amp;rep=rep1&amp;type=pdf.
19. Banerjee S, Chowdhury A, Mukherjee S, Biswas U. An approach towards development of a new cloudlet allocation policy with dynamic time quantum. Autom Control Comput Sci. 2018; 52(3):208–19.
20. Banerjee S, Chowdhury A, Mukherjee S, Biswas U. An approach towards development of an intelligent cloudlet scheduling mechanism for Cloud QoS improvement. Int J Hybrid Intell Syst. 2017; 14(1–2):21–30.
21. Roy S, Banerjee S, Chowdhury KR, Biswas U. Development and analysis of a three phase cloudlet allocation algorithm. J King Saud Univ-Comput Inf Sci. 2017; 29(4):473–83.
22. Liu L, Zhang M, Buyya R, Fan Q. Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing. Concurr Comput Pract Exp. 2017;29(5):e3942.
23. Abrishami S, Naghibzadeh M, Epema DH. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Futur Gener Comput Syst. 2013;29(1):158–69.
24. Calheiros RN, Buyya R. Meeting deadlines of scientific workflows in public clouds with tasks replication. IEEE Trans Parallel Distrib Syst. 2013;25(7):1787–96.
25. Faragardi HR, Sedghpour MRS, Fazliahmadi S, Fahringer T, Rasouli N. GRP-HEFT: A budget-constrained resource provisioning scheme for workflow scheduling in IaaS clouds. IEEE Trans Parallel Distrib Syst. 2019; 31(6):1239–54.
26. Niu M, Cheng B, Feng Y, Chen J. Gmta: A geo-aware multi-agent task allocation approach for scientific workflows in container-based cloud. IEEE Trans Netw Serv Manag. 2020;17(3):1568–81.

27. Wu Z, Ni Z, Gu L, Liu X. A revised discrete particle swarm optimization for cloud workflow scheduling. In: 2010 International Conference on computational intelligence and security. IEEE. 2010; p. 184–8. https://doi.org/10.1109/CIS.2010.46.
28. Yassa S, Chelouah R, Kadima H, Granado B. Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. Sci World J. 2013; pp.1–13. https://www.hindawi.com/journals/tswj/2013/350934/.
29. Poola D, Ramamohanarao K, Buyya R. Enhancing reliability of workflow execution using task replication and spot instances. ACM Trans Auton Adapt Syst (TAAS). 2016;10(4):1–21.
30. Mao M, Humphrey M. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: SC'11: Proceedings of 2011 International Conference for high performance computing, networking, storage and analysis. IEEE. 2011; p. 1–12. https://ieeexplore.ieee.org/abstract/document/6114435.
31. Jackson KR, Ramakrishnan L, Muriki K, Canon S, Cholia S, Shalf J, Wasserman HJ, Wright NJ. Performance analysis of high performance computing applications on the amazon web services cloud. In: 2010 IEEE Second International Conference on cloud computing technology and science. IEEE. 2010; p. 159–68. https://doi.org/10.1109/CloudCom.2010.69.
32. Bharathi S, Chervenak A, Deelman E, Mehta G, Su MH, Vahi K. Characterization of scientific workflows. In: 2008 Third Workshop on workflows in support of large-scale science. IEEE. 2008; p. 1–10. https://doi.org/10.1109/WORKS.2008.4723958.
33. Andonov R, Poirriez V, Rajopadhye S. Unbounded knapsack problem: dynamic programming revisited. Eur J Oper Res. 2000;123(2):394–407.
34. Andonov R, Rajopadhye S. A sparse knapsack algo-tech-cuit and its synthesis. In: Proceedings of IEEE International Conference on application specific array processors (ASSAP'94). IEEE, 1994; p. 302–13. https://doi.org/10.1109/ASAP.1994.331794.
35. Gilmore PC, Gomory RE. The theory and computation of knapsack functions. Oper Res. 1966;14(6):1045–74.
36. Gilmore PC, Gomory RE. A linear programming approach to the cutting stock problem—part II. Oper Res. 1963;11(6):863–88.
37 Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exp. 2011;41(1):23–50.
38. Stadill S. By the numbers: How google compute engine stacks up to amazon ec2. 2013. https://gigaom.com/2013/03/15/by-the-numbers-how-google-compute-engine-stacks-up-to-amazon-ec2/.
39. Bertsekas DP, Gallager RG, Humblet P. Data networks, vol. 2. New Jersey: Prentice-Hall International; 1992.