Received: 24 November 2015 | Revised: 1 May 2016 | Accepted: 12 October 2016

DOI 10.1002/cpe.4041

WILEY

RESEARCH ARTICLE

A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments

Maria Alejandra Rodriguez | **Rajkumar Buyya**

Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia

Correspondence
Maria Rodriguez, Department of Computing and Information Systems, Doug McDonell Building, The University of Melbourne, Parkville 3010, VIC, Australia.
Email: marodriguez@unimelb.edu.au

Summary

Large-scale scientific problems are often modeled as workflows. The ever-growing data and compute requirements of these applications has led to extensive research on how to efficiently schedule and deploy them in distributed environments. The emergence of the latest distributed systems paradigm, cloud computing, brings with it tremendous opportunities to run scientific workflows at low costs without the need of owning any infrastructure. It provides a virtually infinite pool of resources that can be acquired, configured, and used as needed and are charged on a pay-per-use basis. However, along with these benefits come numerous challenges that need to be addressed to generate efficient schedules. This work identifies these challenges and studies existing algorithms from the perspective of the scheduling models they adopt as well as the resource and application model they consider. A detailed taxonomy that focuses on features particular to clouds is presented, and the surveyed algorithms are classified according to it. In this way, we aim to provide a comprehensive understanding of existing literature and aid researchers by providing an insight into future directions and open issues.

KEYWORDS

IaaS cloud, resource provisioning, scientific workflow, scheduling, survey, taxonomy

1 | INTRODUCTION

Various scientific fields use workflows to analyze large amounts of data and to run complex simulations and experiments efficiently. A process can be modeled as a workflow by dividing it into smaller and simpler subprocesses (i.e., tasks). These tasks can then be distributed to multiple compute resources for a faster, more efficient execution. The scheduling of workflow tasks in distributed platforms has been widely studied over the years. Researchers have developed algorithms tailored for different environments: from homogeneous clusters with a limited set of resources, to large-scale community grids, to the most recent paradigm, utility-based, heterogeneous, and resource-abundant cloud computing. This work focuses on the latter case; it studies algorithms developed to orchestrate the execution of scientific workflow tasks in cloud computing environments, in particular, Infrastructure as a Service (IaaS) clouds.

The IaaS clouds offer an easily accessible, flexible, and scalable infrastructure for the deployment of large-scale scientific workflows. They enable workflow management systems to access a shared compute infrastructure on-demand and on a pay-per-use basis.[1] This is done by leasing virtualised compute resources, called virtual machines (VMs), with a predefined CPU, memory, storage, and bandwidth capacity. Different resource bundles (i.e., VM types) are available at varying prices to suit a wide range of application needs. The VMs can be elastically leased and released and are charged per time frame, or billing period. The IaaS providers offer billing periods of different granularity, for example, Amazon EC2[2] charges per hour while Microsoft Azure[3] is more flexible an charges on a per-minute basis. Aside from VMs, IaaS providers also offer storage services and network infrastructure to transport data in, out, and within their facilities. The term cloud will be used to refer to providers offering the IaaS service model.

Several scheduling challenges arise from the multitenant, on-demand, elastic, and pay-as-you-go resource model offered by cloud computing. When compared to other distributed systems such as grids, clouds offer more control over the type and quantity of resources used. This flexibility and abundance of resources creates the need for a resource provisioning strategy that works together with the scheduling algorithm; a heuristic that decides the type and number of VMs to use and when to lease and to release them. Another challenge that must be addressed by scheduling algorithms is the utility-based pricing model of resources. Schedulers need to find a trade-off between performance, nonfunctional requirements, and cost

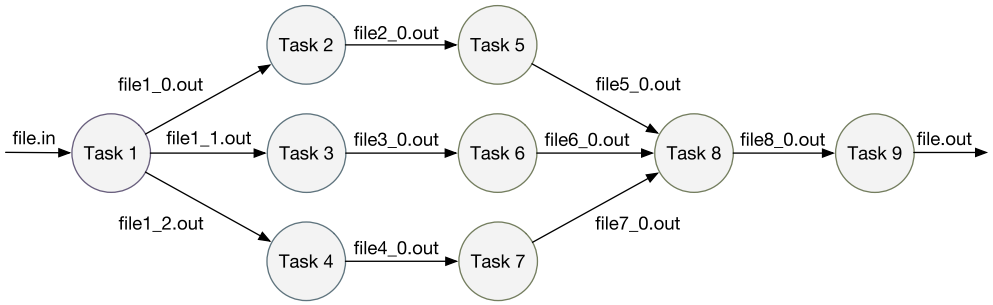

FIGURE 1 Sample workflow with nine tasks. The graph nodes represent computational tasks and the edges the data dependencies between these tasks

to avoid paying unnecessary and potentially prohibitive prices. Finally, algorithms need to be aware of the dynamic nature of cloud platforms and the uncertainties this brings with it because performance variation is observed in resources such as VM CPUs, network links, and storage systems. In addition to this, providers make no guarantees on the time it takes to provision and deprovision VMs, with these values being highly variable and unpredictable in practice. Schedulers need to be aware of this variability to recover from unexpected delays and achieve their performance and cost objectives.

In this survey, different characteristics of existing cloud workflow scheduling algorithms are analyzed. In particular, the scheduling, application, and resource models are studied. Because extensive research has been done on the scheduling field in general, widely accepted and accurate classifications already exist for these features. We extend and complement with a cloud-focused discussion those that are of particular importance to the studied problem. Some of these include the dynamicity of the scheduling and provisioning decisions, the scheduling objectives, and the optimization strategy. The application model is studied from the workflow multiplicity point of view, that is, the number and type of workflows that algorithms are capable of processing. Finally, classifications for the resource model are made on the basis of different cloud features and services such as storage and data transfer costs, pricing models, VM delays, data center and provider deployment models, and VM heterogeneity among others.

The rest of this paper is organized as follows. Section 2 introduces the notion of scientific workflows while Section 3 discusses the workflow scheduling problem and the challenges particular to cloud environments. The proposed classification system is presented in Section 4. A detailed discussion of outstanding algorithms is presented in Section 5 along with the classification of all the studied solutions. Finally, future directions and conclusions are described in Section 6.

2 | SCIENTIFIC WORKFLOWS: AN OVERVIEW

The concept of workflow has its roots in commercial enterprises as a business process modeling tool. These business workflows aim to automate and optimize the processes of an organization, seen as an ordered sequence of activities, and are a mature research area[4] lead by the workflow management coalition* (WfMC), founded in 1993. This notion of workflow has extended to the scientific community in which *scientific workflows* are used support large-scale, complex scientific processes; they are designed to conduct experiments and prove scientific hypotheses by managing, analyzing, simulating, and visualizing scientific data.[5] Therefore, even though both business and scientific workflows share the same basic concept, both have specific requirements and hence need separate consideration. In this survey we focus on scientific workflows, and from now on, we will refer to them simply as workflows.

A workflow is defined by a set of computational tasks with dependencies between them. In scientific applications, it is common for the dependencies to represent a data flow from one task to another; the output data generated by one task becomes the input data for the next one. Figure 1 shows a sample workflow with 9 tasks. These applications can be CPU, memory, or I/O intensive (or a combination of these), depending on the nature of the problem they are designed to solve. In a CPU intensive workflow most tasks spend most of their time performing computations. In a memory-bound workflow most tasks require high physical memory usage. The I/O intensive workflows are composed of tasks that require and produce large amounts of data and hence spend most of their time performing I/O operations.[6]

Scientific workflows are managed by different institutions or individuals in different fields meaning they have different requirements for the software needed by tasks to run. These characteristics make them great candidates to leverage the capabilities offered by cloud computing. Scientists can configure VM images to suit the software needs of a specific workflow, and with the help of scheduling algorithms and workflow management systems, they can efficiently run their applications on a range of cloud resources to obtain results in a reasonable amount of time. In this way, by providing a simple, cost-effective way of running scientific applications that are accessible to everyone, cloud computing is revolutionizing the way e-science is done.

Many scientific areas have embraced workflows as a mean to express complex computational problems that can be efficiently processed in distributed environments. For example, the Montage workflow[7] is an astronomy application characterized by being I/O intensive that is used to create custom mosaics of the sky on the basis of a set of input images. It enables astronomers to generate a composite image of a region of the sky that is too large to be produced by astronomical cameras or that has been measured with different wavelengths and instruments. During the workflow execution, the geometry of the output image is calculated from that of the input images. Afterwards, the input data is reprojected so that they have the same spatial scale and rotation. This is followed by a standardization of the background of all images. Finally,

*http://www.wfmc.org/

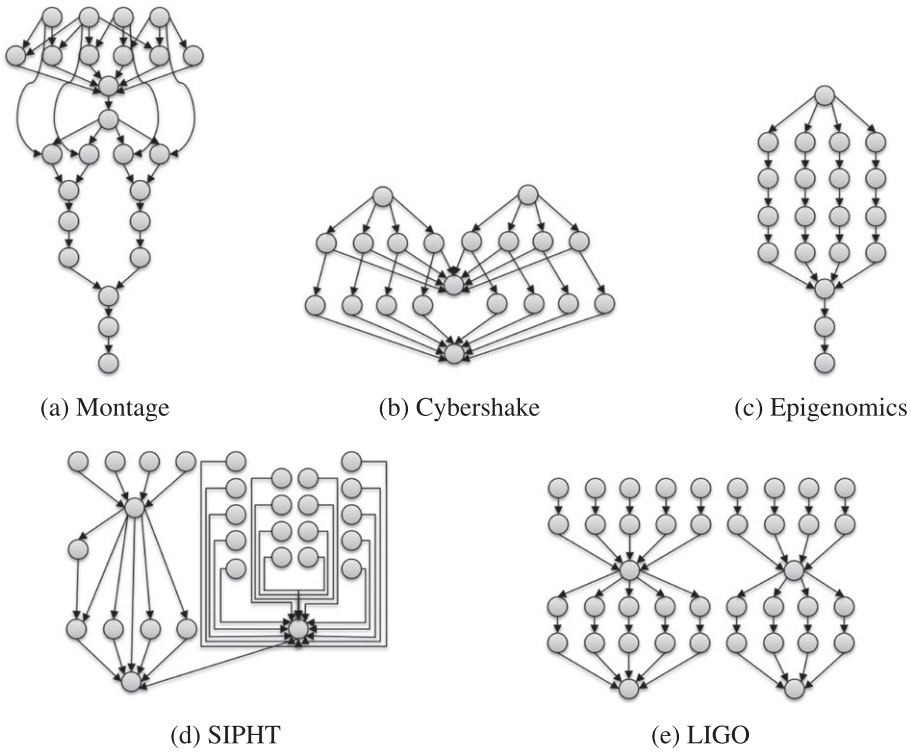

FIGURE 2 Structure of 5 different scientific workflows: Montage (Astronomy), Cybershake (Earthquake science), Epigenomics (Bioinformatics), SIPHT (Bioinformatics), and LIGO (astrophysics)

all the processed input images are merged to create the final mosaic of the sky region.

Another example of a workflow is Cybershake,[8] a data and memory intensive earthquake hazard characterization application used by the Southern California Earthquake Centre. The workflow begins by generating strain green tensors for a region of interest via a simulation. These strain green tensor data are then used to generate synthetic seismograms for each predicted rupture followed by the creation of acceleration and probabilistic hazard curves for the given region. Other examples include the laser interferometer gravitational wave observatory (LIGO),[9], SIPHT,[10] and Epigenomics[11] workflows. The LIGO is a memory intensive application used in the physics field with the aim of detecting gravitational waves. In bioinformatics, SIPHT is used to automate the process of searching for small RNA encoding genes for all bacterial replicons in the National Centre for Biotechnology Information[12] database. Also in the bioinformatics field, the Epigenomics workflow is a CPU intensive application that automates the execution of various genome sequencing operations.

The aforementioned applications are a good representation of scientific workflows as they are taken from different domains and together provide a broad overview of how workflow technologies are used to manage complex analyses. Each of the workflows has different topological structures all common in scientific workflows such as pipelines, data distribution, and data aggregation.[13] They also have varied data and computational characteristics, including CPU, I/O, and memory intensive tasks. Figure 2 shows the structure of these 5 scientific workflows, and their full characterization is presented by Juve et al.[6]

The scope of this work is limited to workflows modeled as directed acyclic graphs (DAGs), which by definition have no cycles or conditional dependencies. Although, there are other models of computation that could be used to express and process scientific workflows such as best effort, superscalar, and streaming pipelines, this survey focuses on DAGs as they are commonly used by the scientific and research community. For instance, workflow management systems such as Pegasus,[14] Cloudbus WfMS,[15] ASKALON,[16] and DAGMan[17] support the execution of workflows modeled as DAGs. We refer the readers to the work by Pautasso and Alonso[18] for a detailed characterization of different models of computation that can be used for optimizing the performance of large-scale scientific workflows.

Formally, a DAG representing a workflow application $W = (T, E)$ is composed of a set of tasks $T = \{t_1, t_2, \dots, t_n\}$ and a set of directed edges E. An edge e_{ij} of the form (t_i, t_j) exists if there is a data dependency between t_i and t_j, case in which t_i is said to be the parent task of t_j and t_j is said to be the child task of t_i. On the basis of this definition, a child task cannot run until all of its parent tasks have completed, and its input data are available in the corresponding compute resource.

3 | WORKFLOW SCHEDULING IN IAAS CLOUDS

In general, the process of scheduling a workflow in a distributed system consists of assigning tasks to resources and orchestrating their execution so that the dependencies between them are preserved. The mapping is also done so that different user-defined quality of service (QoS) requirements are met. These QoS parameters are generally defined in terms of performance metrics such as execution time and nonfunctional requirements such as security and energy consumption. This problem is NP-complete[19] in its general form, and there are only 3 special cases that can be optimally solved within polynomial time. The first one is the scheduling of tree-structured graphs with uniform computation

costs on an arbitrary number of processors.[20] The second one is the scheduling of arbitrary graphs with uniform computation costs on 2 processors,[21] and the third one is the scheduling of interval-ordered graphs.[22] The problem addressed in this work does not fit any of these 3 scenarios, and no optimal solution can be found in polynomial time.

To plan the execution of a workflow in a cloud environment, 2 subproblems need to be considered. The first one is known as *resource provisioning*, and it consists of selecting and provisioning the compute resources that will be used to run the tasks. This means having heuristics in place that are capable of determining how many VMs to lease, their type, and when to start them and shut them down. The second subproblem is the actual *scheduling* or *task allocation* stage, in which each task is mapped onto the best-suited resource. The term *scheduling* is often used to refer to the combination of these 2 subproblems by authors developing algorithms targeting clouds, and we follow the same pattern throughout the rest of this survey.

3.1 | Cloud workflow management system

The execution of workflows in clouds is done via a cloud workflow management system (CWfMS). It enables the creation, monitoring, and execution of scientific workflows and has the capability of transparently managing tasks and data by hiding the orchestration and integration details among the distributed resources.[23] A reference architecture is shown in Figure 3. The depicted components are common to most CWfMS implementations; however, not all of them have to be implemented to have a fully functional system.

User interface. The user interface allows for users to create, edit, submit, and monitor their applications.

Workflow engine. The workflow engine is the core of the system and is responsible for managing the actual execution of the workflow. The parser module within the engine interprets a workflow depicted in a high-level language such as XML and creates the corresponding internal workflow representation such as task and data objects. The scheduler and resource provisioning modules work together in planning the execution of the workflow. The resource provisioning module is responsible of selecting and provisioning the cloud resources, and the scheduling component applies specific policies that map tasks to available resources, both processes on the basis of the QoS requirements and scheduling objectives. The performance prediction and runtime estimation module uses historical data, data provenance, or time series prediction models, among other methods, to estimate the performance of cloud resources, and the amount of time tasks will take to execute in different VMs. These data are used by the resource provisioning and scheduling modules to make accurate and efficient decisions regarding the allocation of tasks. The data management component of the workflow engine handles the movement, placement, and storage of data as required for the workflow execution. Finally, the task dispatcher has the responsibility of interacting with the cloud APIs to dispatch tasks ready for execution onto the available VMs.

Administration and monitoring tools. The administration and monitoring tools of the CWfMS architecture include modules that enable the dynamic and continuous monitoring of workflow tasks and resource performance as well as the

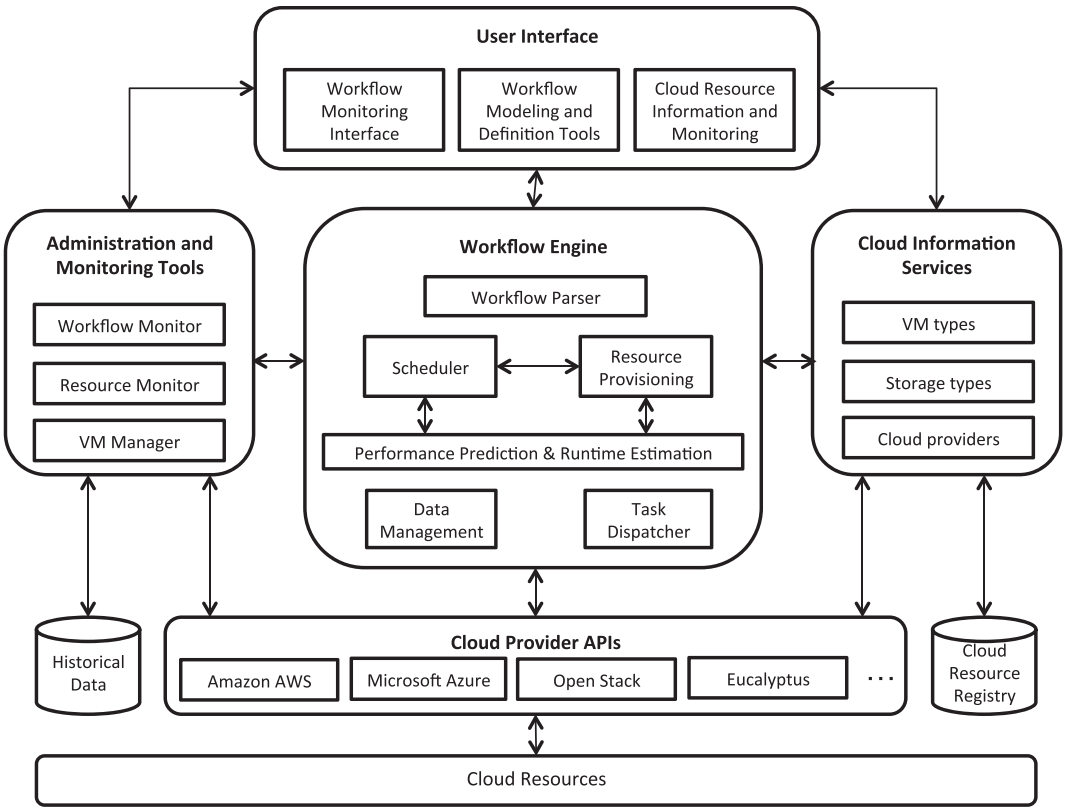

FIGURE 3 Reference architecture of a workflow management system

management of leased resources, such as VMs. The data collected by these tools can be used by fault tolerance mechanisms or can be stored in a historical database and used by performance prediction methods, for example.

Cloud information services. Another component of the architecture is the cloud information services. This component provides the workflow engine with information about different cloud providers, the resources they offer including their characteristics and prices, location, and any other information required by the engine to make the resource selection and mapping decisions.

Cloud provider APIs. These APIs enable the integration of applications with cloud services. For the scheduling problem described in this paper, they enable the on-demand provisioning and deprovisioning of VMs, the monitoring of resource usage within a specific VM, access to storage services to save and retrieve data, transferring data in or out of their facilities, and configuring security and network settings, among others. Most IaaS APIs are exposed as REST and SOAP services, but protocols such as XML-RPC and Javascript are also used. For instance, CloudSigma, Rackspace, Windows Azure, and Amazon EC2 all offer REST-based APIs. As opposed to providing services for a specific platform, other solutions such as Apache JClouds[24] aim to create a cross-platform cloud environment by providing and API to access services from different cloud providers in a transparent manner. Cross-platform interfaces have the advantage of allowing applications to access services from multiple providers without having to rewrite any code, but may have less functionality or other limitations when compared to vendor-specific solutions.

3.2 | Challenges

Scheduling algorithms need to address various challenges derived from the characteristics of the cloud resource model. In this section we discuss these challenges and the importance of considering them to leverage the flexibility and convenience offered by these environments. Other approaches developed for environments such as grids or other parallel platforms could be applied to scheduling workflows on IaaS clouds; however, they would fail to leverage the on-demand access to "unlimited" resources, lead to unnecessary costs by not considering the IaaS cost model, and fail to capture the characteristics of clouds of performance variation and sources of uncertainty.

Resource provisioning. The importance of addressing the resource provisioning problem as part of the scheduling strategy is demonstrated in several studies. The works by Gutierrez-Garcia and Sim,[25] Michon et al,[26] and Villegas et al[27] have demonstrated a dependency between both problems when scheduling bags of tasks (BoTs) in clouds while Frincu et al[28] investigated the impact that resource provisioning has on the scheduling of various workflows in clouds. They all found a relationship between the 2 problems and concluded that the VM provisioning strategy affects the cost and makespan (total execution time of the workflow) achievable by the scheduling strategy.

Choosing the optimal configuration for the VM pool that will be used to run the workflow tasks is a challenging problem. Firstly, the combination of VMs needs to have the capacity to fulfill the scheduling objectives while considering their cost. If VMs are underprovisioned, then the scheduler will not be able to achieve the expected performance or throughput. On the other hand, if VMs are overprovisioned, then the system's utilization will be low, resulting in capacity wastage and unnecessary costs.

Secondly, provisioners need to dynamically scale in and out their resource pool. They need to decide when to add or remove VMs in response the application's demand. This may aid in improving the performance of the application, the overall system utilization, and reducing the total infrastructure cost. In this way, workflow management systems have the ability to use resources opportunistically on the basis of the number and type of workflow tasks that need to be processed at a given point of time. This is a convenient feature for scientific workflows as common topological structures such as data distribution and aggregation[13] lead to significant changes in the parallelism of the workflow over time.

Finally, provisioners have a wide range of options when selecting the VM type to use. Clouds offer VM instances with varying configurations for compute, memory, storage, and networking performance. Different instance types are designed so that they are optimal for certain types of applications. For example, Amazon EC2 offers a family of compute-optimized instances designed to work best for applications requiring high compute power, a family of memory-optimized instances that have the lowest cost per GB of RAM and are best for memory intensive applications, and a storage-optimized instance family best suited for applications with specific disk I/O and storage requirements, among others.[2] Moreover, the price of VM instances varies with each configuration, and it does not necessarily increase linearly with an increase in capacity. While this large selection of VM types offers applications enormous flexibility, it also challenges algorithms to be able to identify not only the best resource for a given task, but also the optimal combination of different instance types that allow for the user's QoS requirements to be met.

Performance variation and other sources of uncertainty. Characteristics such as the shared nature, virtualisation, and heterogeneity of nonvirtualised hardware in clouds result in a variability in the performance of resources. For example, VMs deployed in cloud data centers do not exhibit a stable performance in terms of execution times.[29–33] In fact, Schad et al[29] report an overall CPU performance variability of 24% in the Amazon EC2 cloud. Performance variation is also observed in the network resources, with studies reporting a data transfer time variation of 19% in Amazon EC2.[29] Additionally, if resources are located in different data centers or under different providers, they may be separated by public internet channels with unpredictable behavior. In case of scientific workflows with large data dependencies, this may have a considerable impact in the workflow runtime. This indeterminism makes it extremely difficult for schedulers to estimate runtimes and make accurate scheduling decisions to fulfill QoS requirements.

A variability in performance has also been observed in other environments such as grids,[34–36] and several performance estimation techniques have been developed as a result. However, the public and large-scale nature of clouds makes this problem a more challenging one. For instance, to achieve a more accurate prediction of the runtime of a job in a specific VM, IaaS providers would have to allow access to information such as the type of host where the VM is deployed, its current load and utilization of resources, the overhead of the virtualisation software, and network congestion, among others. Because access to this information is unlikely for users, algorithms, especially those dealing with constraints such as budget and deadline, need to acknowledge their limitations when estimating the performance of resources and have mechanisms in place that will allow them to recover from unexpected delays.

Other sources of uncertainty in cloud platforms are VM provisioning and deprovisioning delays. A VM is not ready for use immediately after its request. Instead, it takes time for it to be deployed on a physical host and booted; we refer to this time as the VM provisioning delay. Providers make no guarantees on the value of this delay, and studies[29,30,37] have shown that it can be significant (in some providers more than others) and highly variable, making it difficult to predict or rely on an average measurement of its value. As an example, Osterman et al[30] found the minimum resource acquisition time for the m1.large Amazon EC2 instance to be 50 seconds and the maximum to be 883 seconds; this demonstrates the potential variability that users may experience when launching a VM. As for the deprovisioning delay, it is defined as the time between the request to shutdown the VM and the actual time when the instance is released back to the provider and stops being charged. Once again, IaaS vendors make no guarantees on this time, and it varies from provider to provider and VM type to VM type. However, Osterman et al[30] found that it has a lower variability and average value than the acquisition time, and hence, it may be slightly easier to predict and have a smaller impact on the execution of a workflow.

Utility-based pricing model. Schedulers planning the execution of workflows in clouds need to consider the cost of using the infrastructure. The use of VMs, as well as network and storage services, is charged for on a pay-per-use basis. Algorithms need to find a balance between performance, nonfunctional requirements, and cost. For example, some schedulers may be interested in a trade-off between execution time, energy consumption, and cost. It is not only this trade-off that adds to the scheduling complexity but also the already mentioned difficulty in predicting the performance of resources, which translates in a difficulty in estimating the actual cost of using the chosen infrastructure.

Additionally, some IaaS providers offer a dynamic pricing scheme for VMs. In Amazon's EC2 terminology for example, these dynamically priced VMs are known as spot instances, and their prices vary with time on the basis of the market's supply and demand patterns.[38] Users can acquire VMs by bidding on them, and their price is generally significantly lower than the "on-demand" price; however, they are subject to termination at any time if the spot price exceeds the bidding price. Offerings such as this give users the opportunity to use VMs at significantly lower prices, and scientific workflows can greatly benefit from this. But schedulers need to address challenges such as selecting a bid, determining the actions to take when a spot instance is terminated, and deciding when it is appropriate to use spot VMs versus statically priced, more reliable, ones.

4 | TAXONOMY

The scope of this survey is limited to algorithms developed to schedule workflows exculsively in public IaaS clouds. As a result, only those that consider a utility-based pricing model and address the VM provisioning problem are studied. Other scope-limiting features are derived from the application model, and all the surveyed algorithms consider workflows with the following characteristics. Firstly, they are modeled as DAGs with no cycles or conditional dependencies. Secondly, their execution requires a set of input data, generates intermediate temporary data, and produces a result of an output data set. Thirdly, tasks are assumed to be nonparallel in the number of VMs they require for their execution. Finally, the structure of the workflows is assumed to be static, that is, tasks or dependencies cannot be updated, added, or removed at runtime.

This section is limited to providing an explanation of each taxonomy classification; examples and references to algorithms for each class are presented in Section 5.

4.1 | Application model taxonomy

All algorithms included in this survey share most of the application model features. They differ however in their ability to schedule either a single or multiple workflows.

4.1.1 | Workflow multiplicity

As shown in Figure 4, algorithms can be designed to schedule a single instance of a workflow, multiple instances of the same workflow, or multiple workflows. On the basis of this, we identify 3 types of scheduling processes from the workflow multiplicity perspective.

Single workflow. Algorithms in this class are designed to optimize the schedule of a single workflow. This is the traditional model used in grids and clusters and is still the most common one in cloud computing. It assumes the scheduler manages the execution of workflows sequentially and independently. In this way, the scheduling algorithm can focus on optimizing cost and meeting the QoS requirements for a single user and a single DAG.

Workflow ensembles. Many scientific applications[39–41] are composed of more than one workflow instance. These interrelated workflows are known as ensembles and are grouped together because their combined execution produces a desired output.[42] In general, the workflows in an ensemble have a similar structure but differ in size and input data. Scheduling algorithms in this category focus on executing every workflow on the ensemble using the available resources. Policies need to be aware of

FIGURE 4 Application model taxonomy

the QoS requirements are meant for multiple workflows and not just a single one. For example, all 100 workflows in an ensemble with a 1-hour deadline need to be completed before this time limit. On the basis of this, algorithms are generally concerned with the amount of work (number of executed workflows) completed and tend to include this in the scheduling objectives. Another characteristic of ensembles is that the number instances is generally known in advance and hence the scheduling strategy can use this when planning the execution of tasks.

Multiple workflows. This category is similar to the workflow ensembles one, but differs from it in the workflows being scheduled are not necessarily related to each other and might vary in structure, size, input data, application, etc. More importantly, the number and type of workflows are not known in advance, and therefore, the scheduling is viewed as a dynamic process in which the workload is constantly changing, and workflows with varying configurations are continuously arriving for execution. Yet another difference is that each workflow instance has its own independent QoS requirements. Algorithms in this category need to deal with the dynamic nature of the problem and need to efficiently use the resources to meet the QoS requirements of as many workflows as possible.

An example of a system addressing these issues is proposed by Tolosana-Calasanz et al.[43] They develop a workflow system for the enforcement of QoS of multiple scientific workflow instances over a shared infrastructure such as a cloud computing environment. However, their proposal uses a superscalar model of computation for the specification of the workflows as opposed to the DAG model considered in this survey.

4.2 | Scheduling model taxonomy

There have been extensive studies on the classification of scheduling algorithms on parallel systems. For instance, Casavant and Kuhl[44] proposed a taxonomy of scheduling algorithms in general-purpose distributed computing systems, Kwok and Ahmad[45] developed a taxonomy for static scheduling algorithms for allocating directed task graphs to multiprocessors, while Yu et al[46] studied the workflow scheduling problem in grid environments. Because these scheduling models still apply to the surveyed algorithms, in this section we identify, and in some cases extend, those characteristics that are most relevant to our problem. Aside from a brief introduction to their general definition, we aim to keep the discussion of each category as relevant to the scheduling problem addressed in this work as possible. Figure 5 illustrates the features selected to study the scheduling model.

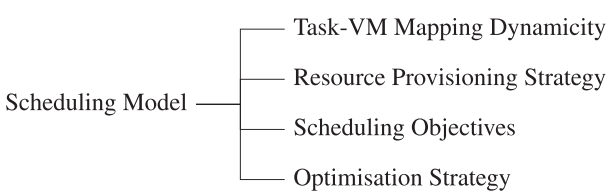

FIGURE 5 Scheduling model taxonomy

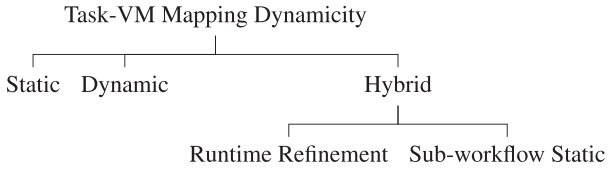

FIGURE 6 Types of task-VM mapping dynamicity

4.2.1 | Task-VM mapping dynamicity

Following the taxonomy of scheduling for general-purpose distributed systems presented by Casavant and Kuhl,[44] workflow scheduling algorithms can be classified as either static or dynamic. This classification is common knowledge to researchers studying any form of scheduling, and hence, it provides readers with a quick understanding of key high-level characteristics of the surveyed algorithms. Furthermore, it is highly relevant for cloud environments as it determines the degree of adaptability that the algorithms have to an inherently dynamic environment. In addition to these 2 classes, we identify a third hybrid one, in which algorithms combine both approaches to find a trade-off between the advantages offered by each of them. This classification is depicted in Figure 6.

Static. These are algorithms in which the task to VM mapping is produced in advance and executed once. Such plan is not altered during runtime, and the workflow engine must adhere to it no matter what the status of the resources and the tasks is. This rigidity does not allow them to adapt to changes in the underlying platform and makes them extremely sensitive to execution delays and inaccurate task runtime estimation; a slight miscalculation might lead to the actual execution failing to meet the user's QoS requirements. This is especially true for workflows due to the domino effect the delay in the runtime of one task will have in the runtime of its descendants. Some static algorithms have strategies in place to improve their adaptability to the uncertainties of cloud environments. These include more sophisticated or conservative runtime prediction strategies, probabilistic QoS guarantees, and resource performance variability models. The main advantage of static schedulers is their ability to generate high-quality schedules by using global, workflow-level, optimization techniques and to compare different solutions before choosing the best suited one.

Dynamic. These algorithms make task to VM assignment decisions at runtime. These decisions are based on the current state

of the system and the workflow execution. For our scheduling scenario, we define dynamic algorithms to be those that make scheduling decisions for a single workflow task, at runtime, once it is ready for execution. These allow them to adapt to changes in the environment so that the scheduling objectives can still be met even with high failure rates, unaccounted delays, and poor estimates. This adaptability is their main advantage when it comes to cloud environments; however, it also has negative implications in terms of the quality of the solutions they produce. Their limited task-level view of the problem hurts their ability to find high-quality schedules from the optimization point of view.

Hybrid. Some algorithms aim to find a trade-off between the adaptability of dynamic algorithms and the performance of static ones. We identify 2 main approaches in this category, namely, runtime refinement and subworkflow static. In *runtime refinement*, algorithms first device a static assignment of tasks before runtime. This assignment is not rigid as it may change during the execution of the workflow on the basis of the current status of the system. For example, tasks may be assigned to faster VMs or they may be mapped onto a different resource to increase the utilization of resources. Algorithms may choose to update the mapping of a single task or to update the entire schedule every cycle. When updating a single task, decisions are made fast, but their impact on the rest of the workflow execution is unknown. When recomputing the schedule for all of the remaining tasks, the initial static heuristic is used every scheduling cycle, resulting in high time and computational overheads.

The *subworkflow static* approach consists on making static decisions for a group of tasks dynamically. That is, every scheduling cycle, a subset of tasks is statically scheduled to resources on the basis of the current system conditions. This allows the algorithm to make better optimization decisions while enhancing its adaptability. The main disadvantage is that statically assigned tasks, although to a lesser extent, are still subject to the effects of unexpected delays. To mitigate this, algorithms may have to implement further rescheduling or refinement strategies for this subset of tasks.

4.2.2 | Resource provisioning strategy

As with the task to VM mapping, algorithms may also adopt a static or dynamic resource provisioning approach. We define *static* resource provisioners to be those that make all of the decisions regarding the VM pool configuration before the execution of the workflow. *Dynamic* provisioners on the other hand make all of the decisions or refine initial ones at runtime, selecting which VMs to keep active, which ones to lease, and which ones to release as the workflow execution progresses. Figure 7 illustrates different types of static and dynamic resource provisioning strategies.

Static VM pool. This strategy may be used by algorithms adopting a static resource provisioning approach. Once the VM pool is determined, the resources are leased, and they remain active throughout the execution of the workflow. When the application finishes running, the resources are released back to the provider. These algorithms are concerned with estimating the resource capacity needed to achieve the scheduling objectives. The advantage is that once the resource provisioning decision is made, the algorithm can focus solely on the task to VM allocation. The effects of VM provisioning and deprovisioning delays is highly amortized and becomes much easier to manage. However, this model does no take advantage of the elasticity of resources and ignores the cloud billing model. This may result in schedules that fail to meet the QoS requirements because of poor estimates and that are not cost-efficient as even billing periods in which VMs are idle are being charged for.

Elastic VM pool. This strategy is suitable for algorithms following either a static or dynamic resource provisioning approach. This method allows algorithms to update the number and type of VMs being used to schedule tasks as the execution of the workflow progresses. Some algorithms make elastic decisions on the basis of their cost-awareness and the *constraint requirements* of tasks. For instance, a new VM can be provisioned so that the task being scheduled can finish before its deadline while idle VMs can be shutdown to save cost. Another way of achieving this is by periodically estimating the resource capacity needed by tasks to meet the application's constraints and adjust the VM pool accordingly. Other algorithms make scaling decisions on the basis of *performance metrics* such as the overall VM utilization and throughput of tasks. For example, new VMs may be provisioned if the budget allows for it and the utilization rises above a specified threshold or if the number of tasks processed by second decreases below a specified limit. Finally, static algorithms that use elastic VM pools do so by determining the leasing periods of VMs when generating the static schedule. These leasing periods are bounded by the estimated start time of the first task assigned to a VM and the estimated finish time of the last task assigned to it.

4.2.3 | Scheduling objectives

Being cost-aware is the common denominator of all the surveyed algorithms. In addition to this objective, most algorithms also consider

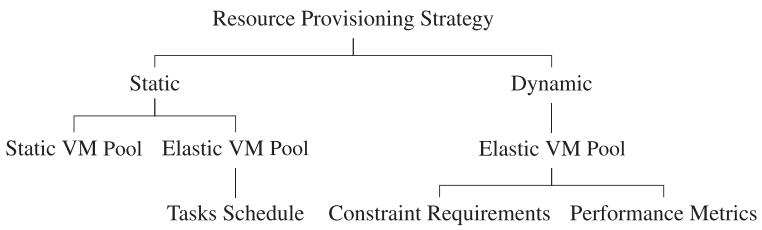

FIGURE 7 Types of resource provisioning strategies

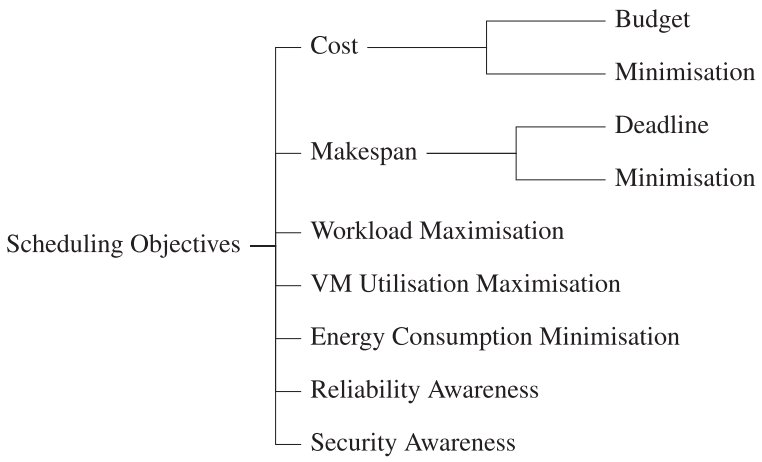

FIGURE 8 Types of scheduling objectives

some sort of performance metric such as the total execution time or the number of workflows executed by the system. Furthermore, some state-of-the-art algorithms also incorporate energy consumption, reliability, and security as part of their objectives. The scheduling objectives included in this taxonomy (Figure 8) are derived from those being addressed by the reviewed literature presented in Section 5.

Cost. Algorithms designed for cloud platforms need to consider the cost of leasing the infrastructure. If they fail to do so, the cost of renting VMs, transferring data, and using the cloud storage can be considerably high. This objective is included in algorithms by either trying to minimize its value or by having a cap on the amount of money spent on resources (i.e., budget). All of the algorithms studied balance cost with other objectives related to performance or nonfunctional requirements such as security, reliability, and energy consumption. For instance, the most commonly addressed QoS requirement is minimizing the total cost while meeting a user-defined deadline constraint.

Makespan. Most of the surveyed algorithms are concerned with the time it takes to run the workflow, or makespan. As with cost, it is included as part of the scheduling objectives by either trying to minimize its value, or by defining a time limit, or deadline, for the execution of the workflow.

Workload maximization. Algorithms developed to schedule ensembles generally aim to maximize the amount of work done, that is, the number of workflows executed. This objective is always paired with constraints such as budget or deadline, and hence, strategies in this category aim at executing as many workflows as possible with the given money or within the specified time frame.

VM utilization maximization. Most algorithms are indirectly addressing this objective by being cost-aware. Idle time slots in leased VMs are deemed as a waste of money as they were paid for but not used, and as a result, algorithms try to avoid them in their schedules. However, it is not uncommon for this unused time slots to arise from a workflow execution, mainly because of the dependencies between tasks and performance requirements. Some algorithms are directly concerned with minimizing these idle time slots and maximizing the utilization of resources, which has benefits for users in terms of cost, and for providers in terms of energy consumption, profit, and more efficient usage of resources.

Energy consumption minimization. Individuals, organizations, and governments worldwide have developed an increased concern to reduce carbon footprints to lessen the impact on the environment. Although not unique to cloud computing, this concern has also attracted attention in this field. A few algorithms that are aware of the energy consumed by the workflow execution have been recently developed. They consider a combination of contradicting scheduling goals as they try to find a trade-off between energy consumption, performance, and cost. Furthermore, virtualization and the lack of control and knowledge of the physical infrastructure limit their capabilities and introduce further complexity into the problem.

Reliability awareness. Algorithms considering reliability as part of their objectives have mechanisms in place to ensure the workflow execution is completed within the users' QoS constraints even if resource or task failures occur. Algorithms targeting unreliable VM instances that are failure prone (eg, Amazon EC2 spot instances) need to have policies addressing reliability in place. Some common approaches include replicating critical tasks and relying on checkpointing to reschedule failed tasks. However, algorithms need to be mindful of the additional costs associated with task replication as well as with the storage of data for checkpointing purposes. Furthermore, it is important to consider that most scientific workflows are legacy applications that are not enabled with checkpointing mechanisms, and hence, relying on this assumption might be unrealistic.

Security awareness. Some scientific applications may require that the input or output data are handled in a secure manner. Even more, some tasks may be composed of sensitive computations that need to be kept secure. Algorithms concerned with these security issues may leverage different security services offered by IaaS providers. They may handle data securely by deeming it *immovable*[47] or may manage sensitive tasks and data in such a way that either resources or providers with a higher security ranking are used to execute and store them. Considering these security measures has an impact when making scheduling decisions as tasks may have to be moved close to immovable data

sets, and the overhead of using additional security services may need to be included in the time and cost estimates.

4.2.4 | Optimization strategy

The optimization strategy taxonomy is shown in Figure 9. Scheduling algorithms can be classified as optimal or suboptimal following the definition of Casavant and Kuhl.[44] Because of the NP-completeness[19] of the discussed problem, finding *optimal* solutions is computationally expensive even for small-scale versions of the problem, rendering this strategy impractical in most situations. In addition, the optimality of the solution is restricted by the assumptions made by the scheduler regarding the state of the system as well as the resource requirements and computational characteristics of tasks. On the basis of this, the overhead of finding the optimal solution for large-scale workflows that will be executed under performance variability may be unjustifiable. For small workflows, however, with coarse-grained tasks that are computationally intensive and are expected to run for long periods, this strategy may be more attractive.

There are multiple methods that can be used to find optimal schedules.[44] In particular, Casavant and Kuhl[44] identify 4 strategies for the general multiprocessor scheduling problem: solution space enumeration and search, graph theoretic, mathematical programming, and queueing theoretic. Most relevant to our problem are solution space enumeration and mathematical models; in the surveyed algorithms mixed integer linear programs (MILPs) have been used to obtain workflow-level optimizations.[48] The same strategy and dynamic programming have been used to find optimal schedules for a subset of the workflow tasks or simplified versions of the problem,[49,50] although this subglobal optimization does not lead to an optimal solution.

Most algorithms focus on generating approximate or near-optimal solutions. For the *suboptimal* category, we identify 3 different methods used by the studied algorithms. The first two are heuristic and meta-heuristic approaches as defined by Yu et al.[46] We add to this model a third hybrid category to include algorithms combining different strategies.

Heuristics. In general, a heuristic is a set of rules that aim to find a solution for a particular problem.[51] Such rules are specific to the problem and are designed so that an approximate solution is found in an acceptable time frame. For the scheduling scenario discussed here, a heuristic approach uses the knowledge about the characteristics of the cloud as well as the workflow application to find a schedule that meets the user's QoS requirements. The main advantage of heuristic-based scheduling algorithms is their efficiency performance; they tend to find satisfactory solutions in an adequate lapse of time. They are also easier to implement and more predictable than meta-heuristic based methods.

Meta-heuristics. While heuristics are designed to work best on a specific problem, meta-heuristics are general-purpose algorithms designed to solve optimization problems.[51] They are higher level strategies that apply problem specific heuristics to find a near-optimal solution to a problem. When compared to heuristic-based algorithms, meta-heuristic approaches are generally more computationally intensive and take longer to run; however, they also tend to find more desirable schedules as they explore different solutions using a guided search. Using meta-heuristics to solve the workflow scheduling problem in clouds involves challenges such as modeling a theoretically unbound number of resources, defining operations to avoid exploring invalid solutions (eg, data dependency violations) to facilitate convergence, and pruning the search space by using heuristics on the basis of the cloud resource model.

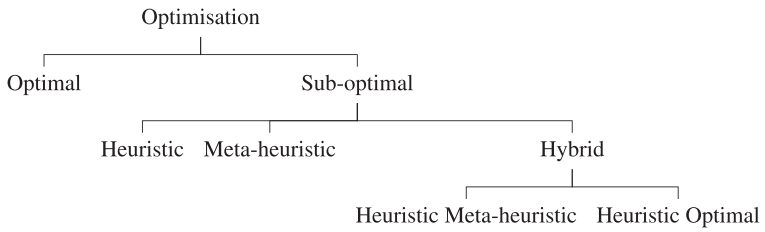

FIGURE 9 Types of optimization strategies

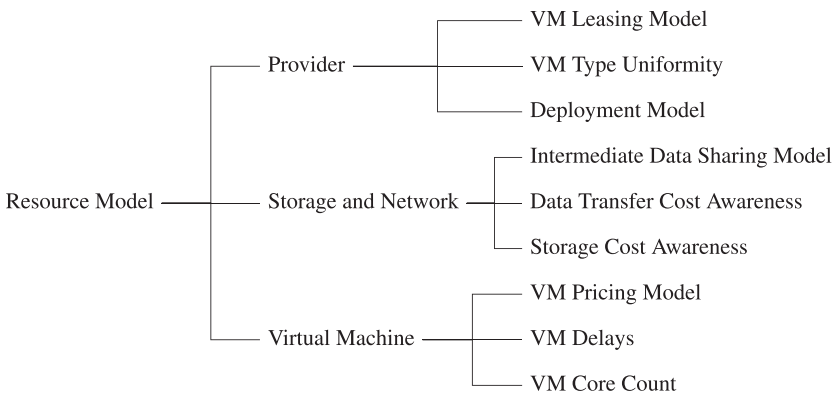

FIGURE 10 Resource model taxonomy

Hybrid. Algorithms using a hybrid approach may use meta-heuristic methods to optimize the schedule of a group of workflow tasks. Another option is to find optimal solutions for simplified and/or smaller versions of the problem and combine them using heuristics. In this way, algorithms may be able to make better optimization decisions than heuristic-based methods while reducing the computational time by considering a smaller problem space.

4.3 | Resource model taxonomy

In this section a taxonomy is presented on the basis of the resource model considerations and assumptions made by algorithms. These design decisions range from high-level ones such as the number of IaaS providers modeled to lower level ones concerned with the services offered by providers, such as the VM pricing model and the cost of data transfers. The characteristics of the resource model considered in this survey are illustrated in Figure 10.

4.3.1 | VM leasing model

This feature is concerned with algorithms assuming providers offer either a bounded or an unbounded number of VMs available to lease for a given user (Figure 11).

Limited. These algorithms assume providers have a cap on the number of VMs a user is allowed to lease. In this way, the resource provisioning problem is somehow simplified and is similar to scheduling with a limited number of processors. However, provisioning decisions are still important because of the overhead and cost associated with leasing VMs.

Unlimited. Algorithms assume they have access to a virtually unlimited number of VMs. There is no restriction on the number of VMs the provisioner can lease, and hence, the algorithm needs to find efficient policies to manage this abundance of resources efficiently.

4.3.2 | VM type uniformity

Algorithms may assume resource homogeneity by leasing VMs of a single type or may use heterogeneous VMs with different configurations on the basis of their scheduling objectives (Figure 12).

Single VM type. In this category, VM instances leased from the IaaS provider are limited to a single type. This assumption is in most cases made to simplify the scheduling process, and the decision of which VM type to use is made without consideration of the workflow and tasks characteristics. This may potentially have a negative impact on the outcome of the algorithm, and as a result, this strategy fails to take full advantage of the heterogeneous nature of cloud resources.

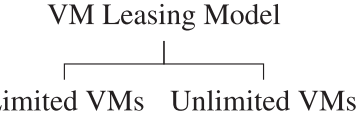

FIGURE 11 Types of virtual machine (VM) leasing models

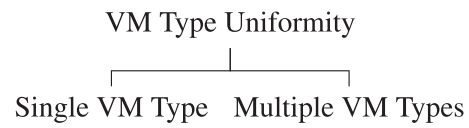

FIGURE 12 Types of virtual machine (VM) uniformity

Multiple VM types. These algorithms acknowledge IaaS clouds offer different types of VMs. They have policies to select the most appropriate types depending on the nature of the workflow, the characteristics of tasks, and the scheduling objectives. This enables the algorithms to use different VM configurations and efficiently schedule applications with different requirements and characteristics.

4.3.3 | Deployment model

Another way of classifying algorithms is based on the number of data centers and public cloud providers they lease resources from as shown in Figure 13.

Single provider. Algorithms in this category consider a single public cloud provider offering infrastructure on a pay-per-use basis. In general, they do not need to consider the cost of transferring data in or out of the cloud as this cost for input and output data sets is considered constant on the basis of the workflow being scheduled.

Multiple providers. This deployment model allows algorithms to schedule tasks onto resources owned by different cloud providers. Each provider has its own product offerings, service level agreements (SLAs), and pricing policies, and it is up to the scheduler to select the best suited one. Algorithms should consider the cost and time of transferring data between providers as these are not negligible. This model may be beneficial for workflows with special security requirements or large data sets distributed geographically. A potential benefit of these intercloud environments is taking advantage of the different billing period granularities offered by different providers. Smaller tasks may be mapped to VMs with finer billing periods such as 1 minute while larger ones to those with coarser-grained periods. While interclouds could be beneficial for the scheduling problem by providing a wider range of services with different prices and characteristics, the lack of standardization, network delays, and data transfer costs, pose real challenges in this area.

Single data . Often, algorithms choose to provision VMs in a single data center, or in the terminology of Amazon EC2, a single availability zone. This deployment model is sufficient for many application scenarios as it is unlikely that the number of VMs required for the execution of the workflow will exceed the data center's capacity. It also offers 2 key advantages of data transfers. The first one is reduced latency and faster transfer times, and the second one is potential cost savings as many providers do not charge for transfers made within a data center.

Multiple data centers. Using a resource pool composed of VMs deployed in different data centers belonging to the same provider is another option for algorithms. This choice is more

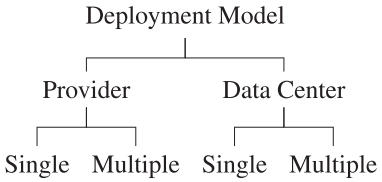

FIGURE 13 Types of provider and data center deployment models

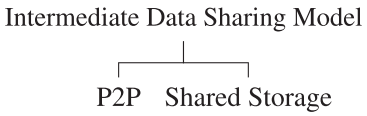

FIGURE 14 Types of intermediate data sharing models

suited for applications with geographically distributed input data. In this way, VMs can be deployed in different data centers on the basis of the location of the data to reduce data transfer times. Other workflows that benefit from this model are those with sensitive data sets that have specific location requirements due to security or governmental regulations. Finally, algorithms under this model need to be aware of the cost of transferring data between different data centers as most providers charge for this service.

4.3.4 | Intermediate data sharing model

Workflows process data in the form of files. The way in which these files are shared affects the performance of scheduling algorithms as they have an effect on metrics such as cost and makespan. A common approach is to assume a peer-to-peer (P2P) model while another technique is to use a global shared storage system as a file repository (Figure 14).

P2P. These algorithms assume files are transferred directly from the VM running the parent task to the VM running the child task. This means tasks communicate in a synchronous manner, and hence, VMs must be kept running until all of the child tasks have received the corresponding data. This may result in higher costs as the lease time of VMs is extended. Additionally, the failure of a VM would result in data loss that can potentially require the re-execution of several tasks to recover. The main advantage of this approach is its scalability and lack of bottlenecks.

Shared Storage. In this case, tasks store their output in a global shared storage system and retrieve their inputs from the same. In practice, this global storage can be implemented in different ways such as a network file system, persistent storage solutions like Amazon S3,[52] or more recent offerings such as Amazon EFS.[53] This model has several advantages. Firstly, the data are persisted and hence can be used for recovery in case of failures. Secondly, it allows for asynchronous computation as the VM running the parent task can be released as soon as the data are persisted in the storage system. This may not only increase the resource utilization but also decrease the cost of VM leasing. Finally, if the application scenario allows for it, redundant computations can be avoided if the persisted data can be reused by multiple tasks, instances of the same workflow or even by other workflows of the same type. There are 3 main disadvantages of this approach. The first one is the potential for the storage system to be a bottleneck, the second one is the extra cost of using the cloud's storage system, and the third one is a potential higher number of data transfers when compared to the P2P model.

4.3.5 | Data transfer cost awareness

The IaaS providers have different pricing schemes for different types of data transfers, depending if the data are being transferred into, within, or out of their facilities. Transferring inbound data are generally free, and hence, this cost is ignored by all of the studied algorithms. On the contrary, transferring data out of the cloud provider is generally expensive. Algorithms that schedule workflows across multiple providers are the only ones that need to be concerned with this cost, as data may need to be transferred between resources belonging to different providers. As for transferring data within the facilities of a single provider, it is common for transfers to be free if they are done within the same data center and to be charged if they are between different data centers. Hence, those algorithms considering multiple data centers in their resource model should include this cost in their estimations. Finally, regarding access to storage services, most providers such as Amazon S3,[52] Google Cloud Storage,[54] and Rackspace Block Storage[55] do not charge for data transfers in and out of the storage system, and hence, this value can be ignored by algorithms making use of these facilities.

4.3.6 | Storage cost awareness

Data storage is charged on the basis of the amount of data being stored. Some providers have additional fees on the basis of the number and type of operations performed on the storage system (i.e., GET, PUT, and DELETE). This cost is only relevant if cloud storage services are used, and even in such cases, it is ignored in many models mainly because the amount of data used and produced by a workflow is constant and independent of the scheduling algorithm. However, some algorithms do acknowledge this cost and generally estimate it on the basis of the data size and a fixed price per data unit.

4.3.7 | VM pricing model

As depicted in Figure 15, we identify 4 different pricing models considered by the surveyed algorithms that are relevant to our discussion: dynamic, static, subscription-based, and time unit.

Dynamic pricing. The price of instances following this model varies over time and is determined by the market dynamics of supply and demand. Generally, users acquire dynamically priced VMs by means of auctions or negotiations. In these auctions, users request a VM by revealing the maximum amount of money they are willing to pay for it, providers then decide to accept or reject the request on the basis of the current market conditions. These type of instances generally offer users an economical advantage over statically priced ones. An example of VMs following this pricing model are Amazon EC2 Spot Instances.[38] The spot market allows users to bid on VMs and run them whenever their bidding prices exceed the current market (i.e., spot) price. Through this model, users can lease instances at considerably lower prices but are subject to the termination of VMs when the

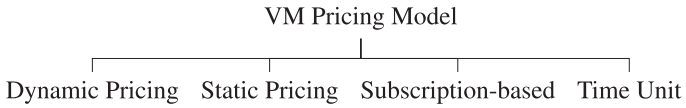

FIGURE 15 Types of virtual machine (VM) pricing models

market price becomes higher than the bidding one. Hence, tasks running on spot instances need to be either interruption-tolerant or scheduling algorithms need to implement a recovery or fault tolerant mechanism; VMs with dynamic pricing are often used opportunistically by scheduling algorithms[56,57] in conjunction with statically priced ones to reduce the overall cost of executing the workflow.

Static pricing. The static pricing model is the conventional cloud pricing model and is offered by most providers; VMs are priced per billing period, and any partial utilization is charged as a full-period utilization. An example of a provider offering instances under this pricing model is Google Compute Engine.[58] All VM types are charged a minimum of 10 minutes, and after this, they are charged in 1-minute intervals, rounded up to the nearest minute.[59] For example, if a VM is used for 3 minutes, it will be billed for 10 minutes of usage, and if it is used for 12.4 minutes, it will be billed for 13.

Subscription-based. Under this model, instances are reserved for a longer time frame, usually monthly or yearly. Generally, payment is made upfront and is significantly lower when compared to static pricing; VMs are billed at the discounted rate for every billing period (eg, hour) in the reserved term regardless of usage. For the cloud workflow scheduling problem, this pricing model means that schedulers need to use a fixed set of VMs with fixed configurations to execute the tasks. This transforms the problem, at least from the resource provisioning point of view, into one being designed for a platform with limited availability of resources such as a grid or cluster. An example of a provider offering subscription-based instances is Cloudsigma.[60] It offers unbundled resources such as CPU, RAM, and storage (users can specify exactly how much of each resource they need without having to select from a predefined bundle) that can be leased for either 1, 3, or 6 months or 1, 2, or 3 years. They offer a fixed discount on the basis of the leasing period selected; the longer the leasing period, the larger the discount.

Time unit. Algorithms in this category assume VMs are charged per time unit. Under this model, there is no resource wastage or additional costs because of unused time units in billing periods. Hence, the scheduling is simplified as there is no need to use idle time slots of leased VMs as the cost of using the resources corresponds to the exact amount time they are used for. This may be considered as an unrealistic approach as there are no known cloud providers offering this level of granularity and flexibility yet; however, some algorithms do assume this model for simplicity. Additionally, there is the possibility of new pricing models being offered by providers or emerging from existing ones, as is pointed out by Arabnejad et al,[61] a group of users may, for example, rent a set of VMs on a subscription-based basis, share them, and price their use on a time-unit basis.

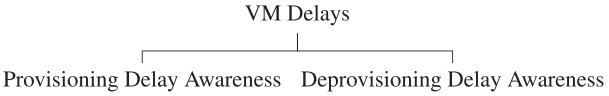

FIGURE 16 Types of virtual machine (VM) delays

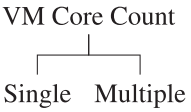

FIGURE 17 Types of virtual machine (VM) core count

4.3.8 | VM delays

This category is concerned with the awareness of algorithms regarding the VM provisioning and deprovisioning delays (Figure 16).

VM provisioning delay. In Section 3.2, VM provisioning delays have non-negligible, highly variable values. To make accurate scheduling decisions, algorithms need to consider this delay when making runtime estimates. Its effect is specially noticeable in situations in which the number of VMs in the resource pool is highly dynamic because of performance requirements, topological features of the workflow, and provisioning strategies designed to save cost. All of the algorithms that acknowledge this delay do so by associating an estimate of its value to each VM type. Another strategy used is to avoid these delays by reusing leased VMs when possible.

VM deprovisioning delay. The impact of VM deprovisioning delays is strictly limited to the execution cost. To illustrate this, consider the case in which a scheduler requests to shutdown a VM just before the end of the first billing period. By the time the instance is actually released, the second billing period has started, and hence, 2 billing periods have to be paid for. Those algorithms that consider this delay do so by allowing some time, an estimate of the deprovisioning value, between the request to shutdown the VM and the end of the billing period.

4.3.9 | VM core count

This category refers to wether algorithms are aware of multicore VMs for the purpose of scheduling multiple, simultaneous tasks on them (Figure 17).

Single. Most algorithms assume VMs have a single core and hence are only capable of processing one task at a time. This simplifies the scheduling process and eliminates further performance degradation and variability due to resource contention derived from the coscheduling of tasks.

Multiple. The IaaS providers offer VMs with multiple cores. Algorithms that decide to take advantage of this feature may schedule multiple tasks to run simultaneously in the same VM,

potentially saving time, cost, and avoiding intermediate data transfers. However, this coscheduling of tasks may result in significant performance degradation because of resource contention. Being mindful of this is essential when making scheduling decisions as estimating task runtimes assuming optimal performance will most definitely incur in additional significant delays. Zhu et al,[62] for example, bundle tasks together on the basis of their resource usage characteristics; tasks assigned to the same VM should have different computational requirements to minimize resource contention.

5 | SURVEY

This section discusses a set of algorithms relevant to each of the categories presented in the taxonomy and depicts a complete classification including all of the surveyed algorithms, these results are summarized in Tables 1, 2, 3, and 4.

5.1 | Scheduling multilevel deadline-constrained scientific workflows

Malawski et al[50] present a mathematical model that optimizes the cost of scheduling workflows under a deadline constraint. It considers a multicloud environment where each provider offers a limited number of heterogeneous VMs, and a global storage service is used to share intermediate data files. Their method proposes a global optimization of task and data placement by formulating the scheduling problem as a mixed integer program (MIP). Two different versions of the algorithm are presented, one for coarse-grained workflows, in which tasks have an execution time in the order of one hour, and another for fine-grained workflows with many short tasks and with deadlines shorter than one hour.

The MIP formulation to the problem takes advantage of some characteristics of large-scale scientific workflows: they are composed of sequential levels of independent tasks. On the basis of this, the authors decided to group tasks in each level on the basis of their computational cost and input/output data and schedule these groups instead of single tasks, reducing the complexity of the MIP problem considerably. Another design choice to keep the MIP model simple is that VMs cannot be shared between levels; however, this may potentially lead to low resource utilization and higher costs for some workflows. Because the MIP model already assumes VMs cannot be shared between levels, a potential improvement could be to design the MIP program so that the schedule for each level can be computed in parallel. Finally, the algorithm is too reliant on accurate runtime, storage, and data transfer time estimations, considering its main objective is to finish executions before a deadline.

5.2 | Security-aware and budget-aware

The security-aware and budget-aware (SABA) algorithm[47] was designed to schedule workflows in a multicloud environment. The authors define the concept of immoveable and movable datasets. Movable data have no security restrictions and hence can be moved between data centers in replicated if required. Immoveable data on the other hand are restricted to a single data center and cannot be migrated or replicated because of security or cost concerns. The algorithm consists of 3 main phases. The first one is the clustering and prioritization stage in which tasks and data are assigned to specific data centers on the basis of the workflow's immoveable data sets. In addition to this, priorities are assigned to tasks on the basis of their computation and I/O costs on a baseline VM type. The second stage statically assigns tasks to VMs on the basis of a performance-cost ratio. Finally, the intermediate data are moved dynamically at runtime with the location of tasks that are ready for execution guiding this process; SABA calculates the cost of a VM on the basis of the start time of the first task assigned to it and the end time of the last task mapped to it. Even though the authors do not specifically describe a resource provisioning strategy, the start and end times of VMs can be derived from the start and end times of tasks, and therefore, we classify it as adopting an elastic resource pool strategy.

In addition to the security of data, SABA also considers tasks that may require security services such as authentication, integrity, and confidentiality and includes the overheads of using these services in their time and cost estimations. What is more, instead of considering just the CPU capacity of VMs to estimate runtimes, SABA also considers features such I/O, bandwidth, and memory capacity. The cost of VMs is calculated on the basis of the total units of time the machine was used for, and billing periods imposed by providers are not considered. This may result in higher VM costs than expected when using the algorithm on a real cloud environment. Other costs considered include data transfer costs between data centers as well as the storage used for input and output workflow data.

5.3 | Particle swarm optimization–based resource provisioning and scheduling algorithm

Rodriguez and Buyya[63] developed a static, cost minimization, deadline-constrained algorithm that considers features such as the elastic provisioning and heterogeneity of unlimited compute resources as well as VM performance variation. Both resource provisioning and scheduling are merged and modeled as a particle swarm optimization problem. The output of the algorithm is hence a near-optimal schedule determining the number and types of VMs to use, as well as their leasing periods and the task to resource mapping.

The global optimization technique is an advantage of the algorithm as it allows it to generate high-quality schedules. Also, to deal with the inability of the static schedule to adapt to environmental changes, the authors introduce an estimate of the degradation in performance that would be experienced by VMs when calculating runtimes. In this way, a degree of tolerance to the unpredictability of the environment is introduced. The unlimited resource model is successfully captured by the algorithm; however, the computational overhead increases rapidly with the number of tasks in the workflow and the types of VMs offered by the provider.

5.4 | Multi-objective heterogeneous earliest finish time

Durillo and Prodan developed the multi-objective heterogeneous earliest finish time (MOHEFT) algorithm[64] as an extension of the well-known DAG scheduling algorithm HEFT.[65] The heuristic-based method computes a set of pareto-based solutions from which users can select the best-suited one. The MOHEFT builds several intermediate

workflow schedules, or solutions, in parallel in each step, instead of a single one as is done by HEFT. The quality of the solutions is ensured by using dominance relationships while their diversity is ensured by making use of a metric known as crowding distance. The algorithm is generic in the number and type of objectives it is capable of handling; however, makespan and cost were optimized when running workflow applications in an Amazon-based commercial cloud.

The flexibility offered by MOHEFT as a generic multi-objective algorithm is very appealing. In addition, the pareto front is an efficient tool for decision support as it allows users to select the most appropriate trade-off solution on the basis of their needs. For example, their experiments demonstrated that in some cases, cost could be reduced by half with a small increment of 5% in the schedule makespan. Finally, as noted by the authors, most of the solutions computing the pareto front are based on genetic algorithms. These approaches require high computation time while MOHEFT offers an approximate time complexity of $O(n \times m)$ where n is the number of tasks and m the number of resources.

5.5 | Fault-tolerant scheduling using spot instances

Poola et al[57] propose an algorithm that schedules tasks on 2 types of cloud instances, namely, on-demand and spot. Specifically, it considers a single type of spot VM type (the cheapest one) and multiple types of on-demand VMs. The authors define the concept of latest time to on-demand, or LTO. It determines when the algorithm should switch from using spot to on-demand instances to ensure the user-defined deadline is met. A bidding strategy for spot VMs is also proposed; the bidding starts close to the initial spot price and increases as the execution progresses so that it gets closer to the on-demand price as the LTO approaches. This lowers the risk of out-of-bid events closer to the LTO and increases the probability of meeting the deadline constraint.

This algorithm is one of the few exploring the benefits of using dynamically priced VMs. It addresses a challenging problem by aiming to meet deadlines not only under variable performance but also under unreliable VMs that can be terminated at any time. The benefits are clear with the authors finding that by using spot instances, the algorithm is able to considerably lower the execution cost. However, this advantage may be reduced because only the cheapest spot VM is considered. If deadlines are strict, the cheapest VM may not be able to process many tasks before the LTO, and hence, most of the workflow execution would happen in on-demand instances. Another potential drawback of the algorithm is its reliance on checkpointing. Not only are many scientific workflows legacy applications lacking checkpointing capabilities but also storing data for this purpose may considerably increase the infrastructure cost.

5.6 | IaaS cloud partial critical path

The IaaS cloud partial critical path (IC-PCP) algorithm[66] has objective to minimize the execution cost while meeting a deadline constraint. The algorithm begins by finding a set of tasks, namely, partial critical paths (PCPs), associated to each exit node of the workflow (an exit node is defined as a node with no children tasks). The tasks on each path are then scheduled on the same VM and are preferably assigned to an already leased instance, which can meet the latest finish time requirements of the tasks. If this cannot be achieved, the tasks are assigned to a newly leased VM of the cheapest type that can finish them on time. The PCPs are recursively identified, and the process is repeated until all of the workflow tasks have been scheduled.

Along with IC-PCP and with the same scheduling objectives, the authors propose the IC-PCPD2 (IC-PCP with deadline distribution algorithm). The main difference between both algorithms is that, instead of assigning all tasks in a path to the same VM, IC-PCPD2 places each individual task on the cheapest VM that can finish it on time. According to the authors, IC-PCP outperforms IC-PCPD2 in most of the cases. This highlights one of the main advantages of IC-PCP and an important consideration regarding workflow executions in clouds: data transfer times can have a high impact on the makespan and cost of a workflow execution. The IC-PCP successfully addresses this concern by scheduling parent and child tasks on the same VM, thereby reducing the amount of VM to VM communication.

A disadvantage of IC-PCP is that it does not account for VM provisioning delays or for resource performance variation. This makes it highly sensitive to CPU performance degradation and causes deadlines to be missed because of unexpected delays. Because it is a static and heuristic-based algorithm, it is capable of finding high-quality schedules efficiently, making it suitable to schedule large-scale workflows with thousands of tasks. Hence, IC-PCP could be better suited to schedule large workflows with tasks that have low CPU requirements so that the impact of resource performance degradation is reduced.

5.7 | Enhanced IC-PCP with replication

Calheiros and Buyya[67] propose the enhanced IC-PCP with replication (EIPR) algorithm, a scheduling and provisioning solution that uses the idle time of provisioned VMs and a budget surplus to replicate tasks to mitigate the effect of performance variation and meet the application's deadline. The first step of the algorithm consists in determining the number and type of VMs to use as well as the order and placement of the tasks on these resources. This is achieved by adopting the main heuristic of IC-PCP,[66] that is, identifying PCPs and assigning their tasks to the same VM. The second step is to determine the start and stop time of VMs. The EIPR does this by considering both the start and end time of tasks as well as input and output data transfer times. Finally, the algorithm replicates tasks in idle time slots of provisioned VMs or on new VMs if the replication budget allows for it. The algorithm prioritizes the replication of tasks with a large ratio of execution to available time, then tasks with long execution times, and finally tasks with a large number of children.

Although a static algorithm, EIPR is successful in mitigating the effects of poor and variable performance of resources by exploiting the elasticity and billing scheme of clouds. This allows it to generate high-quality schedules while being robust to unexpected environmental delays. However, the replication of tasks may not be as successful in cases in which the execution time of tasks is close to the size of the billing period. This is mainly because there are less chances or reusing idle time slots. Another advantage of the algorithm is its accountability of VM provisioning delays and its data transfer aware provisioning adjust, which enables VMs to be provisioned before the actual start time of their first task to allow for input data to be transferred beforehand.

TABLE 1 Workflows used in the evaluation of the surveyed algorithms

Algorithm	Evaluation Strategy	Montage	Cybershake	Epigenomics	SIPHT	LIGO	Randomly Generated	Other
Malawski et al[50]	Real cloud	✓	✓	✓	✓	✓		
SABA [47]	Simulation				✓	✓	✓	Gene2Life, Motif, NCFS, PSMerge
WRPS [49]	Simulation	✓		✓	✓	✓		
RNPSO [72]	Simulation						✓	
Rodriguez&Buyya [63]	Simulation	✓	✓		✓	✓		
MOHEFT [64]	Simulation						✓	WIEN2K, POVRay
Poola et. al[57]	Simulation					✓		
Poola et al (Robust)[73]	Simulation	✓	✓	✓	✓	✓		
IC-PCP/IC-PCPD2 [66]	Simulation	✓	✓	✓	✓	✓		
EIPR [67]	Simulation	✓	✓		✓	✓		
Strech&Compact [74]	Simulation	✓	✓	✓	✓	✓		
Oliveira et al[75]	Real cloud							SciPhy
Genez et al[48]	Simulation	✓					✓	
PBTS [68]	Simulation	✓	✓	✓	✓	✓	✓	
BTS [69]	Simulation	✓	✓	✓	✓	✓	✓	
Zhu et al[76]	Simulation	✓	✓	✓	✓	✓	✓	
SPSS [42]	Simulation	✓	✓	✓	✓	✓		
DPDS [42]	Simulation	✓	✓	✓	✓	✓		
SPSS-ED [70]	Simulation	✓			✓	✓		
SPSS-EB [70]	Simulation	✓			✓	✓		
Wang et al[77]	Simulation						✓	
ToF [78]	Real cloud	✓				✓		
Dyna [56]	Simulation						✓	
SCS [71]	Simulation						✓	

5.8 | Workflow scheduling considering 2 SLA levels

Genez et al[48] implement an SaaS provider offering a workflow execution service to its customers. They consider 2 types of SLA contracts that can be used to lease VMs from IaaS providers: static and subscription based. Specifically, they consider the corresponding options offered by Amazon EC2, namely, on-demand and reserved instances. In their model, the SaaS provider has a pool of reserved instances that are used to execute workflows before a user-defined deadline. However, if the reserved instance infrastructure is not enough to satisfy the deadline, then on-demand instances are acquired and used to meet the workflow's requirements. Even though their algorithm is presented in the context of an SaaS provider potentially serving multiple users, the solution is designed to schedule a single workflow at time. They formulate the scheduling problem as a MILP with the objective of minimizing the total execution cost while meeting the application deadline of the workflow. They then propose 2 heuristics to derive a feasible schedule from the relaxed version of the MILP. Their algorithm is capable of selecting the best-suited IaaS provider as well as the VMs required to guarantee the QoS parameters.

The scalability of the MILP model presented is a concern. The number of variables and constraints in the formulation increases rapidly with the number of providers, maximum number of VMs that can be leased from each provider, and the number of tasks in the DAG. This may rule the algorithm as impractical in many real-life scenarios, especially considering even after a time-expensive schedule computation; the workflow may still finish after its deadline because of poor and variable resource performance. Aware of this limitation, the authors propose a relaxation approach and application of time limits to the MILP solver; however, the scalability concerns still remain in this cases as workflows are likely to have thousands of tasks. On the positive side, the MILP finds an optimal solution to their formulation of the problem and can be successfully used in scenarios where workflows are small or even as a benchmark to compare the quality of schedules generated by different algorithms.

5.9 | Partitioned balanced time scheduling

The partitioned balanced time scheduling (PBTS) algorithm[68] was designed to process a workflow in a set of homogeneous VMs

TABLE 2 Algorithm classification for the application model

Algorithm	Workflow Dynamicity
Malawski et al[50]	Single
SABA[47]	Single
WRPS[49]	Single
RNPSO[72]	Single
Rodriguez&Buyya[63]	Single
MOHEFT[64]	Single
Poola et al[57]	Single
Poola et al (Robust)[73]	Single
IC-PCP/IC-PCPD2[66]	Single
EIPR[67]	Single
Strech&Compact[74]	Single
Oliveira et al[75]	Single
Genez et al[48]	Single
PBTS[68]	Single
BTS[69]	Single
Zhu et al[76]	Single
SPSS[42]	Ensemble
DPDS[42]	Ensemble
SPSS-ED[70]	Ensemble
SPSS-EB[70]	Ensemble
Wang et al[77]	Multiple
ToF[78]	Multiple
Dyna[56]	Multiple
SCS[71]	Multiple

by partitioning its execution so that scheduling decisions are made every billing period. Its main objective is to estimate, for each scheduling cycle or partition, the minimum number of compute resources required to execute the workflow within the user-specified deadline. For each partition, PBTS first identifies the set of tasks to run on the basis of an approximate resource capacity estimate that considers the total cost. Then, it estimates the exact number of resources needed to run the tasks during the partition using the balanced time scheduling (BTS) algorithm,[69] which was previously proposed by the same authors. Finally, the actual VMs are allocated, and the tasks executed on the basis of the schedule obtained from running BTS.

Adjusting the number of VMs and monitoring the execution of tasks every scheduling cycle allows the algorithm to have a higher tolerance to performance variability and take advantages of the elasticity of clouds. The PBTS is a good example of an algorithm using a subworkflow static hybrid approach to address the task to VM mapping, statically scheduling tasks every billing period. It also uses runtime refinement to handle delays on the statically scheduled tasks. The algorithm is capable of handling tasks that require multiple hosts for their execution (eg, MPI tasks), and even though this is out of the scope of this survey, we include it as it still has the ability to schedule workflows were all tasks require a single host. The PBTS was clearly designed for coarse-grained billing periods, such as 1 hour. For finer-grained periods, such as 1 minute, PBTS may not be as successful as tasks are unlikely to finish within a single partition, and it would be difficult to assign a large-enough number of tasks to each partition to make the scheduling overhead worthwhile.

5.10 | Static provisioning static scheduling and dynamic provisioning dynamic scheduling

Malawski et al[42] propose 2 algorithms to schedule workflow ensembles that aim to maximize the number of executed workflow instances while meeting deadline and budget constraints. The dynamic provisioning dynamic scheduling (DPDS) algorithm first calculates the initial number of VMs to use on the basis of the budget and deadline. This VM pool is then updated periodically on the basis of the VM utilization; if the utilization falls below a predefined threshold then VMs are shutdown, and if it exceeds this threshold and the budget allows for it then new VMs are leased. The scheduling phase assigns tasks on the basis of their priority to arbitrarily chosen VMs dynamically until all of the workflow instances are executed or until the deadline is reached. The WA-DPDS (workflow aware DPDPS) is a variant of the algorithm that aims to be more efficient by executing only tasks of workflows that can be finished within the specified QoS constraints. It incorporates an admission control procedure so that only those workflow instances that can be completed within the specified budget are scheduled and executed. The authors demonstrate the ability of DPDS to adapt to unexpected delays, including considerable provisioning delays and innacurate task runtime estimates. A drawback of the algorithm is leasing as many VMs as allowed by the budget from the beginning of the ensemble execution. This may result in VMs being idle for long periods as they wait for tasks to become ready for execution resulting in wasted time slots and additional billing periods.

The static provisioning static scheduling (SPSS) algorithm assigns sub-deadlines to each task on the basis of the slack time of the workflow (the time that the workflow can extend its critical path and still finish by the deadline). The tasks are statically assigned to free time slots of existing VMs so that the cost is minimized and their deadline is met. If there are no time slots that satisfy these constraints then new VMs are leased to schedule the tasks. Being a static approach, the authors found that SPSS is more sensitive to dynamic changes in the environment than DPDS. However, it outperforms its dynamic counterpart in terms of the quality of schedules as it has the opportunity to use its knowledge on the workflow structure and to compare different outputs before choosing the best one. The major drawback of SPPS is its static nature and unawareness of VM provisioning times, as the authors found it to be too sensitive to these delays for it to be of practical use.

5.11 | SPSS-ED and SPSS-EB

Pietri et al[70] propose 2 algorithms to schedule workflow ensembles in clouds, both on the basis of SPSS.[42] One of them, called SPSS-ED, focuses on meeting energy and deadline constraints while another one, called SPSS-EB, focuses on meeting energy and budget constraints. Both algorithms aim to maximize the number of completed workflows. For each workflow in the ensemble, SPSS-EB plans the execution of the workflow by scheduling each task so that the total energy consumed is minimum. It then accepts the plan and executes the workflow only if the energy and budget constraints are met. The same processes is used in SPSS-ED but instead of budget, deadline is considered as a constraint.

This work does not consider data transfer times and considers only a single type of VM for simplicity. It also assumes the data center is composed of homogeneous hosts with fixed capacity in VMs.

TABLE 3 Algorithm classification for the scheduling model

Algorithm	Task-VM Mapping Dynamicity	Resource Provisioning Strategy	Scheduling Objectives	Optimization Strategy
Malawski et al[50]	Static	Static EP	Deadline and cost	Hybrid HO
SABA[47]	Hybrid RR	Static EP	Budget and makespan and security	Heuristic
WRPS[49]	Hybrid SS	Dynamic CR	Deadline and cost	Hybrid HO
RNPSO[72]	Static	Static EP	Deadline and cost	Meta-heuristic
Rodriguez&Buyya[63]	Static	Static EP	Deadline and cost	Meta-heuristic
MOHEFT[64]	Static	Static EP	Generic multi-objective	Heuristic
Poola et al[57]	Dynamic	Dynamic CR	Deadline and cost and reliability	Heuristic
Poola et al (Robust)[73]	Static	Static EP	Makespan and cost	Heuristic
IC-PCP/IC-PCPD2[66]	Static	Static EP	Deadline and cost	Heuristic
EIPR[67]	Static	Static EP	Deadline and cost	Heuristic
Strech&Compact[74]	Static	Static SP	Makespan and res. util. and cost	Heuristic
Oliveira et al[75]	Dynamic	Dynamic PM	Deadline and budget and reliability	Heuristic
Genez et al[48]	Static	Static EP	Deadline and cost	Optimal
PBTS[68]	Hybrid SS	Dynamic CR	Deadline and cost	Heuristic
BTS[69]	Static	Static SP	Deadline and cost	Heuristic
Zhu et al[76]	Static	Static EP	Makespan and cost	Meta-heuristic
SPSS[42]	Static	Static EP	Budget and deadline and workload	Heuristic
DPDS[42]	Dynamic	Dynamic PM	Budget and deadline and workload	Heuristic
SPSS-ED[70]	Static	Static EP	Deadline and workload and energy	Heuristic
SPSS-EB[70]	Static	Static EP	Budget and workload and energy	Heuristic
Wang et al[77]	Dynamic	Dynamic CR	Makespan and cost	Heuristic
ToF[78]	Static	Static EP	Deadline and cost	Heuristic
Dyna[56]	Dynamic	Dynamic CR	Probabilistic deadline and cost	Heuristic
SCS[71]	Dynamic	Dynamic CR	Deadline and cost	Heuristic

In reality, however, data centers are composed of heterogeneous servers with different characteristics. Furthermore, it assumes physical hosts are exclusively used for the execution of the workflows in the ensemble, and this again is an unrealistic expectation. Despite these disadvantages, this is the first work that considers energy consumption when scheduling ensembles and hence can be used as a stepping stone to make further advances in this area.

5.12 | Dyna

Dyna[56] is a scheduling framework that considers the dynamic nature of cloud environments from the performance and pricing point of view. It is based on a resource model similar to Amazon EC2 as it considers both spot and on-demand instances. The goal is to minimize the execution cost of workflows while offering a probabilistic deadline guarantee that reflects the performance variability of resources and the price dynamics of spot instances. Spot instances are used to reduce the infrastructure cost and on-demand instances to meet the deadline constraints when spot instances are not capable of finishing tasks on time. This is achieved by generating a static hybrid instance configuration plan (a combination of spot and on-demand instances) for every task. Each configuration plan indicates a set of spot instances to use along with their bidding price and one on-demand instance type which should be used in case the execution fails on each of the spot instances on the configuration set. At runtime, this configuration plan, in addition to instance consolidation and reuse techniques are used to schedule the tasks.

Contrary to most algorithms, Dyna recognizes that static task runtime estimations and deterministic performance guarantees are not suited to cloud environments. Instead, the authors propose offering users a more realistic, probabilistic deadline guarantee that reflects the cloud dynamics. Their probabilistic models are successful in capturing the variability in I/O and network performance, as well as in spot prices. However, Dyna does not consider CPU performance variations as according to the authors their findings show it is relatively stable. Additionally, by determining the best instance type for each task statically, Dyna is able to generate better quality schedules; however, it still makes scheduling decisions for one task at a time, limiting its global task to VM mapping optimization capabilities.

TABLE 4 Algorithm classification for the resource model

Algorithm	VM Leasing Model	VM Type Uniformity	Provider Dep. Model	Data Center Dep. Model	Data Sharing Model	Data Transfer Cost	Storage Cost	VM Pricing Model	VM Delays	VM Core Count
Malawski et al[50]	Limited	Multiple	Multiple	Multiple	Shared	Yes	No	Static	No	Multiple
SABA[47]	Limited	Multiple	Multiple	Multiple	P2P	Yes	Yes	Time Unit	No	Single
WRPS[49]	Unlimited	Multiple	Single	Single	Shared	No	No	Static	Prov. and deprov.	Single
RNPSO[72]	Unlimited	Multiple	Single	Single	P2P	No	No	Static	Prov.	Single
Rodriguez&Buyya[63]	Unlimited	Multiple	Single	Single	P2P	No	No	Static	Prov.	Single
MOHEFT[64]	Limited	Multiple	Single	Single	P2P	Yes	Yes	Static	No	Single
Poola et al[57]	Unlimited	Multiple	Single	Single	P2P	No	No	Dynamic and static	Prov.	Single
Poola et al (Robust)[73]	Unlimited	Multiple	Single	Single	P2P	No	No	Static	Prov.	Single
IC-PCP/IC-PCPD2[66]	Unlimited	Multiple	Single	Single	P2P	No	No	Static	No	Single
EIPR[67]	Unlimited	Multiple	Single	Single	P2P	No	No	Static	Prov.	Single
Strech&Compact[74]	Unlimited	Single	Single	Single	P2P	No	No	Static	No	Multiple
Oliveira et al.[75]	Unlimited	Multiple	Multiple	Multiple	Shared	Yes	No	Static	No	Single
Genez et al.[48]	Limited	Multiple	Multiple	Single	P2P	No	No	Static and subscription	No	Multiple
PBTS[68]	Unlimited	Single	Single	Single	Shared	No	No	Static	No	Single
BTS[69]	Unlimited	Single	Single	Single	Shared	No	No	Time unit	No	Single
Zhu et al[76]	Unlimited	Multiple	Single	Single	P2P	No	No	Static	No	Single
SPSS[42]	Unlimited	Single	Single	Single	Shared	No	No	Static	No	Single
DPDS[42]	Unlimited	Single	Single	Single	Shared	No	No	Static	Prov. and deprov.	Single
SPSS-ED[70]	Unlimited	Single	Single	Single	Shared	No	No	Static	No	Single
SPSS-EB[70]	Unlimited	Single	Single	Single	Shared	No	No	Static	No	Single
Wang et al.[77]	Unlimited	Multiple	Single	Single	P2P	No	No	Static	No	Single
ToF[78]	Limited	Multiple	Single	Single	P2P	No	No	Static	Prov.	Multiple
Dyna[56]	Unlimited	Multiple	Single	Single	P2P	No	No	Dynamic and static	Prov.	Single
SCS[71]	Unlimited	Multiple	Single	Single	P2P	No	No	Static	Prov.	Single

5.13 | SCS

SCS[71] is a deadline-constrained algorithm that has an auto-scaling mechanism to dynamically allocate and deallocate VMs on the basis of the current status of tasks. It begins by bundling tasks to reduce data transfer times and by distributing the overall deadline among tasks. Then, it creates a *load vector* by determining the most cost-efficient VM type for each task. This load vector is updated every scheduling cycle and indicates how many machines of each type are needed in order for the tasks to finish by their assigned deadline with minimum cost. Afterwards, the algorithm proceeds to consolidate partial instance hours by merging tasks running on different instance types into a single one. This is done if VMs have idle time and can complete the additional tasks by its original deadline. Finally, the earliest deadline first algorithm is used to map tasks onto running VMs, that is, the task with the earliest deadline is scheduled as soon as an instance of the corresponding type is available.

The SCS is an example of an algorithm that makes an initial resource provisioning plan on the basis of a global optimization heuristic and then refines it at runtime to respond to delays that were unaccounted for. The optimization heuristic allows it to minimize the cost and the runtime refinement to ensure there are always enough VMs in the resource pool so that tasks can finish on time. However, the refinement of the provisioning plan is done by running the global optimization algorithm for the remaining tasks every time a task is scheduled. This introduces a high computational overhead and hinders its scalability in terms of the number of tasks in the workflow.

5.14 | Summary

This section contains the classification of the surveyed algorithms on the basis of the presented taxonomy. In addition to this classification, Table 1 presents a summary indicating whether the algorithms were evaluated in real cloud environments or using simulation. This table also indicates whether the algorithms were evaluated using any the workflows presented in Section 2, randomly generated ones, or other scientific applications.

Table 2 displays the application model summary. Table 3 depicts the classification of the algorithms from the scheduling model perspective. In the task-VM mapping dynamicity category, *RR* refers to the hybrid runtime refinement class and *SS* to the hybrid subworkflow static one. In the resource provisioning strategy, the term *SP* is short for static VM pool. As for the algorithms in the dynamic elastic VM pool category, the abbreviation *CR* is used for constraint requirement while *PM* is used for performance metric. Finally, *HO* refers to the hybrid heuristic-optimal class in the optimization strategy category. Table 4 shows the resource model classification.

6 | CONCLUSIONS AND FUTURE DIRECTIONS

This paper studies algorithms developed to schedule scientific workflows in cloud computing environments. In particular, it focuses on techniques considering applications modeled as DAGs and the resource model offered by public cloud providers. It presents a taxonomy on the basis of a comprehensive study of existing algorithms that focuses on features particular to clouds offering infrastructure services, namely, VMs, storage, and network access, on a pay-per use basis. It also includes and extends existing classification approaches designed for general-purpose scheduling algorithms[44] and DAG scheduling in grids.[46] These are included as they are of particular importance when dealing with cloud environments and are complimented with a cloud-focused discussion. Existing algorithms within the scope of this work are reviewed and classified with the aim of providing readers with a decision tool and an overview of the characteristics of existing research. A description and discussion of various algorithms is also included, and it aims to provide further details and understanding of prominent techniques as well as further insight into the field's future directions.

The abundance of resources and flexibility to use only those that are required is a clear challenge particular to cloud computing. Most algorithms address this problem by elastically adding new VMs when additional performance is required and shutting down existing ones when they are not needed anymore. In this way, algorithms are careful not to overprovision so that cost can be reduced and not to underprovision so that the desired performance can be achieved. Furthermore, some algorithms recognize that the ability to scale horizontally does provide not only aggregated performance, but also a way of dealing with the potential indeterminism of a workflow's execution because of performance degradation. The difficulty of this provisioning problem under virtually unlimited resources calls for further research in this area. Efficiently utilizing VMs to reduce wastage and energy consumption[79] should be further studied. The maximum efficient reduction[79] algorithm is a recent step toward this goal. It was proposed as a post-optimization resource efficiency solution and produces a consolidated schedule, on the basis of the original schedule generated by any other algorithm that optimizes the overall resource usage (minimizes resource wastage) with a minimal makespan increase. Further, awareness and efficient techniques to deal with the provisioning and deprovisioning delays of VMs is also necessary. For example, pro-actively starting VMs earlier in the scheduling cycle so that tasks do not have to be delayed due to provisioning times may be a simple way of reducing their impact on the workflow's makespan.

The cost model of clouds is another challenge faced by algorithms. Most assume a model where VMs are leased with a static price and have a billing period longer than the average task execution time. Hence, they focus on reusing idle slots on leased VMs so that cost is reduced, as long as the performance objectives are not compromised. While this is true for most applications and providers offering coarse-grained billing periods such as Amazon EC2,[2] emergent services are offering more flexibility by reducing the length of their billing periods. For example, Google Compute Engine[58] charges for the first 10 minutes and per minute afterwards while Microsoft Azure[3] bills per minute. This finer-grained billing periods eliminate the need to reuse VMs to increase resource utilization and reduce cost and allows algorithms to focus on obtaining better optimization results. As a future direction, algorithms could focus on this specific scenario instead of focusing on either hourly billed instances or generic algorithms that work for any period length.

The growth in the development and adoption of sensor networks and ubiquitous computing sees a change in the data requirements of

applications. This creates an increased demand for scientific workflow management systems that are capable of supporting the processing of real time data produced by sensor or distributed devices. An example of a model of computation supporting this particular type of workflows is defined by Pautasso and Alonso[18] as *streaming pipelines* in which intermediate results are fed into continuously running tasks. Although some management systems such as Kepler[80] already incorporate this model, additional research on how to efficiently schedule workflows with the particular challenges and characteristics of streaming applications would greatly aid in supporting the scientific community.

Finally, an interesting and emerging service model is workflow as a service (WaaS). This type of platforms offer to manage the execution of scientific workflows submitted by multiple users and hence are directly related to scheduling algorithms designed to process multiple workflows simultaneously. Out of the surveyed algorithms, only a few target this application model. As the popularity and use of cloud computing becomes more widespread, so will services such as WaaS. Therefore, it is important to gain a better understanding and further investigate this type of algorithms. Multiple scenarios can be explored, for instance, the WaaS system may acquire a pool of subscription-based instances, and hence, algorithms may be concerned with maximizing their utilization, maximizing the profit of the WaaS provider, and supporting generic QoS requirements to suit the needs of multiple users. A recent step toward this model is presented by Esteves and Veiga.[81] They define a prototypical middleware framework that embodies the vision of a WaaS system and address issues such as workflow description and WfMS integration, cost model, and resource allocation. We refer the readers to this work for a more comprehensive understanding of this service model.

ACKNOWLEDGMENTS

We would like to thank Dr Rodrigo Calheiros, Dr Amir Vahid Dastjerdi, and Deepak Poola for their insights and comments on improving this paper.

REFERENCES

1. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Gener Comput Syst*. 2009;25(6):599–616.
2. Elastic compute cloud EC2. Available from: http://aws.amazon.com/ec2. Accessed June 2016.
3. Microsoft Azure. Available from: https://azure.microsoft.com. Accessed June 2016.
4. Yildiz U, Guabtni A, Ngu AH. Business versus scientific workflows: a comparative study. *Proceedings of the Fourth World Conference on Services (SERVICES)*. IEEE, Los Angeles, CA; 2009:340–343.
5. Barker A, Van Hemert J. Scientific workflow: a survey and research directions. *Parallel Processing and Applied Mathematics*. Springer, Gdansk, Poland; 2008:746–753.
6. Juve G, Chervenak A, Deelman E, Bharathi S, Mehta G, Vahi K. Characterizing and profiling scientific workflows. *Future Gener Comput Syst*. 2013;29(3):682–692.
7. Berriman GB, Laity AC, Good JC, Jacob JC, Katz DS, Deelman E, Singh G, Su MH, Prince TA. Montage: The architecture and scientific applications of a national virtual observatory service for computing astronomical image mosaics. *Proceedings of Earth Sciences Technology Conference*, College Park, Maryland; 2006.
8. Graves R, Jordan TH, Callaghan S, Deelman E, Field E, Juve G, Kesselman C, Maechling P, Mehta G, Milner K, et al. Cybershake: a physics-based seismic hazard model for southern california. *Pure Appl Geophys*. 2011;168(3-4):367–381.
9. Abramovici A, Althouse WE, Drever RWP, Gürsel Y, Kawamura S, Raab FJ, Shoemaker D, Sievers L, Spero RE, Thorne KS, et al. Ligo: the laser interferometer gravitational-wave observatory. *Science*. 1992;256(5055):325–333.
10. Livny J, Teonadi H, Livny M, Waldor M. K. High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas. *PloS one*. 2008;3(9):e3197.
11. USC epigenome center. Available from: http://epigenome.usc.edu. Accessed June 2016.
12. National Centre for Biotechnology Information. Available from: http://www.ncbi.nlm.nih.gov. Accessed June 2016.
13. Bharathi S, Chervenak A, Deelman E, Mehta G, Su Mei-Hui, Vahi K. Characterization of scientific workflows. *Proceedings of the Third Workshop on Workflows in Support of Large-Scale Science (WORKS)*, Austin, TX, USA; 2008:1–10.
14. Deelman E, Singh G, Su MH, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K, Berriman GB, Good J, et al. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Sci Program*. 2005;13(3):219–237.
15. Pandey S, Karunamoorthy D, Buyya R. Workflow engine for clouds. *Cloud Computing: Principles and Paradigms*, Hoboken, New Jersey, United States; 2011;87:321–344.
16. Fahringer T, Prodan R, Duan R, Nerieri F, Podlipnig S, Qin J, Siddiqui M, Truong HL, Villazon A, Wieczorek M. Askalon: a grid application development and computing environment. *Proceedings of the Sixth IEEE/ACM International Workshop on Grid Computing*. IEEE Computer Society, Seattle, Washington, USA; 2005:122–131.
17. Couvares P, Kosar T, Roy A, Weber J, Wenger K. Workflow management in condor. *Workflows for e-Science*. Springer; 2007:357–375.
18. Pautasso C, Alonso G. Parallel computing patterns for grid workflows. *Workshop on Workflows in Support of Large-Scale Science, 2006. WORKS'06.*, IEEE, Paris; 2006:1–10.
19. Ullman JD. NP-complete scheduling problems. *J Comput Syst Sci*. 1975;10(3):384–393.
20. Hu TC. Parallel sequencing and assembly line problems. *Oper Res*. 1961;9(6):841–848.
21. Mayr EW, Stadtherr H. Optimal parallel algorithms for two processor scheduling with tree precedence constraints. 1995.
22. Fishburn P. C. *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets*: John Wiley & Sons; 1985.
23. Yu J, Buyya R. A novel architecture for realizing grid workflow using tuple spaces. *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*. IEEE, Pittsburgh, USA; 2004:119–128.
24. Apache Jclouds. Available from: http://jclouds.apache.org. Accessed June 2016.
25. Gutierrez-Garcia JO, Sim KM. A family of heuristics for agent-based elastic cloud bag-of-tasks concurrent scheduling. *Future Gener Comput Syst*. 2013;29(7):1682–1699.
26. Michon E, Gossa J, Genaud S, et al. Free elasticity and free CPU power for scientific workloads on IaaS clouds. *Proceedings of the Eighteen IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, Singapore; 2012:85–92.
27. Villegas D, Antoniou A, Sadjadi SM, Iosup A. An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds. *Proceedings of the Twelfth IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, Ottawa, Canada; 2012:612–619.
28. Frincu ME, Genaud S, Gossa J, et al. Comparing provisioning and scheduling strategies for workflows on clouds. *Proceedings of the Second International Workshop on Workflow Models, Systems, Services and Applications in the Cloud (CloudFlow)*. IEEE, Boston, MA; 2013.

29. Schad J, Dittrich J, Quiané-Ruiz JA. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proc of the VLDB Endowment*. 2010;3(1-2):460–471.
30. Ostermann S, Iosup A, Yigitbasi N, Prodan R, Fahringer T, Epema D. A performance analysis of EC2 cloud computing services for scientific computing. *Cloud Computing*. Springer, Munich, Germany; 2010:115–131.
31. Gupta A, Milojicic D. Evaluation of HPC applications on cloud. *Open Cirrus Summit (OCS), 2011 Sixth*, Atlanta, Georgia; 2011:22–26.
32. Iosup A, Ostermann S, Yigitbasi MN, Prodan R, Fahringer T, Epema D. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Trans Parallel Distrib Syst*. 2011;22(6): 931–945.
33. 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. *Proceedings of the IEEE Second International Conference on Cloud Computing Technology and Science (CLOUDCOM)*. IEEE, Athens, Greece; 2010: 159–168.
34. Nabrzyski J, Schopf JM, Weglarz J. *Grid Resource Management: State of the Art and Future Trends*, vol. 64: Berlin, Germany: Springer Science & Business Media; 2012.
35. Valentin C, Ciprian D, Corina S, Florin P, Alexandru C. Large-scale distributed computing and applications: models and trends. 2010.
36. Berman F, Fox G, Hey AnthonyJG. *Grid Computing: Making the Global Infrastructure a Reality*, vol. 2: Hoboken, New Jersey, United States: John Wiley and sons; 2003.
37. Mao M, Humphrey M. A performance study on the VM startup time in the cloud. *Proceedings of the Fifth IEEE International Conference on Cloud Computing (CLOUD)*, IEEE, Honolulu, Hawaii, USA; 2012:423–430.
38. Amazon EC2 Spot Instances. Available from: https://aws.amazon.com/ec2/purchasing-options/spot-instances/. Accessed June 2016.
39. Maechling P, Deelman E, Zhao L, Graves R, Mehta G, Gupta N, Mehringer J, Kesselman C, Callaghan S, Okaya D, et al. SCEC CyberShake Workflows—Automating Probabilistic Seismic Hazard Analysis Calculations. *Workflows for e-Science*. Springer; 2007:143–163.
40. Deelman E, Singh G, Livny M, Berriman B, Good J. The cost of doing science on the cloud: the montage example. *Proceedings of the ACM/IEEE Conference on Supercomputing (SC)*. IEEE Press, Austin, Texas; 2008:Article No. 50.
41. Vöckler JS, Juve G, Deelman E, Rynge M, Berriman B. Experiences using cloud computing for a scientific workflow application. *Proceedings of the Second International Workshop on Scientific Cloud Computing (ScienceCloud)*. ACM, San Jose, California; 2011:15–24.
42. Malawski M, Juve G, Deelman E, Nabrzyski J. Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*. EEE, Salt Lake City, Utah; 2012:Article No. 22.
43. Tolosana-Calasanz R, BañAres JÁ, Pham C, Rana OF. Enforcing qos in scientific workflow systems enacted over cloud infrastructures. *J Comput Syst Sci*. 2012;78(5):1300–1315.
44. Casavant TL, Kuhl JG. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans Softw Eng*. 1988;14(2):141–154.
45. Kwok YK, Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput Surv (CSUR)*. 1999;31(4):406–471.
46. Yu J, Buyya R, Ramamohanarao K. Workflow scheduling algorithms for grid computing. *Metaheuristics for Scheduling in Distributed Computing Environments*. Springer; 2008:173–214.
47. Zeng L, Veeravalli B, Li X. Saba: a security-aware and budget-aware workflow scheduling strategy in clouds. *J Parallel Distrib Comput*. 2015;75:141–151.
48. Genez TA, Bittencourt LF, Madeira ER. Workflow scheduling for SaaS/PaaS cloud providers considering two sla levels. *Proceedings of the IEEE Network Operations and Management Symposium (NOMS)*. IEEE, Maui, Hawaii, USA; 2012:906–912.
49. Rodriguez MA, Buyya R. A responsive knapsack-based algorithm for resource provisioning and scheduling of scientific workflows in clouds. *Proceedings of the Fourty-Fourth International Conference on Parallel Processing (ICPP)*, vol. 1. IEEE; 2015:839–848.
50. Malawski M, Figiela K, Bubak M, Deelman E, Nabrzyski J. Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization. *Sci Program*. 2015;2015.
51. Talbi EG. *Metaheuristics: From Design to Implementation*, vol. 74: Hoboken, New Jersey, United States: John Wiley & Sons; 2009.
52. Amazon Simple Storage Service. Available from: http://aws.amazon.com/s3/. Accessed June 2016.
53. Amazon EFS. Available from: https://aws.amazon.com/efs/. Accessed June 2016. Accessed June 2016.
54. Google Cloud Storage. Available from: https://cloud.google.com/storage/. Accessed June 2016.
55. Rackspace Block Storage. Availeble from: http://www.rackspace.com.au/cloud/block-storage. Accessed June 2016.
56. Zhou A, He B, Liu C. Monetary cost optimizations for hosting workflow-as-a-service in iaas clouds. *IEEE Trans Cloud Comput*. 2015;PP(99):1–1.
57. Poola D, Ramamohanarao K, Buyya R. Fault-tolerant workflow scheduling using spot instances on clouds. *Procedia Comput Sci*. 2014;29:523–533.
58. Google Compute Engine. Available from: https://cloud.google.com/products/compute-engine/. Accessed June 2016.
59. Google Compute Engine Pricing. Available from: https://developers.google.com/compute/pricing. Accessed June 2016.
60. Cloudsigma. Available from: https://www.cloudsigma.com. Accessed June 2016.
61. Arabnejad H, Barbosa JG, Prodan R. Low-time complexity budget–deadline constrained workflow scheduling on heterogeneous resources. *Future Gener Comput Syst*. 2016;55:29–40.
62. Zhu Q, Zhu J, Agrawal G. Power-aware consolidation of scientific workflows in virtualized environments. *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, New Orleans, Louisiana, USA; 2010:1–12.
63. Rodriguez MA, Buyya R. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. *IEEE Trans Cloud Comput*. 2014April;2(2):222–235.
64. Durillo JJ, Prodan R. Multi-objective workflow scheduling in amazon ec2. *Cluster Comput*. 2014;17(2):169–189.
65. Topcuoglu H, Hariri S, Wu Min-you. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans Parallel Distrib Syst*. 2002;13(3):260–274.
66. Abrishami S, Naghibzadeh M, Epema DickHJ. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Gener Comput Syst*. 2013;29(1):158–169.
67. Calheiros RN, Buyya R. Meeting deadlines of scientific workflows in public clouds with tasks replication. *IEEE Trans Parallel Distrib Syst*. 2014;25(7):1787–1796.
68. Byun EK, Kee YS, Kim JS, Maeng S. Cost optimized provisioning of elastic resources for application workflows. *Future Gener Comput Syst*. 2011;27(8):1011–1026.
69. Byun EK, Kee YS, Kim JS, Deelman E, Maeng S. Bts: Resource capacity estimate for time-targeted science workflows. *J Parallel Distrib Comput*. 2011;71(6):848–862.
70. Pietri I, Malawski M, Juve G, Deelman E, Nabrzyski J, Sakellariou R. Energy-constrained provisioning for scientific workflow ensembles. *Proceedings of The Third International Conference on Cloud and Green Computing (CGC)*, IEEE, Karlsruhe, Germany; 2013:34–41.
71. Mao M, Humphrey M. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. *Proceedings of The International*

Conference for High Performance Computing, Networking, Storage and Analysis (SC), ACM, Seattle, WA, USA; 2011:Article No. 49.

72. Li HH, Fu YW, Zhan ZH, Li JJ. Renumber strategy enhanced particle swarm optimization for cloud computing resource scheduling. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*. IEEE; 2015:870–876.
73. Poola D, Garg SK, Buyya R, Yang Y, Ramamohanarao K. Robust scheduling of scientific workflows with deadline and budget constraints in clouds. *Proceedings of the Twenty-Eighth IEEE International Conference on Advanced Information Networking and Applications (AINA)*, Victoria, Canada; 2014:1–8.
74. Lee YC, Zomaya AY. Stretch out and compact: workflow scheduling with resource abundance. *Proceedings of the Thirteenth IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, Delft, Netherlands; 2013:219–226.
75. 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):521–552.
76. Zhu Z, Zhang G, Li M, Liu X. Evolutionary multi-objective workflow scheduling in cloud. *IEEE Transactions on Parallel and Distributed Systems*. 2016;27(5):1344–1357.
77. Wang J, Korambath P, Altintas I, Davis J, Crawl D. Workflow as a service in the cloud: architecture and scheduling algorithms. *Procedia Comput Sci*. 2014;29:546–556.
78. Zhou AC, He B. Transformation-based monetary cost optimizations for workflows in the cloud. *IEEE Trans Cloud Comput*. 2014:1–1.
79. Lee YC, Han H, Zomaya AY, Yousif M. Resource-efficient workflow scheduling in clouds. *Knowl-Based Syst*. 2015;80:153–162.
80. Altintas I, Berkley C, Jaeger E, Jones M, Ludascher B, Mock S. Kepler: an extensible system for design and execution of scientific workflows. *16th International Conference on Scientific and Statistical Database Management, 2004. Proceedings*. IEEE; 2004:423–424.
81. Esteves S, Veiga L. Waas: workflow-as-a-service for the cloud with scheduling of continuous and data-intensive workflows. *Comput J*. 2016;59(3):371–383.

How to cite this article: Rodriguez MA, Buyya R. A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. *Concurrency Computat: Pract Exper*. 2017;29:e4041. https://doi.org/10.1002/cpe.4041