Multimedia Tools and Applications (2024) 83:50981–51007
https://doi.org/10.1007/s11042-023-17549-2

# Budget-based resource provisioning and scheduling algorithm for scientific workflows on IaaS cloud

**Rajasekar P[1] · Santhiya P[1]**

Received: 12 January 2023 / Revised: 11 September 2023 / Accepted: 16 October 2023 /
Published online: 9 November 2023


**Abstract**
The deployment of cloud computing, specifically Infrastructure as a Service (IaaS) clouds, have become an interested topic in recent years for the execution of compute-intensive scientific workflows. These platforms deliver on-demand connectivity to those infrastructure needed for workflow execution, providing customers to pay only for the service they utilize. As a result schedulers are forced to meet a quid-pro-quo among two main QoS criteria: cost and time. The maximum of this research work has been on making scheduling algorithms with the goal of reducing infrastructure costs as fulfilling a user-specified deadline. Few algorithms, on the other hand, have considered the problem of reducing workflow execution time while staying within a budget. This work consider on the latter scenario. We offer a Budget-based resource Provisioning and Scheduling (BPS) algorithm for scientific workflows used in IaaS service. This proposal was developed to face challenges specifically to clouds like resource performance variation, resource heterogeneity, infinite on-demand connectivity, and pay-as-you-go type (i.e. per-minute pricing). It is efficient of responding to the cloud dynamics, and is powerful in creating suitable solutions that fulfill a user-specified budget and reduce the makespan of the leveraged environment. At last, the experimental events confirms that it runs a workflow efficiently with respect to achieving budget of 94% and minimizing makespan of 29% than the state-of-the-art budget-aware algorithms.

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

✉ Rajasekar P
rajasekar.cse@sathyabama.ac.in

Santhiya P
santhiya.cse@sathyabama.ac.in

[1] Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Chennai, Tamilnadu, India

## 1 Introduction

Advanced scientific tools can acquire massive amount of data, allowing scientists to do more specific and significant evaluations and simulations. These scientific evaluations are usually referred to as workflows, which are applications made up of many computing tasks that are interdependent. Such workflows are compute-intensive applications that necessitate a lot of computing power to process the data under a specified period. Thus, they are mostly run over distributed machines. Scheduling algorithms are necessary for dynamically executing workflows as they optimize tasks over distributed machines. They make decisions based on predefined set of QoS definitions specified by the workflow customers like reducing the makespan while fulfilling a defined budget. This non-trivial obstacle of scheduling, in general, it is a familiar NP-Complete job [1–3] and as a result, algorithms should consider on obtaining suitable solution in a reasonable interval of time.

IaaS clouds distribute a scalable, adaptable, and conveniently accessible infrastructure for the execution of compute-intensive scientific workflows [4–6]. They let customers to utilize a distributed computing platform on-demand while spending simply for what they consume. This is made possible through the leasing of Virtual Machines (VMs), with a pre-specified network bandwidth, CPU capacity, storage, and memory space. Users can choose from a variety of resource bag (i.e. VM models) at different costs to serve a large-scale of workflow requirements. Except computing resources, IaaS distributors also deliver storage machines and network provisions to send the data out, in, and within their machines. To properly utilize these customized products and facilities, scheduling algorithms be made to consider multiple significant factors of clouds [6–8].

The initial feature is the on-demand, scalable resource platform. This service offers re-enhancing the scheduling problem as it has been specified for former distributed services like cluster and grids. Clouds do not deliver a bounded collection of computing machines, rather than, they deliver a virtual unbounded collection of machines with a variety of configurations available to be obtained and utilized simply for whenever they are required. This platform necessitates the use of a resource provisioning protocol that interoperates with the scheduling algorithm; a methodology that decides not only the number of VMs and their models to obtain from the cloud but also decides when is the appropriate time to obtain and shutdown them. As this contribution is adapted for cloud scenarios, the phrase scheduling should be considered to define to an algorithm that could make both scheduling and resource provisioning selections [9].

Next factor to investigate is the profit-based billing type adopted by cloud distributors. The expenses of deploying the infrastructure requires to be properly evaluated, if not, customers risk spending excessive and unreasonable costs. For example, the overall cost of executing the workflow application in the cloud is affected by the total sum of VMs provisioned, their model as well as the period of time they are run. Therefore, schedulers take to discover a quid-pro-quo in-between cost and makespan [10, 11].

A third factor affecting clouds is their dynamic condition, which has its own set of uncertainties. A typical case is the dynamics in performance shown by VMs with respect to running times [12]. This dynamics represents that regardless of a VM model being reported to have a defined CPU power, it would almost likely function at a less power that will modify over time. It also defines that two VMs of the similar models may run very differently. Additionally, when numerous concurrent users use a network at the same time, performance variations is detected in the network [12]. Yet an additional root of cloud dynamics are the resource deprovisioning and provisioning delays; there are no certainty

on these times and they would be highly varied and unstable [13]. Being aware of performance variation is significant for schedulers so that they could escape from uncertain delays and meet the QoS requirements.

The motivation of this research could be defined as:

- BPS includes both dynamic and static instances. Its dynamic nature relates to the instance that the scheduling selections are taken at compile-time, once tasks are transferred over to a scheduling queue.
- This enables it to respond to significant delays generated by inaccurate estimates or changes in the environment like those of VM provisioning delays, resource performance variation, and network congestion.
- The static phase broadens the algorithm's capability further into creating scheduling selections according to a single task to creating scheduling selections according to a cluster of tasks.
- The goal is to make quid-pro-quo between dynamic algorithm's local expertise and static algorithm's global expertise. This is accomplished by incorporating the pipeline concept and heuristically scheduling multiple tasks in the scheduling queue at the same time. As a result of this, BPS is capable to provide better optimization selections and identify enhanced quality schedules.

The contribution of this research could be defined in this manner:

- Due to these aforementioned conditions, we offer a Budget-based resource Provisioning and Scheduling (BPS) algorithm for scientific workflows on IaaS cloud.
- Our findings determines a quid-pro-quo among providing efficient solutions to respond to dynamics in the IaaS cloud and plan in advance to create high-quality schedules.
- It targets to minimize the total execution time as fulfilling a user-specified budget.
- It is potential of selecting what compute machines to deploy regarding heterogeneous VM models, and when they should be obtained and when they be made to terminate to escape from unnecessary costs.
- At last, our results reveal that it is elastic to the sum of tasks in the workflow, as well as dynamic and quick to react to cloud performance variations and capable of creating more efficient solutions than the advanced budget-aware algorithms.

The subsequent section of this paper is specified as does. Section II review works that are corresponding to our research. Section III discusses the application and resource model, Section IV explains proposed algorithm, and followed by their experimental results in Section V. The conclusion and future work are discussed in Section VI.

## 2 Related work

It has been extensively researched how to schedule scientific workflows in IaaS clouds [14–17]. The most of optimization techniques have the goal of meeting a deadline and minimizing the cost of infrastructure rental. Examples that have been proposed are [18–22]. Only very few available algorithms aim to satisfy a budget while reducing workflow makespan. An example is the $PCB^2$ algorithm [23], it divides the workflow as pipelines and determines the best resource model that optimizes the budget allocation. Moreover, it

uses a time unit pricing model based on the new billing period model of per minute, which is similar to our research. Next work, if there is budget leftover, the CG algorithm [24] generates a workflow's schedule through iteratively revising an original schedule plan that favours the usage of more efficient VM models. Other proposals with similar goals have used CGA [25], C-PSO [26], HPSO [27], CTDHH [28], CbCP [29], and I_MaOPSO [30] to create a static plan that reduces the makespan prior to the execution time. These algorithms depend on highly requiring meta-heuristic approaches to calculate a near-optimal schedule. The methodology varies to our solution in that we adopts an adaptable, lightweight, heuristic-made dynamic approach that creates resource provisioning and scheduling decisions according to the current state of the system at runtime. The BDCWS [31] optimization technique likewise considers budget as such a constraint, but unlike our solution, it also has a deadline as function of its scheduling goals.

Another such budget-constrained algorithms are BAGS [32], it divides the workflow as SoTs (sets of tasks) under the same workflow phase. BAGS is designed on an automated budget distribution model that dynamically manages the scheduling and resource provisioning strategies of SoTs when tasks are ready to be executed. BAGS, like us, assumes fine-grained billing intervals (e.g., one minute) that are suitable for average workflow task execution time. Next, the BDT algorithm [33] adopts similar plan, integrating tasks under the same workflow phase. The budget is allocated to every phase, and any excess budget is gradually shifted to the next phase by the algorithm. It works on the basis of an hourly billing slot, but pay no attention to VM performance variations. The findings of BDT investigate a set of phase-based budget allocation approaches determined by factors like the total count of tasks in each level and the total count of phases in the workflow. Our approach differs compare to BDT in that it schedules tasks separately when they are ready to execute, for example, once the task's precedents have done running and the input file is accessible.

Algorithms HBCS [34], BDAS [35], and MSLBL [36] are another three examples of budget-constrained approaches, and tasks are prioritized first in all of them, then, for each task, a suitable solution is identified in a certain manner. For each task, the resource with the largest aggregate weight of certainty over normalized cost and time is chosen using the HBCS algorithm [34]. If the available budget is substantial, the time element has more effect; if the available budget is little, the cost consideration takes precedence. The HBCS algorithm is inappropriate to low-priority tasks since higher-priority tasks have more budget available than low-priority tasks. The given budget is allocated on the workflow tasks first, and the share of each task out of the overall budget is decided in the BDAS algorithm [35]. Then, the fastest VM is chosen based on the defined budget for each task. The problem is divided into two components in the MSLBL algorithm [36], specifically, meeting the budget constraint and reducing the makespan. The initial step is handled by allocating the budget requirement of workflow to that of every task, and the next step is handled by scheduling every task heuristically. The MSLBL algorithm attempts to address the unfairness of the HBCS algorithm. The major drawback of these strategies is their task-level optimal solution, which is a quid-pro-quo for their capability to adapt to unanticipated delays.

Another example is the MW-DBS [37], a scheduling algorithm for processing multiple workflows execution within user specified budget and deadline limits. Anyway, the drawback is that it chooses a task according to the order of the task's rank and without having to consider the actual task's execution cost. Hence, it fails to achieve an optimal balance between budget and deadline. Next algorithm is the MQ-PAS [38], proposed for similar scenario under user specified budget and deadline limits. Anyway, this strategy did not take advantage of concurrent tasks entering from multiple workflows to reduce the amount of

time when computing resources are idle and use the workflow budget as such a backup to the deadline constraint. Other works are SFTD [39] and EBPSM [40], proposed for multi-workflows execution under budget constraints. Anyway, these strategies are also not efficient to minimize the unused slots of running resources. As a result, neither of them aim to make full use of the allotted budget of each workflow in order to achieve quicker execution time, however, contrary to ours, those are proposed for multiple workflows execution.

An adaptive budget-based algorithm that potential of generating scheduling and auto-scaling selections to reduce the total execution time of workflow is developed by [41–44]. Anyway, they examine an hourly budget in place of a budget constraint for the whole workflow completion and target to enhance the running workload of workflows completion. In contrast to our objective, the critical greedy [45–48] approach examines a financial restriction as reducing the end-to-end delay of the workflow completion. Anyway, it performs not incorporate billing slots on its cost evaluation and therefore examines VMs charged per time unit. Moreover, the solution of the algorithm is mapping of a task to VM and motivators do not suggest a heuristic to allocate the task to existing VMs as examining their boot time and performance variation.

Unlike approaches that are entirely dynamic or static, our approach integrates both to create a quid-pro-quo between scalability and the potential benefits of optimization algorithms. NBWS [49] and CB-DT [50] are the sample algorithms focussing to fulfill this. It uses a global optimization technique to discover the best task-to-VM type mapping. This methodology is using at running time to expand the resource pool in/out as needed, as well as to schedule tasks as they become available. Our algorithm differs from NBWS and CB-DT in that the static element does not analyse the complete workflow model and rather, it significantly enhances the partition of the workflow tasks. Additionally, rather than selecting a VM type, our algorithm makes a general schedule for these tasks. Moreover, in Table 1, we give an overview of the works that have been discussed. To the best of our knowledge, there is not a single model of aforementioned budget constrained approaches that has been designed to handle the problem of single workflow scheduling, which is related to the challenges faced by IaaS cloud infrastructures.

## 3 Application and resource model

We regard workflows designed as Directed Acyclic Graphs (DAGs); in other words, graphs with fixed edges and no cycles dependencies. According to protocol, a workflow W is formed of a collection of tasks $T = \{t_1, t_2,...,t_n\}$ and a collection of edges E. An edge $e_{ij} = (t_i,t_j)$ persists if there is a data interdependency in-between task $t_i$ and $t_j$, scenario in which task $t_i$ is supposed to be parent of $t_j$ and task $t_j$ is supposed to be child of $t_i$. According to this, a child task cannot execute until such time as all of its parents tasks have ended and its input file is ready in the related computing resource. Additionally, a workflow is integrated with a budget $B_w$, specified as a cost bound for its execution. On top of that, we consider that the size of a task $S_t$ is estimable in Million of Instructions (MIs) and that, for each single task, this findings is given as recorded information to scheduler. The mathematical notations taken in this paper are described in Table 2.

VMs are obtained according to an on-demand charging model and are priced per charging slot *cs,* with any insufficient usage outcome in the VM utilization being billed up to nearest pricing slot. Our proposal handles a heterogeneous platform where VMs with various VM models $VMT = \{vmt_1, vmt_2,...,vmt_n\}$ which have different processing

**Table 1** Summary of related works

| Existing Works | Strategies | | | |
|---|---|---|---|---|
| | Static Heuristic | Static Meta-heuristic | Dynamic level-based | Dynamic task-based |
| $PCB^2$ [23] | ✓ | – | – | – |
| CG [24] | – | ✓ | – | – |
| CGA [25] | – | ✓ | – | – |
| C-PSO [26] | – | ✓ | – | – |
| HPSO [27] | – | ✓ | – | – |
| CTDHH [28] | – | ✓ | – | – |
| CbCP [29] | – | ✓ | – | – |
| I_MaOPSO [30] | – | ✓ | – | – |
| BDCWS [31] | ✓ | – | – | – |
| BAGS [32] | ✓ | – | ✓ | – |
| BDT [33] | ✓ | – | – | – |
| HBCS [34] | ✓ | – | – | – |
| BDAS [35] | – | – | ✓ | – |
| MSLBL [36] | – | – | ✓ | – |
| MW-DBS [37] | – | – | ✓ | – |
| MQ-PAS [38] | – | – | ✓ | – |
| SFTD [39] | – | – | ✓ | – |
| EBPSM [40] | – | – | ✓ | – |
| BDDC [41] | – | ✓ | – | – |
| EBABC-PF [42] | – | ✓ | – | – |
| Min-Max [43] | – | ✓ | – | – |
| Hybrid-ED [44] | – | – | ✓ | – |
| APRS [45] | – | – | ✓ | – |
| FDPM [46] | – | – | ✓ | – |
| SRPSM [47] | – | – | ✓ | – |
| EPSM [48] | – | – | ✓ | – |
| NBWS [49] | – | – | ✓ | – |
| CB-DT [50] | ✓ | – | – | – |
| [BPS] Ours | – | – | – | ✓ |

potential $PC_{vmt}$ and various cost per-pricing slot $C_{vmt}$. The processing potential of a VM is computed in Million of Instruction Per Second (MIPS). We consider the CPU capacity of VM is not static as studied by [6] and that vendors promote the maximum CPU potential utilizable by VMs. Additionally, we consider an infinite sum of VMs could be obtained from the vendor.

The runtime of a workflow task t in a VM of model *vmt* is defined as $RT^t_{vmt}$ and is computed according to the size $S_t$ of the task and the processing potential $PC_{vmt}$ of the VM. This is demonstrated in Eq. 1. Regard that this measure is just an evaluation and our algorithm does not depend on it being 100% fully precise. Additionally, we consider that VMs with high CPU power are high expensive to obtain than VMs with low power. In this manner, the task runtime computation through the less cost VM model to the larger estimate value (slowest runtime) but possibly the lesser cost.

**Table 2** Mathematical notations

| Notations | Definitions |
|---|---|
| W | A workflow |
| T | A collection of tasks |
| E | A collection of edges |
| $(t_i, t_j)$ | $t_i$ is the parent task, $t_j$ is the child task |
| $\boldsymbol{e_{i,j}}$ | Transfer time between $t_i$ and $t_j$ |
| $D^t_{in}$ | Data input |
| $D^t_{out}$ | Data output |
| $VM$ | Virtual Machine |
| $VMT$ | Virtual Machine Type |
| $PC_{vmt}$ | Processing potential of VM |
| $C_{vmt}$ | Cost of VM |
| $B_{vmt}$ | Bandwidth of VM |
| $T_{pdelay}$ | Provisioning delay of VM |
| $T_{ddelay}$ | De-provisioning delay of VM |
| $cs$ | Charging slot |
| $GS$ | Global Storage |
| $GS_{read}$ | Global Storage reading rate |
| $GS_{write}$ | Global Storage writing rate |
| $T^{D^t_{in}}_{vmt}$ | Time required for input data transfer |
| $T^{D^t_{out}}_{vmt}$ | Time required for output data transfer |
| $RT^t_{vmt}$ | Runtime of a task tin a VM |
| $S_t$ | Task Size of a task |
| $PT^t_{vmt}$ | Processing time of a task t in a VM |
| $eft^t_{vmt}$ | Earliest finish time of a task t in a VM |
| $C^t_{vmt}$ | Cost of task t in a VM |
| $TET_w$ | Makespan or Total Execution Time of a workflow |
| $TEC_w$ | Total Execution Cost of a workflow |
| $B_w$ | Budget of a workflow |

$$RT^t_{vmt} = S_t / PC_{vmt} \tag{1}$$

$$B_{vmt}(t) = \left(B_{vmt} / Tr_t\right) \tag{2}$$

$$GS_{read}(t) = \left(GS_{read} / Tr^{read}_t\right) \tag{3}$$

$$GS_{write}(t) = \left(GS_{write} / Tr^{write}_t\right) \tag{4}$$

We adopt a global storage (GS) service like Amazon S3 for data distribution in-between tasks. Every task get back its input file $D^t_{in}$ from the data repository and send its output file $D^t_{out}$ on the similar way. The read and write rates of the GS are $GS_{read}$ and $GS_{write}$ accordingly. Also every VM has a bandwidth power $B_{vmt}$ connected with it. This bandwidth power and I/O processing rate of the storage resource modify over time, according to the sum of

transactions $T_r$ processing at time t. And also we believe network congestion makes a variance in data transport times [22]. The bandwidth distributed to a transfer relies upon the prevailing variance for the network component being run. Along with, we consider a global storage with an unbounded storage power. In this manner, the speed at which it is efficient of reading and writing file modify according to the sum of processes at the moment reading or writing file from the resource. This is demonstrated in Eqs. 2, 3, and 4. The time period it needs to input file from the shared storage to a VM is demonstrated in Eq. 5. Correspondingly, the time period it needs to output file from a VM into the shared storage is demonstrated in Eq. 6.

$$T_{vmt}^{D_{in}^t} = \left(D_{in}^t / B_{vmt}\right) + \left(D_{in}^t / GS_{read}\right) \tag{5}$$

$$T_{vmt}^{D_{out}^t} = \left(D_{out}^t / B_{vmt}\right) + \left(D_{out}^t / GS_{write}\right) \tag{6}$$

$$PT_{vmt}^t = \left( PT_{vmt}^t + T_{vmt}^{D_{out}^t} + T_{vmt}^{D_{out}^t} \right) \tag{7}$$

$$eft_{vmt}^t = \max_{p \in \text{t.parents}} \{eft\} + PT_{vmt}^t \tag{8}$$

$$C_{vmt}^t = \left\lceil \left( PT_{vmt}^t + T_{pdelay} + T_{ddelay} \right) / cs \right\rceil * C_{vmt} \tag{9}$$

We adopt a design in which the GS and VMs are put in the same availability location. Therefore, data transfer in-between VMs and storage is cost-free, as is the context for majority IaaS vendors. Although, we consider that the output file of a task is also placed on the VM's common storage. In this manner, child tasks running on exactly the same VM do not want to retrieve their input file from the shared storage. By executing this style, the spending cost of the utilized environment could be considerably minimized. Therefore, the overall processing time of a task on a VM of model *vmt* is demonstrated in Eq. 7. The earliest finish time $eft_{vmt}^t$ of running a workflow task $t$ after its parent tasks completion on an offered *vmt* as defined in Eq. 8. Moreover, the cost of a task that executes on a VM of model *vmt* including the VMs provisioning time delay $T_{pdelay}$ and deprovisioning time delay $T_{ddelay}$ is demonstrated in Eq. 9. Eq. 10 and 11 demonstrate how the total execution time (i.e. makespan) and total execution cost of a workflow.

$$TET = \max \left\{ eft_{vmt}^t : t \in T \right\} \tag{10}$$

$$TEC = \max \left\{ C_{vmt}^t : vmt \in VMT \right\} \tag{11}$$

### 3.1 Problem formulation

According to the former calculations, the problem could be formally specified in this manner: find a solution with lesser TET (Total Execution time or makespan) and for which the sum of TEC (Total Execution Cost) does not violate the workflow's budget is demonstrated in Eq. 12. The above mentioned standard equations have been referred from (45-47). And a proposed architecture for this system is shown in Fig. 1.

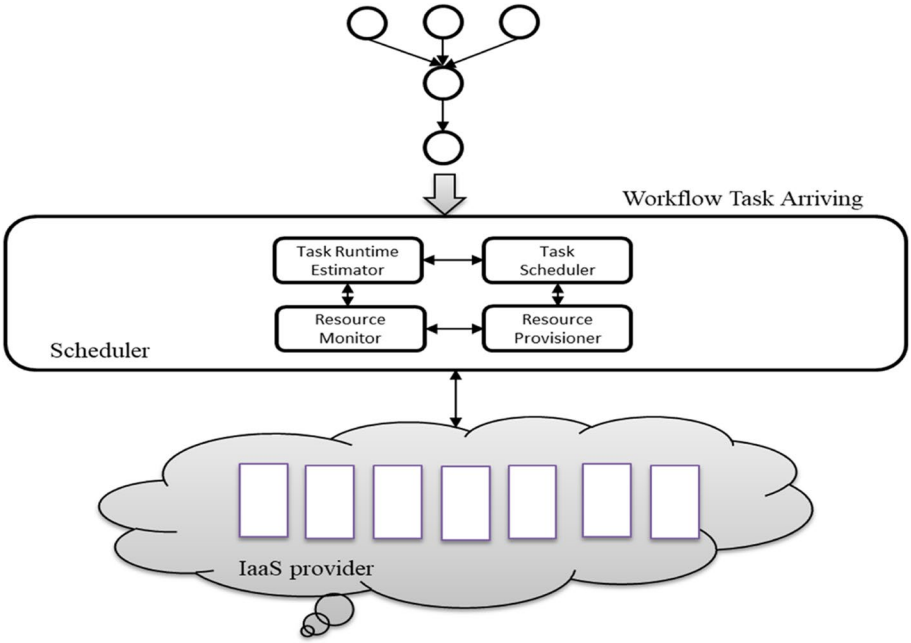


**Fig. 1** Proposed Scheduling Architecture

$$\begin{aligned} &\text{Minimize TET or Makespan M} \\ &\text{Constraint to } \mathrm{TEC}_{\mathrm{w}} \leq B_{\mathrm{w}} \end{aligned} \tag{12}$$

## 4 BPS algorithm

This section explains the BPS heuristics' concept as well as the algorithm in detail.

### 4.1 Overview

BPS includes both dynamic and static instances. Its dynamic nature relates to the instance that the scheduling selections are taken at compile-time, once tasks are transferred over to a scheduling queue. This enables it to respond to significant delays generated by inaccurate estimates or changes in the environment like those of VM provisioning delays, resource performance variation, and network congestion. The static phase broadens the algorithm's capability further into creating scheduling selections according to a single task to creating scheduling selections according to a cluster of tasks. The goal is to make quid-pro-quo between dynamic algorithm's local expertise and static algorithm's global expertise. This is accomplished by incorporating the pipeline concept and heuristically scheduling multiple tasks in the scheduling queue at the same time. As a result of this, BPS is capable to provide better optimization selections and identify enhanced quality schedules.

A pipeline is a typical structural form in workflow applications and consisting of a cluster of tasks having a 1 to 1 linear relationship in-between both of them. A pipeline P is formally specified as a cluster of tasks $T_p = \{t_a, t_b,...,t_z\}$ wherein $z \geq 2$ and there is a directed edge $e_{a,a+1}$ connects task $t_a$ with task $t_{a+1}$. To put it another way, $t_1$ is the parent task of $t_2$, $t_2$ of $t_3$ and etc. The 1st task in a pipeline can have multiple parents but it can only share a single child task. All such tasks could only have a single parent (the initial task of pipeline) and a single child (the following task of pipeline). A pipeline therefore has a budget $B_p$, which is same to the budget of the end task within the set. Figure 1a depicts this one.

With regard to exploring pipelines within a workflow, we should necessarily broaden the context from a single task to such a cluster of tasks that could be scheduled very conveniently together instead of separately. To minimize overheads in execution and transfer of data, as well as the delays in VM provisioning and deprovisioning, tasks within a pipeline are grouped widely and always executed within the same VM. There are two significant reasons for this. First and foremost, tasks are consecutive and must be completed sequentially. So, assigning them to various VMs has no value in terms of parallelization. Second, by executing within the same VM, the output data of parent task becomes the input data of the child task, we save cost and time by not having to transport these data from and to the global storage.

The solution for scheduling queued jobs is based on workflow application structure. In addition to pipelines, a workflow can also have parallel layers comprised of tasks that do not have any dependencies in-between them. These tasks could execute in parallel and are commonly identified only when data consolidation or sharing exists. In the sharing of data [51], a task's output is allocated to various tasks for execution. In the consolidation of data [51], the output of several tasks is transferred, or aggregated, by a single task. Both of these structures is depicted in Fig. 2a.

These application's parallel tasks could be homogeneous (similar type). In scientific workflows, the circumstance where the tasks are homogeneous is common; instances of popular applications by this attributes are CyberShake Epigenomics, Montage, and Ligo. As a result, we develop an approach to optimally schedule homogeneous paralleled tasks within the same size (MIs) while at the same level in the workflow application. These parallel tasks will do have the same budget when adopting a level-based budget distributing plan. Consider the consolidation of data as an example, all parallel tasks must be completed prior to the consolidation task may proceed, thus they could be allotted the same budget, which is the same as the budget for the consolidation task to execute.

BPS's major static scheduling technique is to bundle queued tasks of the similar type even with the same budget into clusters. Figure 1a shows two sample clusters, the first of which contains all Type 1 tasks and the second of which has all Type 4 tasks. It's a lot easier to schedule these clusters of tasks than it is to schedule a workflow. There are no inter-dependencies, the tasks are all the same, and they must all be completed within the same budget. We define the problem of executing these tasks within budget and for the least makespan as a form of the unbounded knapsack problem, and use dynamic programming to find the best solution. Moreover, pipelines are also consolidated into clusters and scheduled in the similar way as tasks are consolidated into clusters of tasks. Figure 2c and d shows an example of a cluster of pipelines.

As a result, we created a dynamic algorithm that can adjust to some extent to unanticipated delays caused by the uncertainty of cloud environments but also contains a static module that allows it to create higher quality schedules and fulfill budget at less makespan. Additionally, it consolidates a heuristic-based strategy and dynamic programming to execute workflows in an elastic and cost-effective manner.

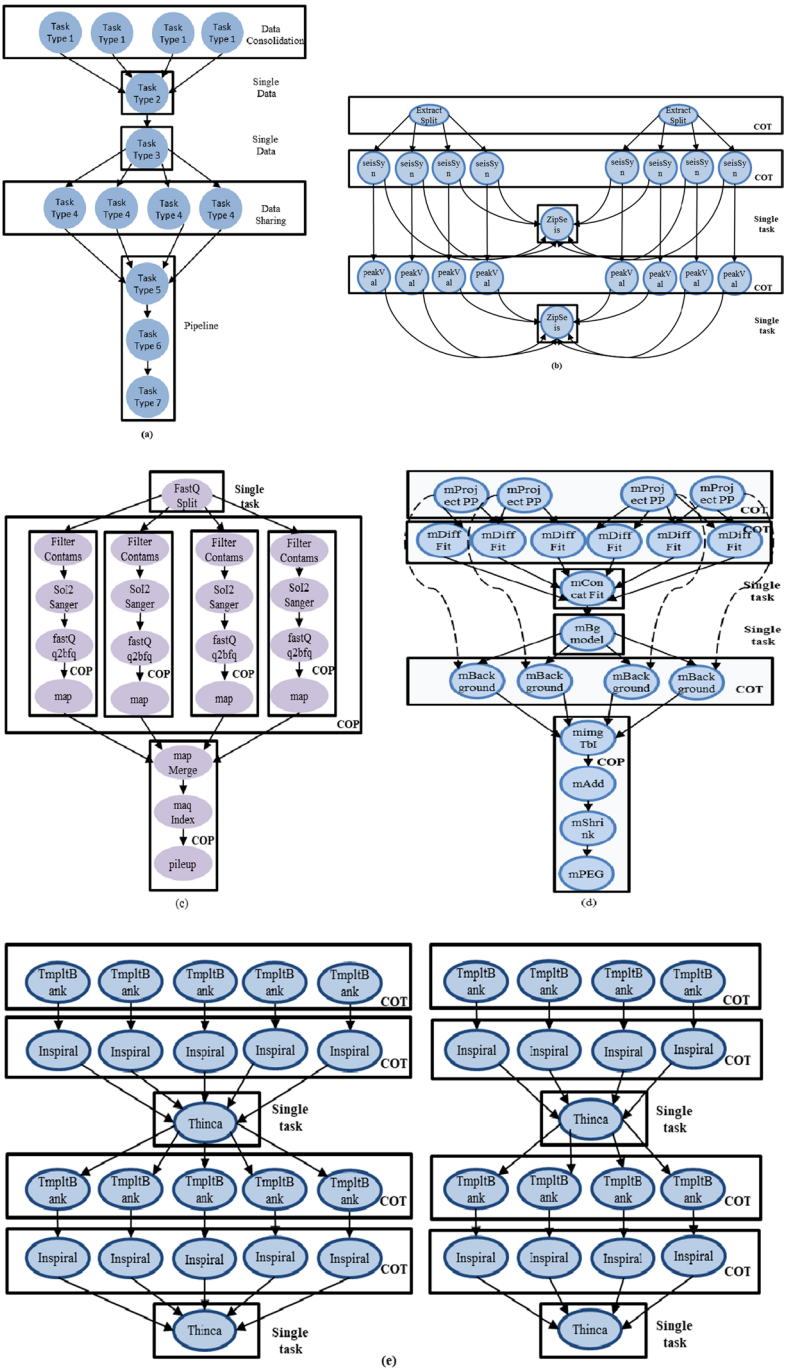


**Fig. 2** Workflow applications examples (**a**) Samples of cluster of tasks and three various structural forms identified in workflows: data consolidation, data sharing & pipelines. (**b**) CyberShake workflow (**c**) Epigenomics workflow (**d**) Montage workflow (**e**) LIGO workflow

### 4.2 Unbounded knapsack problem (UKP)

The UKP is a NP-hard problem evolved from the complexity of deciding which element to put in a given knapsack. Having n elements of various possibilities, each element category $1 \leq i \leq n$ along with a related weight $w_i$ & value $v_i$, the purpose is to identify the number and category of elements to put such that the knapsack weight ratio W is not surpassed and the element's total value is optimized. Infinite load of each element category are considered.

Let $x_i \geq 0$ be the number of variables of element $i$ that will be put in the knapsack. Therefore UKP is specified as (13).

$$Maximize \sum_{i=1}^{n} v_i x_i \quad Subject\ to \sum_{i=1}^{n} w_i x_i \leq W \tag{13}$$

This challenge could be handled suitably with dynamic programming by taking smaller capacity knapsacks as sub-problems and keeping the optimal combination intended for each capacity. As $w_i > 0$, hence a vector would be written as $m[w_i]$, whereby $m[w_i]$ is the best solution that would be achieved with such weight $\leq w_i$. As a result, $m[0] = 0$ and $(m[w_i] = \max_{wj \leq wi} (v_j + m[w_i - w_j])$. This solution has an $O(nW)$ time complexity since evaluating every $m[w_i]$ requires finding n elements and there would be W values of $m[w_i]$ to compute. This computational time is pseudo-polynomial since it increases exponentially with size of W. However, there are various algorithms that could solve UKP efficiently. The EDUK [52] algorithm, for example, which incorporates the concepts of monotonic recurrence [53], dominance [54] and periodicity [55]. Findings considered by the authors represent its adaptability. For e.g., the average runtime for $W > 2 \times 10^8$, $n = 10^5$, and the elements with weight in the $[1, 10^5]$ ratio was computed to be 0.150 seconds. Another example that could solve UKP efficiently by WRPS algorithm [56].

### 4.3 Algorithm

BPS is divided into three major steps. The first is an offline technique that divides the DAG as CoTs before scheduling. The next is an automated budget-distribution process that is continued during whole workflow execution. It allots a share of the available budget to tasks that have yet to be scheduled. As tasks become ready for execution, the final phase is capable for generating a scheduling plan with respect to CoTs. The next sub-sections go through each of these scenarios in further detail. The flow chart for this system is shown in Fig. 3.

#### 4.3.1 DAG pre-optimization

The objective of this step is to find and split the DAG as CoTs. Tasks are divided into clusters if two or more tasks associated from the parallel data-distribution context and have the same single parent or if the tasks are entry-level and do not have any parent tasks. If a task fails to meet any of these criteria, it is specified to its own single-task cluster. CoTs with too many tasks are therefore divided into three types. The first type is clusters of homogeneous tasks, which means that all of the tasks in the cluster do the identical computations. The second is filled of clusters of parallelizable tasks that are pipelines. The last is filled of clusters of a single task. As a result of the pre-optimization stage, the following sets are identified:

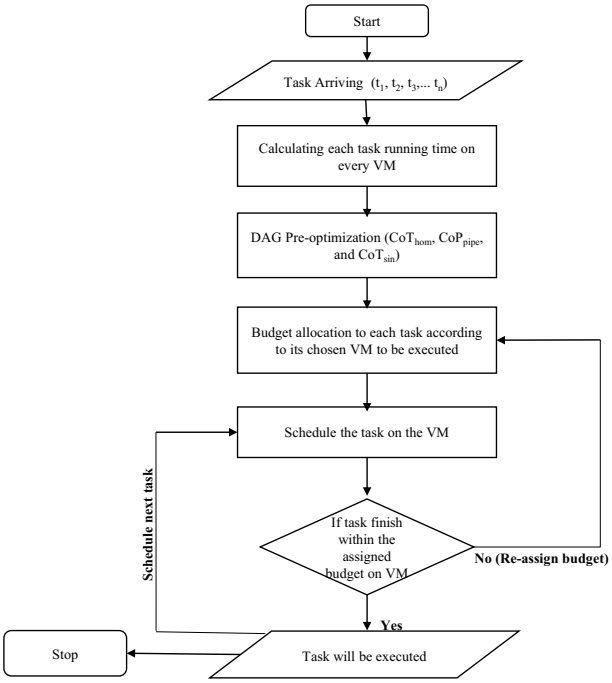


**Fig. 3** Flow chart for the proposed system

$$
\begin{aligned}
---CoT_{homogeneous} &= \{\mathrm{cot}_1, \mathrm{cot}_2, \ldots, \mathrm{cot}_n\} \text{ Clusters of homogeneous task} \\
---CoP_{pipelines} &= \{cop_1, cop_2, \ldots, cop_n\} \text{ Clusters of pipelines} \\
---CoT_{single\ task} &= \{\mathrm{cot}_1, \mathrm{cot}_2, \ldots, \mathrm{cot}_n\} \text{ Clusters of a single task}
\end{aligned}
$$

### 4.3.2 Budget distribution

Then the budget distribution algorithm assigns sub-budget for each workflow task according to a budget defined by the user. This sub-budget, combined with an possibility of input/output file sharing, will determine whether a task is scheduled on an already provisioned VM in the pool of resources or a new VM supplied from the IaaS vendor at runtime. This budget distribution algorithm's process is to predict possible task scheduling sequence in a workflow. The entrance task(s) in a workflow's entry level are scheduled early, then by their successors of next level while the last task is scheduled. Under this scenario, we use the DTL technique to allocate each task from a level in a workflow's hierarchy, as shown in Eq. 14.

$$
level(t) = \begin{cases} 0, & if\ pred(t) = \varnothing \\ \max\limits_{P \in \mathrm{parent(t)}} level\ (p) + 1, & otherwise \end{cases} \tag{14}
$$

In addition, to identify the prioritization of tasks in a level, we rank them in ascending order based on their (EFT), as given in Eq. 15.

$$eft(t) = \begin{cases} PT^{t}_{vmt}, & if\ pred(t) = \emptyset \\ \max\limits_{P \in \text{parent(t)}} eft\ (p) + PT^{t}_{vmt}, & otherwise \end{cases} \quad (15)$$

The algorithm iterates over the tasks based on this order and allocates the budget to each task after predicting the possible task scheduling sequence. In addition to this, the algorithm computes the sub-budget of a task according to the cost $C_{vmt}$ of specific VM instances. Initially, the process begins by selecting VM with the cheapest instances for the task. If there is any extra budget after all tasks have been assigned sub-budgets, the algorithm uses it to update the sub-budget distribution for such faster VM instance, starting with the first task in the sequence. This process is called SFTD [32] and its complete strategy is defined in Algorithm 1.

```
1:  procedure DistributeBudget(β, T)
2:      S = tasks' estimated execution order
3:      for each task t ϵ T do
4:          assignLevel(t,l)
5:          initiateBudget(0, t)
6:      end for
7:      for each level l do
8:          T_l = set of all tasks in level l
9:          sort T_l based on ascending Earliest Finish Time (EFT)
10:         put (T_l , S)
11:     end for
12:     while β > 0 do
13:         t = S.poll
14:         vmt = chosen VM model
15:         distributeBudget (C^t_vmt, t)
16:         β = β - Cft^t_vmt
17:     end while
18: end procedure
```

**Algorithm 1** Budget distribution

### 4.3.3 Scheduling

After a workflow has been pre-optimized and assigned sub-budget, task scheduling could occur. Over the entry-level iteration, all entrance tasks (or share no parent tasks) get ready to be executed and are assigned in a scheduling queue. These tasks are run first, and whenever they are completed, their successor tasks are added to the queue. This action is iterated till all workflow tasks are executed efficiently. To execute the tasks in the scheduler, they should be divided into clusters of tasks, clusters of pipelines and cluster of a single task. A *cot* is a cluster of homogeneous tasks $T_{cot}$ that could process in simultaneously. Every task in a cluster has the same budget, is of the similar type (homogeneous), and does not belong to a pipeline. Systematically, $cot_{homogeneous} = (T_{cot}, \beta_{cot}, T_{cot})$. The concept of cluster of pipelines *cop* is same but rather than a cluster of tasks, the cluster comprises one or many parallelizable pipelines $P_{cop}$. The subset $cop_{pipelines} = (P_{cop}, \beta_{cop})$, where $\beta_{cop}$ is the common budget shared by all pipelines in the cluster, explicitly specifies the idea. And the concept of cluster of a single task $cot_{single}$, holds only one task with a budget.

To pick the bundles of cluster of homogeneous tasks $CoT_{homogeneous} = \{cot_1,..., cot_n\}$, cluster of pipelines $CoP_{pipelines} = \{cop_1,...,cop_n\}$, and cluster of a single task

$CoT_{single} = \{cot_1,...,cot_n\}$, every task in the scheduler is handled as follows. If a task is not part of a pipeline, then it is incorporated into $cot_i$ that has same tasks with the same budget. In there is no suchlike $cot_i$, a new cluster is made and the task given to it. If the task, however, is part of a pipeline, the similar pipeline is added to the $cop_i$ which comprises pipelines with similar type of tasks and budget. If no such $cop_i$ with these qualities exists, a new cluster is made with the specified pipeline as its single entity. At last, if a task fails to meet both of these criteria (i.e. *Cot* & *Cop*), it is specified to its own single-task cluster $CoT_{single}$.

After *CoT*, *CoP* and $CoT_{single}$ are made, we begin to execute them. First two kinds of clusters are scheduled adopting the similar strategy, with the exception that pipeline tasks should be regarded as a single entity. We take *CoT* to demonstrate the heuristic; however, relatively similar criteria apply for scheduling *CoP*. To schedule *BoT*, we repeat the steps following for every cluster $cot_i \in CoT$ with multiple tasks (clusters with a particular component are regarded as a special instance and scheduled immediately). Firstly, BPS aims to lower the capacity of the cluster and reutilize early provisioned VMs by distributing too many tasks as feasible to idle VMs. The set of tasks are allocated to a single VMs is based on set of tasks that can be completed and prior to the VM's next charging slot and prior to their deadline. As a result, wasting of early leased CPU cycles is avoided without compromising the budget of the workflow. Following that, a resource provisioning strategy for the upcoming tasks in the cluster is made.

To make a resource provisioning plan that is as efficient as possible, BPS should investigate several solutions through suitable VM types and evaluate their potential costs. We fulfill this by defining the problem as a version of UKP and solving it through dynamic programming to identify the optimal selection of VMs that could execute the tasks in the bag in the shortest amount of time with the least amount of expense. A knapsack element is specified by its kind, load, and costs. With respect to our scenario, we specify a scheduling knapsack element $SKI_j = (VMT_j, NT_j, C_j)$ where the element kind is relevant to VM type $VMT_j$, the load is the total count of tasks $NT_j$ that can execute in such a VM related to the type $NT_j$, and the cost $C_j$ of executing $NT_j$ tasks in such a VM related to the type $VMT_j$ within their budget. Furthermore, we assume that each type of VM might possibly lease an infinite number of them, and we define the knapsack load capacity as the total count of tasks in the cluster, that is, $W = |T_{cot}|$. The aim is to explore a cluster of SKI elements whose total capacity (number of tasks) is at least equal as the knapsack load capacity (total count of tasks in the cluster) and whose total value is as minimal as possible (the cost of executing the tasks is minimal). The scheduling problem regarding a cluster of tasks is stated as Eq. 16.

$$Minimize \sum_{i=1}^{n} C_i \times x_i \quad Subject\ to \sum_{i=1}^{n} NT_i \times x_i \geq \left|T_{cot}\right| \tag{16}$$

For each VM type, a resource provisioning $RP_i^{cot} = (VMT_i, numVM_i, NT_i)$ is obtained after solving the UKP problem. It determines the number of $\boldsymbol{VMT_i}$ type VMs to be used ($numVM_i$) and represents the number of tasks to execute on each VM ($NT_i$). And a provisioning plan of the type $RP_{cot}^{fastest} = (VMT^{fastest}, W, 1)$ is made in rare instances where no VM types are capable of completing the tasks within the budget. This proves that for each task in the cluster, a VM of the quickest type should be provisioned so that they could execute in simultaneously and finished as soon as possible. BPS then searches for a VM of type $VMT_i$ that has early been provisioned and is available to utilize for every $RP_i^{sot}$ for which $numVM_i > 0$. If it remains, it is used to schedule $NT_i$ tasks from the cluster. So, we focus on reducing cost by utilizing time slots that have been early leased for and try to minimize

provisioning delays with respect to newly provisioning VMs. If no free VMs of the desired type are available, a new VM is launched and $NT_i$ tasks are executed to it. Algorithm 2 shows the pseudo-code for scheduling CoT.

```
1.  Procedure ScheduleCoT(CoT)
2.     for all cot ϵ CoT do
3.        if all cot.tasks.size > 1 then
4.           RP^cot = UKPBasedProvisoningPlan(cot)
5.           for all RP_i^cot = (VMT_i, numVM_i, Nt_i) ϵ RP^cot do
6.              for k=0; k<numVM_i; k++ do
7.                 n tasks = min{Nt_i,cot.size}
8.                 tasks = {t_1,....t_n Tasks|t_i ϵ cot.tasks}
9.                 remove tasks from cot.tasks
10.                VM = findFreeVM(VMT_i)
11.                if vm = = null then
12.                   Vm =provisionNewVM(VMT)
13.                end if
14.                scheduleTasks(task,vm)
15.             end for
16.          end for
17.       else
18.          task = cot.tasks[0]
19.          vm = findfreeVMForTask(task, budget);
20.          if vm == null then
21.             vm = provisionNEWVM(VMT_i)
22.          end if
23.          scheduleTask(task,vm)
24.       end if
25.    end for
26. end Procedure
```

**Algorithm 2** CoT scheduling

As previously stated, the clusters of pipelines *CoP* set is scheduled by the same approach as *CoT*. Pipelines, like tasks, have a budget and a capacity (the actual capacity of all pipeline tasks). As a result, we may use the similar scheduling strategy and describe the problem as a version of UKP with a little modification in how a knapsack element is defined. With respect to pipelines, $SKI_j = (VMT_j, NP_j, C_j)$, where the load of an element $NP_j$ is equal to total count of pipelines a VM type could execute within their budget. Finally, clusters of a single pipeline are executed as specific scenarios and are scheduled to idle VMs if they could execute them without violating their budget or to a new provisioned VM that could execute them without violating their budget with minimum makespan.

At last, we regard the scenario of a cluster of a single task $CoT_{single}$ as specific scenario. Single tasks are executed on idle VMs if they could meet the budget and even prior to the end of their existing billing cycle. If no idle VMs are available, a new VM of model that is suitable of executing the task by its budget at the minimum makespan is leased and the task is assigned to it. If there is no such model of VM could meet the budget, the speediest availability VM type is deployed and execute the task on it.

To better respond to the uncertainties that arise from performance variability and unanticipated delays at the time of execution, the BPS has a control mechanism in place to modify sub-budget allocation if a task's execution cost exceeded its allocated budget. This approach adds an extra budget parameter to hold the balance sub-budget obtained through the available cost execution. Whenever a task has been completed, the algorithm uses Eq. 9 to compute the available cost of execution and reallocates the balance sub-budget

to unexecuted tasks. If the available cost exceeding the allocated sub-budget, the difference will be deducted from the unexecuted task sub-budget. As a result, every time a task is completed, the budget reschedule (i.e., budget reallocation) will occur. Therefore, the uncertainties that a specific task encounters (e.g., performance variability, unanticipated delays) do not spread to the rest of the tasks. The description of this approach are defined in Algorithm 3.

```
procedure RESHEDULEBUDGET(T)
    t_f = completed task
    T_c = collection of unscheduled t ϵ T
    β_c = total sum of t.budget, where t ϵ T_c
    T_u = collection of unscheduled t ϵ (T - T_c)
    β_u = total sum of t.budget, where t ϵ T_u
    eb = extra budget
    if C_vmt^tf ≤ (tf.budget + eb) then
        eb = (t_f.budget + eb) − C_vmt^tf
    else
        balance budget = C_vmt^tf − (t_f.budget + eb)
        β_c = β_c − balance budget
        DISTRIBUTEBUDGET(β_c , T_c)
        if β_c < 0 then
            β_u = β_u + β_c
    DISTRIBUTEBUDGET(β_u , T_u )
end procedure
```

**Algorithm 3** Budget re-schedule

Eventually, BPS deactivates a VM if its billing period is nearing and it has no tasks scheduled to it. A computation of the VM deprovisioning delay is provided to confirm the VM deactivation signal is sent immediately hence that it doesn't get billed again when the current billing cycle is over.

## 5 Experimental results

Four popular workflow applications across multiple engineering fields were used to evaluate BPS: Epigenomics relating to bioinformatics discipline, CyberShake relating to physics discipline, Montage relating to astronomy discipline, and LIGO relating to astrophysics discipline. The CyberShake, Epigenomics, Montage, and LIGO are defined in Fig. 1b, c, d, and e. Every scientific workflow has a unique structural features as well as computational and data attributes. Their specification and representation is made available by [51].

The efficiency of the schedules made by BPS was evaluated using two algorithms. One of these is the Normalization-based Budget-constraint Workflow Scheduling NBWS [49], made for IaaS environment. It has an auto-scaled element that distributes and terminates VMs according to the existing state of tasks. It was made to schedule multi-workflows budget constrained problem. We tailored NBWS to run a single workflow with a few basic changes. It has two optimization phases; first one is task-selection, and next one is resource selection. It then picks the most cost-efficient form of VM for every task with respect to the budget. At every scheduling cycle, this algorithm is refined the schedule, and it specifies

how many VMs of each model are required to complete tasks within budget with minimum makespan. The goal is to show how BPS' dynamic component makes it to provide efficient schedules than NBWS.

Next one is the Constrained Based-Decreased Time CB-DT [50], made for IaaS environment. It distributes sub-budget to every task using back-tracking technique and schedule them onto already provisioned or newly provisioned VMs to reduce makespan. It was considered since it is a static technique with the capability to make a near optimal solution. It has also two optimization phases; first one is obtaining acceptable schedule, and next one is improving the acceptable schedule. Its major drawback is its inefficiency to fulfill budgets when uncertainties occur in cloud. However, we are particularly focused in contrasting BPS to CB-DT when both are able to fulfill budget constraints. Moreover, we may further test our solution's scalability by comparing them, demonstrating how BPS succeeds while NBWS and CB-DT failed to adapt from unanticipated delays.

Additionally, both comparison algorithms of NBWS [49] and CB-DT [50] do not consider cloud basic feature of VM performance variation, and VM provisioning and de-provisioning delays.

We deployed WorkflowSim [57] to setup the simulation of cloud framework. By WorkflowSim, we created a single IaaS distributor with four VM instances. Table 3 shows the VM instance setups according to the Google Cloud Platform (GCP) offerings. A 60-second pricing cycle was adopted, as provided by tech giants like GCP. The provisioning delay was fixed to 30 seconds [58] for all VM types, while the deprovisioning delay was fixed to 3 seconds [7]. The results of Leitner and Cito [6] were used to model CPU Performance variability. The VM processing power is minimized nearly 24% according to a normal distribution by a 12% mean along with a 10% standard deviation. The maximum capacity of a network bandwidth interface is distributed within all data transmissions through the interface during any specific time. To represent bottleneck and data transfer rate degradation, the progressive filling network system [59] was used to allocate bandwidth. A GS was also modelled with maximum writing and reading speeds. The number of activities constantly reading from the GS determines the reading speed of a specific transfer, and relatively similar logic can be applied to writing speed. Like so, bottleneck in the GS is modelled.

The evaluation was conducted using workflows with roughly 1000 tasks. We consider that computation of task size is not 100% optimal, thus we include ±10% variance in the size of every task in our simulation according to a, uniform distribution. The evaluations were made by three various budgets, $\beta_{w1}$ would be the hardest one, $\beta_{w2}$ and $\beta_{w3}$ would be the average and flexible one. With respect to every workflow, $\beta_{w1}$ is the cost of processing the workflow's critical path tasks plus the cost of transferring all of the input files into GS

**Table 3** Types of VM based on Google compute engine offering

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

and all of the output files from GS. The rest of the budgets are made on $\beta_{w1}$ and the size of the parameter $\beta_{int} = \beta_{w1} / 2$: $\beta_{w2} = \beta_{w1} + \beta_{int}$ and $\beta_{w3} = \beta_{w2} + \beta_{int}$. The graphs shown are the average of the findings obtained after repeating every evaluation 25 times.

### 5.1 Analysis and results

Budget and Makespan Evaluation: The purpose of these evaluation is to see how well the algorithms perform in terms of cost and time to achieve. The ability of an algorithm to fulfill a budget constraint by the workflow's cost to budget ratio. As a result, cost ratio more than one imply a cost bigger than the budget, a cost close to one imply a cost identical to the budget, and a cost less than one imply a cost less than the budget. The simulations carried out 25 times for every workflow, every budget interval, and every algorithm.

In Fig. 4, for CyberShake, the hard budget constraint is impossible to meet for NBWS and CB-DT. BPS proves its capability to handle unanticipated delays by being the only algorithm capable of keeping under this budget. In terms of makespan, BPS consistently beats the other algorithms for the remaining of the budget intervals. With respect to $\beta_{w2}$ and $\beta_{w3}$, BPS not only gets the best makespan, but also the best price (achieves less cost).

In Fig. 4, for Epigenomics, BPS is satisfied the three different budget criteria, and NBWS is satisfied the last two different budget criteria, while CG not meets any of the three budget criteria. BPS consistently gets the shortest time to completion among those algorithms that finish under budget. This outcomes prove once again the ability of the makespan-minimizing strategies in BPS.

In Fig. 4, for Montage, the hard budget constraint is far too difficult to satisfy for NBWS and CB-DT. On the other hand, BPS' ratio is significantly lower than the other algorithms' ratios. By obtaining shorter makespans regarding three budget intervals, BPS outperforms among other algorithms capable of fulfilling the budget. Most of the algorithms fulfill the required budget interval, with BPS and NBWS generating fairly similar makespans that are far less than those acquired by CB-DT once again.

In Fig. 4, for LIGO, for all five budget intervals, BPS is the only algorithm capable of reaching a ratio less than one. With respect to second and third budget intervals, the mean ratio obtained by NBWS is below one, whereas CB-DT fails to satisfy the budget in all three cases. BPS produces a shorter makespan in every situation where BPS and NBWS satisfy the budget, representing its capacity to make high-quality schedules.

Generally, BPS algorithm is the most effective at fulfilling budgets. It meets the constraint in most of the cases on average, compared to NBWS and CB-DT. These outcomes are accurate with what was expected about every algorithm. The static algorithm of CB-DT fails to meet budget, whereas the dynamic nature of BPS and NBWS, helps them to achieve their objectives more frequently. The evaluation also shows that BPS is capable of producing low cost solution. In all circumstances, it outperforms NBWS and CB-DT; BPS has the cheapest cost in comparison to the algorithms that fulfilled the budget. And these solutions demonstrate the effectiveness of BPS' DAG-pre-optimization contribution to improvement, which means partitioning the DAG into clusters and execute on same VM to reduce the data transfer time, makespan, and execution cost. An additional beneficial feature of BPS'

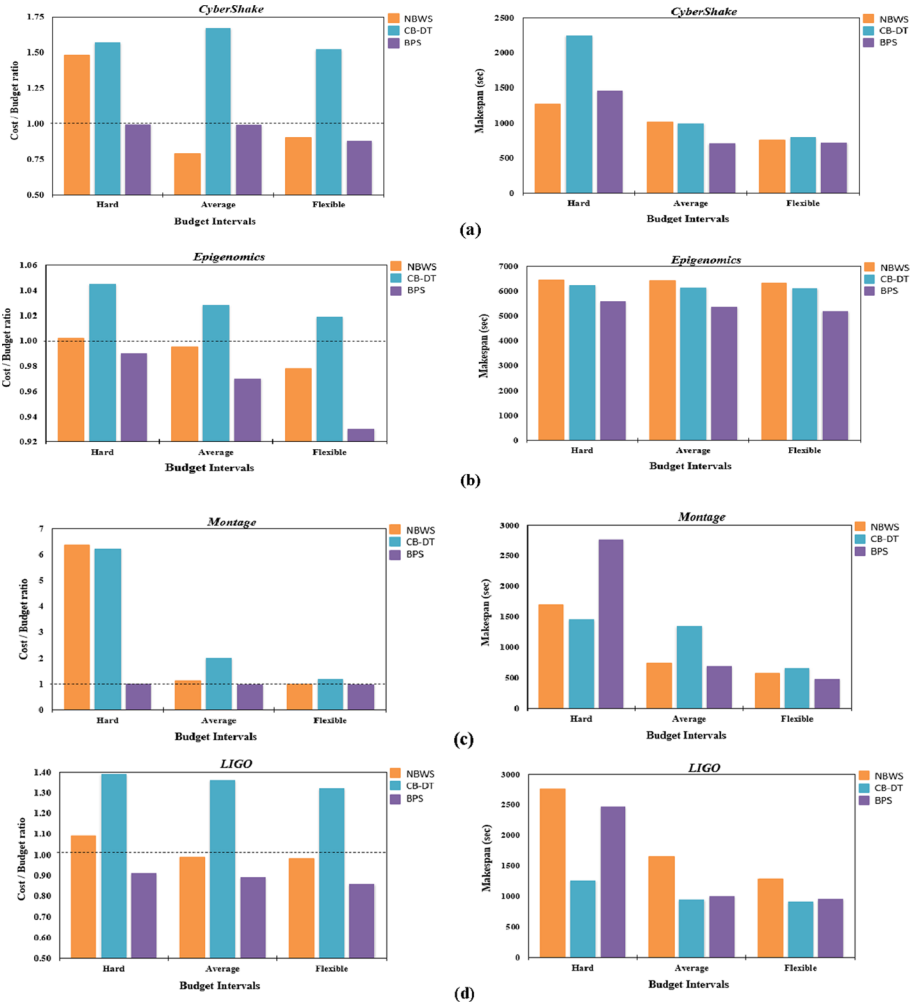


**Fig. 4** Budget and makespan evaluation. The three column in the budget graph defines the three budget types and represented its makespan under cost/budget ratio graph (**a**) CyberShake workflow budget and makespan (**b**) Epigenomics workflow budget and makespan (**c**) Montage workflow budget and makespan (**d**) LIGO workflow budget and makespan

revealed by the data is its capability to continuously reduce the amount of time required to execute the workflow while its budget increases. Moreover, NBWS and CB-DT do not consider the cloud basic feature of VM performance variation, and VM provisioning and de-provisioning delays during the workflow scheduling. If those algorithms considered that cloud basic features, it would have increased the cost and makespan than the cost and makespan it generates in Fig. 3.

### 5.2 Evaluating network usage

In cloud platforms, network bandwidth interfaces are still facing bottleneck problems. For e.g., the findings [4, 60] define a 65% variation in data transmission time in Google compute cloud. As a result, as a manner of eliminating the causes of variability and optimizing the efficiency of workflow execution, it is necessary for schedulers to aim to minimizing the number of data transmission over the cloud network technology. In this context, the number of files read from the GS according to each algorithm is evaluated. It's worth noting that a task doesn't need to be read by the GS if the input files it needs are early present with the VM where it will run.

Figure 5 represents the actual number of files retrieved for each workflow and algorithm throughout the three different types of budget. The referenced line refer to the number of files that the considered workflow retrieves as input. By partitioning the DAG into the cluster of homogeneous tasks, the cluster of pipeline tasks, and the cluster of a single task set, and executing them in a single VM or re-use already provisioned VM, BPS is efficient in minimizing the amount of files retrieved from GS for each workflow (CyberShake, Epigenomics, LIGO and Montage) than the comparison algorithm of NBWS and CB-DT. For execution, NBWS input file retrieval process is higher than BPS due to the utilization of already provisioned VMs partially and partially the newly provisioned VMs for each workflow task. At last, CB-DT input file retrieval process from GS is higher than BPS and NBWS due to the execution of mostly newly provisioned VMs and fail to reutilize the already provisioned VMs for each workflow task. Overall, BPS optimizes workflow structure by DAG pre-optimization and executes parents and child tasks are mostly run on

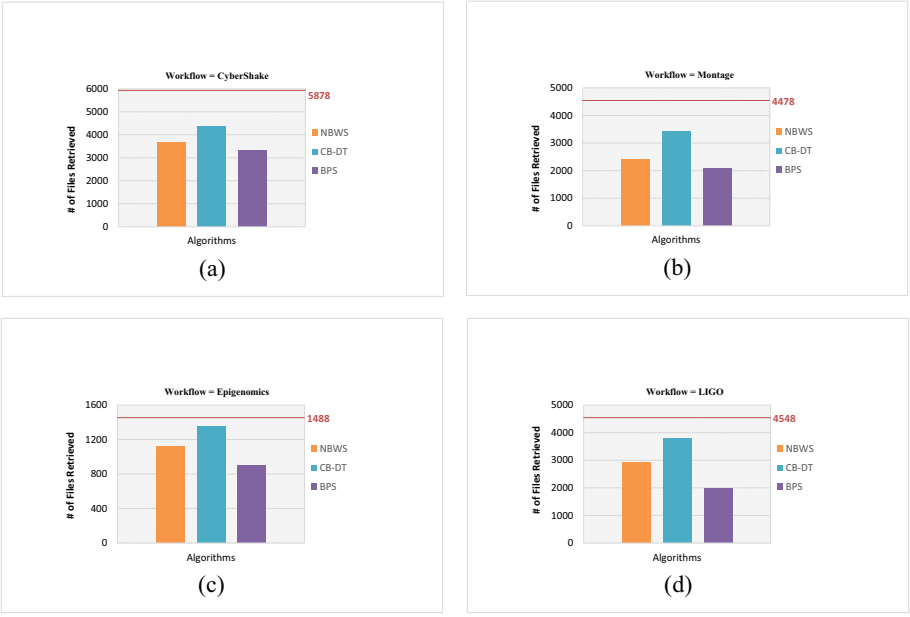


**Fig. 5** Actual number of files retrieved from the GS according to each algorithm. The referenced line refer to the number of files that the considered workflow requires as input. **a** CyberShake (**b**) Epigenomics (**c**) LIGO (**d**) Montage

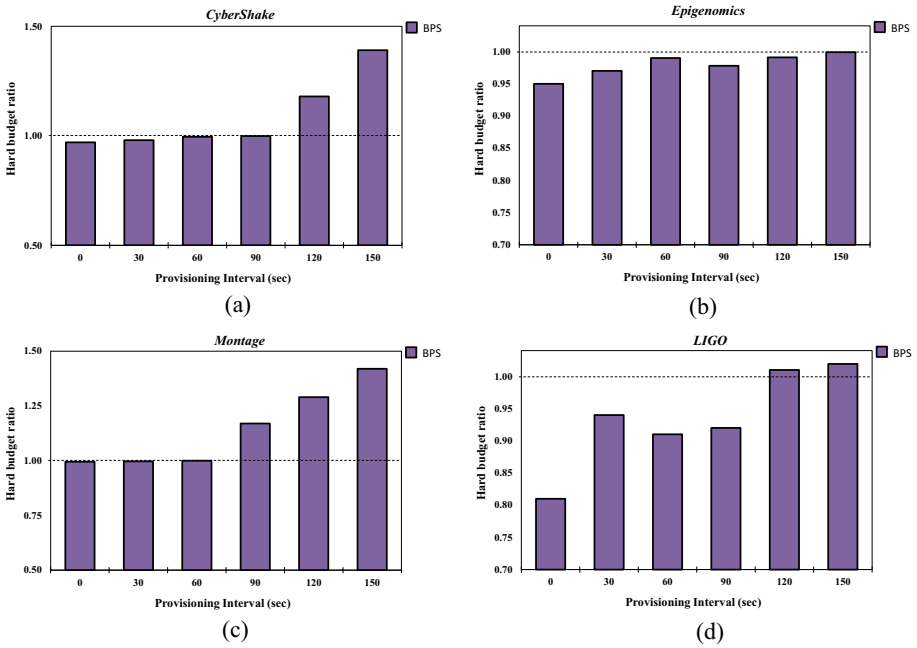


**Fig. 6** Hard budget ratio acquired for every considered workflow with various provisioning intervals

same VM to reduce the retrieval of input files and reduce the makespan while minimizing the budget for each workflow.

### 5.3 Responsiveness to provisioning delay

As workflow application execution facilitate more VM provisioning activities, it is necessary to consider BPS abilities to perform execution with a reasonable cost no bigger than the considered budget over different provisioning intervals. The interval were differed from 0 to 6 charging slots (150 s). Figure 6 represents the cost to budget ratios are computed for each workflow application over hard budget ratio.

The CyberShake workflow's last two ratio parameters have been exceeded. This occurs because the algorithm budget seems to be too hard, and when the provisioning interval rises, ratio parameters also elevate. According to Epigenomics workflow, all ratio values are smaller than one, illustrating BPS capacity to adjust to getting high provisioning intervals only if budget permits it. As respects to the Montage application, maximum ratio values larger than one apart from the last three. This is due to a too much hard budget for the BPS algorithm to execute within budget with high provisioning intervals. At last, in terms of the LIGO workflow, the significant number of the ratio values are smaller than one over all provisioning intervals, with the exception of final two factors of 120 and 150. The obtained provisioning interval ratio values are marginally larger than one. Since it turns too hard to comply with such long provisioning intervals, BPS algorithm is not fulfilling the budget standards.

### 5.4 Responsiveness to performance variation

Schedulers must be aware of performance fluctuation in order to avoid from unanticipated delays and meet QoS standards. The cost-to-budget ratio at various degraded parameters was used to evaluate the algorithm's responsiveness to VM CPU performance fluctuation, calculated by a normal distribution with a 1% variance and respective distinctive parameters. The estimated parameters were computed by taking halve of the greatest CPU power degradation, ranges from 0% to 60%.

Figure 7 depicts the acquired results. Regarding CyberShake workflow, majority of the ratio parameters are relatively close to one by efficient dynamic provisioning and scheduling of BPS, and surpass the parameters of 50% and 60% performance deterioration caused by the given budget is too hard. Relating to Epigenomics and Ligo workflows, most parameter values are smaller than one apart from the last one, which represents 60% degradation caused by hard budget scenario. In relation to Montage workflow, attain the ratio parameters less than one at 0%, 10%, 30% performance deterioration and exceed the budget ratio parameter at 20%, 40% 50% and 60% deterioration caused by the hard budget scenario.

Next potential factor that could force the budget to be violated for is that BPS makes a dynamic provisioning policy for CoTs, CoPs and $CoT_{single}$ with multiple tasks. Though that helps the algorithm to make efficient optimization selection in order to reduce the time it takes for the workflow applications to complete, it also facilitates its ability to respond to

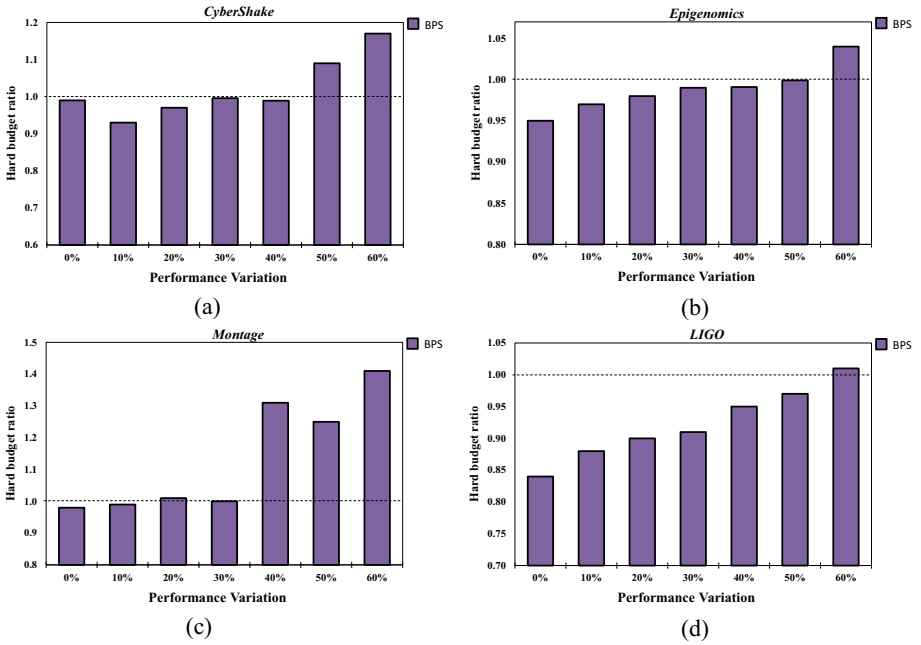


**Fig. 7** Hard budget ratio acquired for every considered workflow with various performance variation

the environmental dynamics. These estimates reveal that BPS remains efficient in meeting its budget target in the majority of the situations.

Overall, compare with NBWS and CB-DT, BPS first splits the workflow into three topological parts according to their characteristics at each level and executes the tasks at each level frequently on VMs which have the input data it claims (i.e., precedent and later tasks are mostly run on same VMs) with its efficient resource provisioning and scheduling plan considered with VM provisioning interval, performance variation and resource abundance features while NBWS executes the tasks partially on same VMs which does have the input file it claims (i.e., precedent and later tasks are partially run on different VMs by its strategy of linear and non-linear graph) but it does not consider the significant IaaS cloud features of VM provisioning interval, performance variation and resource abundance. CB-DT executes the tasks on VMs which doesn't have the input file it claims (i.e., precedent and later tasks are mostly run on different VMs) as well as lease new VMs during the process of scheduling due to the ignorance of resource abundance, hence demanding in higher costs, lower utilization rate, and high makespan with more delays such as VM provisioning interval.

Considerably, BPS is the topmost achievable algorithm in satisfying the deadline constraints by attaining its aim in all of the conditions. This shows the significance of customizing an algorithm to investigate the fundamental cloud components to utilize benefits of the consideration provided by the platform and satisfy the QoS specifications. The evaluations also shows the effectiveness of BPS in making higher-standard schedules by attaining a lesser execution cost in every scenario. These outcomes emphasize the effectiveness of the time-optimization plans adopted by BPS. Another worthwhile feature of BPS that can be noticed from the outcomes is its capability to repeatedly reducing the time it needs to execute the workflow while the deadline increases. The significance of this considers in the information that numerous users are ready to quid-pro-quo makespan for lesser cost during which remaining users are ready to take care of larger cost for quicker executions. The algorithm focus to perform in under this logic with the aim of the deadline value considered by users to be reasonable.

## 6 Conclusion and future work

BPS was offered the solution to make high standard schedules. Its aim include shortening the makespan as well as fulfilling a user-specified budget. The algorithm was efficient to some extent to adapt to the uncertain delays and dynamics that are typical in cloud computing. It also included dynamic provisioning, which enabled it build efficient schedules for a cluster of workflow tasks. By optimizing the workflow into cluster of pipelines, cluster of homogeneous tasks, and cluster of a single task that split a budget, we were capable to represent their scheduling as a UKP and find a solution through dynamic programming within pseudo-polynomial timespan.

The experimental events reveal that our findings outperformed other advanced budget-aware algorithms in terms of overall performance. And it fulfilled budget under uncertain scenarios such as VM provisioning delay, network congestion, VM performance variation, and indefinite task-size computations. It fulfilled this at lesser cost, even cheaper than the scheduling algorithms that may optimize the overall workflow system and compare multiple solutions before executing the workflow.

As future work, we want to execute the workflow consisting of heterogeneous tasks at every instance, like SIPHT and results, using several optimization methodologies, such as the GA algorithm and comparing their efficiency with BPS. Additionally, we want to expand the resource model to consider the cost of data communication between various data locations so that VMs could be adopted on multiple datacentres. Moreover, a rescheduling approach for multiple tasks CoTs will be analysed with the objective of further minimizing the significance of performance degradation. Various budget allocation approaches in addition to a greater scalable proposal to schedule heterogeneous CoTs will also be analyzed. Finally, we want to evaluate BPS with real-time tasks.

**Author contribution** Rajasekar P contributed to technical, conceptual and mathematical model (Unbounded Knapsack Problem) of this paper, and Santhiya P contributed to guidance and counselling on writing of this paper**.**

**Funding** No funding was received.

**Data availability** The data that support the findings of this study are available from the corresponding author [Rajasekar p], upon reasonable request.

## Declarations

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

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

## References

1. Menaka M, Kumar KS (2022) Workflow scheduling in cloud environment–Challenges, tools, limitations & methodologies: a review. Measurement: Sensors 100436. https://doi.org/10.1016/j.measen.2022.100436
2. Stergiou C, Psannis KE, Kim BG, Gupta B (2018) Secure integration of IoT and cloud computing. Futur Gener Comput Syst 78:964–975
3. Lin W, Xu S, He L, Li J (2017) Multi-resource scheduling and power simulation for cloud computing. Inf Sci 397:168–186
4. Prakash V, Bawa S, Garg L (2021) Multi-dependency and time based resource scheduling algorithm for scientific applications in cloud computing. Electronics 10(11):1320
5. Prakash V, Bala A (2014) July. A novel scheduling approach for workflow management in cloud computing. In 2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014) (pp. 610-615). IEEE
6. Doostali S, Babamir SM, Eini M (2021) CP-PGWO: multi-objective workflow scheduling for cloud computing using critical path. Clust Comput 24(4):3607–3627
7. Garg N, Singh D, Goraya MS (2021) Energy and resource efficient workflow scheduling in a virtualized cloud environment. Clust Comput 24:767–797
8. Xue S, Peng Y, Xu X, Zhang J, Shen C, Ruan F (2019) DSM: a dynamic scheduling method for concurrent workflows in cloud environment. Clust Comput 22:693–706
9. Mousavi Nik SS, Naghibzadeh M, Sedaghat Y (2021) Task replication to improve the reliability of running workflows on the cloud. Clust Comput 24:343–359
10. Taghinezhad-Niar A, Pashazadeh S, Taheri J (2022) QoS-aware online scheduling of multiple workflows under task execution time uncertainty in clouds. Clust Comput 25(6):3767–3784
11. Patra SS (2018) Energy-efficient task consolidation for cloud data center. Int J Cloud Appl Comput (IJCAC) 8(1):117–142
12. Lin W, Xu S, Li J, Xu L, Peng Z (2017) Design and theoretical analysis of virtual machine placement algorithm based on peak workload characteristics. Soft Comput 21(5):1301–1314

13. Leitner P, Cito J (2016) Patterns in the chaos—a study of performance variation and predictability in public iaas clouds. ACM Trans Internet Technol (TOIT) 16(3):1–23
14. Mao M, Humphrey M (2012) A performance study on the vm startup time in the cloud. In 2012 IEEE Fifth International Conference on Cloud Computing, pp. 423–430. IEEE
15. Ahmad W, Alam B, Ahuja S, Malik S (2021) A dynamic VM provisioning and de-provisioning based cost-efficient deadline-aware scheduling algorithm for big data workflow applications in a cloud environment. Clust Comput 24(1):249–278
16. Toussi GK, Naghibzadeh M (2021) A divide and conquer approach to deadline constrained cost-optimization workflow scheduling for the cloud. Clust Comput 24(3):1711–1733
17. Sun T, Xiao C, Xu X (2019) A scheduling algorithm using sub-deadline for workflow applications under budget and deadline constrained. Clust Comput 22(3):5987–5996
18. Iranmanesh A, Naji HR (2021) DCHG-TS: a deadline-constrained and cost-effective hybrid genetic algorithm for scientific workflow scheduling in cloud computing. Clust Comput 24(2):667–681
19. Geng X, Mao Y, Xiong M, Liu Y (2019) An improved task scheduling algorithm for scientific workflow in cloud computing environment. Clust Comput 22(3):7539–7548
20. Saeedizade E, Ashtiani M (2021) DDBWS: a dynamic deadline and budget-aware workflow scheduling algorithm in workflow-as-a-service environments. J Supercomput 77:14525–14564. https://doi.org/10.1007/s11227-021-03858-6
21. Deldari A, Naghibzadeh M, Abrishami S (2017) CCA: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud. J Supercomput 73(2):756–781
22. Khorsand R, Safi-Esfahani F, Nematbakhsh N, Mohsenzade M (2017) ATSDS: adaptive two-stage deadline-constrained workflow scheduling considering run-time circumstances in cloud computing environments. J Supercomput 73(6):2430–2455
23. Wu F, Wu Q, Tan Y, Li R, Wang W (2016) PCP-B2: partial critical path budget balanced scheduling algorithms for scientific workflow applications. Futur Gener Comput Syst 60:22–34
24. Medara R, Singh RS, Sompalli M (2022) Energy and cost aware workflow scheduling in clouds with deadline constraint. Concurr Comput: Pract Experience e6922. https://onlinelibrary.wiley.com/doi/epdf/10.1002/cpe.6922
25. Liu L, Zhang M, Buyya R, Fan Q (2017) Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing. Concurrency and Computation: Practice and Experience 29(5):e3942
26. Nirmala SJ, Bhanu SMS (2016) Catfish-PSO based scheduling of scientific workflows in IaaS cloud. Computing 98(11):1091–1109
27. Verma A, Kaushal S (2017) A hybrid multi-objective particle swarm optimization for scientific workflow scheduling. Parallel Comput 62:1–19
28. Alkhanak EN, Lee SP (2018) A hyper-heuristic cost optimisation approach for scientific workflow scheduling in cloud computing. Futur Gener Comput Syst 86:480–506
29. Nik SSM, Naghibzadeh M, Sedaghat Y (2020) Cost-driven workflow scheduling on the cloud with deadline and reliability constraints. Computing 102(2):477–500
30. Saeedi S, Khorsand R, Bidgoli SG, Ramezanpour M (2020) Improved many-objective particle swarm optimization algorithm for scientific workflow scheduling in cloud computing. Comput Ind Eng 147:106649
31. Zhou N, Lin W, Feng W, Shi F, Pang X (2020) Budget-deadline constrained approach for scientific workflows scheduling in a cloud environment. Clust Comput 1–15. https://doi.org/10.1007/s10586-020-03176-1
32. Rodriguez MA, Buyya R (2017) Budget-driven scheduling of scientific workflows in IaaS clouds with fine-grained billing periods. ACM Trans Auton Adapt Syst (TAAS) 12(2):1–22
33. Arabnejad V, Bubendorfer K, Ng B (2016) October. Budget distribution strategies for scientific workflow scheduling in commercial clouds. In 2016 IEEE 12th International Conference on e-Science (e-Science) (pp. 137-146). IEEE
34. Arabnejad H, Barbosa JG (2014) A budget constrained scheduling algorithm for workflow applications. J Grid Comput 12(4):665–679
35. Arabnejad V, Bubendorfer K, Ng B (2018) Budget and deadline aware e-science workflow scheduling in clouds. IEEE Trans Parallel Distributed Syst 30(1):29–44
36. Chen W, Xie G, Li R, Bai Y, Fan C, Li K (2017) Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems. Futur Gener Comput Syst 74:1–11
37. Arabnejad H, Barbosa JG (2017) Maximizing the completion rate of concurrent scientific applications under time and budget constraints. J Comput Sci 23:120–129
38. Arabnejad H, Barbosa JG (2017) Multi-QoS constrained and profit-aware scheduling approach for concurrent workflows on heterogeneous systems. Futur Gener Comput Syst 68:211–221

39. Hilman, M.H., Rodriguez, M.A. and Buyya, R., (2017) October. Task-based budget distribution strategies for scientific workflows with coarse-grained billing periods in iaas clouds. In 2017 IEEE 13th International Conference on e-Science (e-Science), pp. 128–137. IEEE
40. Hilman MH, Rodriguez MA, Buyya R (2019) Resource-sharing Policy in Multi-tenant Scientific Workflow-as-a-Service Cloud Platform. arXiv preprint arXiv:1903.01113
41. Taghinezhad-Niar A, Pashazadeh S, Taheri J (2021) Workflow scheduling of scientific workflows under simultaneous deadline and budget constraints. Clust Comput 24(4):3449–3467
42. Zeedan M, Attiya G, El-Fishawy N (2023) Enhanced hybrid multi-objective workflow scheduling approach based artificial bee colony in cloud computing. Computing 105(1):217–247
43. Stavrinides GL, Karatza HD (2021) Dynamic scheduling of bags-of-tasks with sensitive input data and end-to-end deadlines in a hybrid cloud. Multimed Tools Appl 80(11):16781–16803
44. Stavrinides GL, Karatza HD (2019) A hybrid approach to scheduling real-time IoT workflows in fog and cloud environments. Multimed Tools Appl 78(17):24639–24655
45. Rajasekar P, Palanichamy Y (2021) Adaptive resource provisioning and scheduling algorithm for scientific workflows on IaaS cloud. SN Comput Sci 2:1–16
46. Rajasekar P, Palanichamy Y (2022) A flexible deadline-driven resource provisioning and scheduling algorithm for multiple workflows with VM sharing protocol on WaaS-cloud. J Supercomput 78:8025–8055
47. Rajasekar P, Palanichamy Y (2021) Scheduling multiple scientific workflows using containers on IaaS cloud. J Ambient Intell Humaniz Comput 12:7621–7636
48. Rodriguez MA, Buyya R (2018) Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms. Futur Gener Comput Syst 79:739–750
49. Chakravarthi KK, Shyamala L, Vaidehi V (2020) Budget aware scheduling algorithm for workflow applications in IaaS clouds. Clust Comput 23(4):3405–3419
50. Ghafouri R, Movaghar A, Mohsenzadeh M (2019) A budget constrained scheduling algorithm for executing workflow application in infrastructure as a service clouds. Peer-to-Peer Netw Appl 12(1):241–268
51. Bharathi S, Chervenak A, Deelman E, Mehta G, Su MH, Vahi K (2008) Characterization of scientific workflows. In 2008 third workshop on workflows in support of large-scale science, pp. 1–10. IEEE
52. Andonov R, Poirriez V, Rajopadhye S (2000) Unbounded knapsack problem: dynamic programming revisited. Eur J Oper Res 123(2):394–407
53. Andonov R, Rajopadhye S (1994) A sparse knapsack algo-tech-cuit and its synthesis. In Proceedings of IEEE International Conference on Application Specific Array Processors (ASSAP'94), pp. 302–313. IEEE
54. Gilmore PC, Gomory RE (1963) A linear programming approach to the cutting stock problem—part II. Oper Res 11(6):863–888
55. Gilmore PC, Gomory RE (1966) The theory and computation of knapsack functions. Oper Res 14(6):1045–1074
56. Rodriguez MA, Buyya R (2015) September. A responsive knapsack-based algorithm for resource provisioning and scheduling of scientific workflows in clouds. In 2015 44th International Conference on Parallel Processing, pp. 839–848. IEEE
57. Chen W, Deelman E (2012) Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In 2012 IEEE 8th international conference on E-science, pp. 1–8. IEEE
58. Stadill S (2013) By the numbers: How google compute engine stacks up to amazon ec2. Available: https://gigaom.com/2013/03/15/by-the-numbers-how-google-compute-engine-stacks-up-to-amazon-ec2/
59. Bertsekas DP, Gallager RG, Humblet P (1992) Data networks, vol 2. Prentice-Hall International, Hoboken
60. Jackson KR, Ramakrishnan L, Muriki K, Canon S, Cholia S, Shalf J, Wasserman HJ, Wright NJ (2010) November. 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, pp. 159–168. IEEE