J Supercomput
DOI 10.1007/s11227-015-1438-4

Workflow scheduling in cloud: a survey

Fuhui Wu[1] · **Qingbo Wu**[1] · **Yusong Tan**[1]

Abstract To program in distributed computing environments such as grids and clouds, workflow is adopted as an attractive paradigm for its powerful ability in expressing a wide range of applications, including scientific computing, multi-tier Web, and big data processing applications. With the development of cloud technology and extensive deployment of cloud platform, the problem of workflow scheduling in cloud becomes an important research topic. The challenges of the problem lie in: NP-hard nature of task-resource mapping; diverse QoS requirements; on-demand resource provisioning; performance fluctuation and failure handling; hybrid resource scheduling; data storage and transmission optimization. Consequently, a number of studies, focusing on different aspects, emerged in the literature. In this paper, we firstly conduct taxonomy and comparative review on workflow scheduling algorithms. Then, we make a comprehensive survey of workflow scheduling in cloud environment in a *problem–solution* manner. Based on the analysis, we also highlight some research directions for future investigation.

Keywords Cloud computing · Workflow scheduling · QoS constrained scheduling · Workflow-as-a-service · Robust scheduling · Hybrid environment · Data-intensive workflow scheduling

1 Introduction

Many applications consist of a number of cooperative tasks which typically require more computing power beyond single machine capability. These applications include

✉ Fuhui Wu
fuhui.wu@nudt.edu.cn

[1] School of Computer Science, National University of Defense Technology, Changsha 410073, China

Published online: 14 May 2015

scientific workflows [80,102], multi-tier web service workflows [106], and big data processing workflows such as MapReduce [49] from Google and Dryad [77] from Microsoft. An easy and popular way is to describe complex applications using high-level representation in workflow method.

The workflow scheduling problem has been studied extensively over past years, focusing on multiprocessor system [92] and distributed environments like grids [178] and clusters. Wieczorek et al. make a taxonomy of the multi-criteria grid workflow scheduling problem in [163]. The workflow scheduling is a general form of task scheduling problem, where the applications are modeled as directed acyclic graphs (DAGs). Workflow and DAG are usually interchangeable in the literature. It is a well-known fact that the complexity of DAG scheduling and its many variants are NP-complete [113]. Even in two simple cases [155]: (1) scheduling tasks with uniform weights to an arbitrary number of processors and (2) scheduling tasks with weights equal to one or two units to two processors, they were also proven to be NP-complete. There are three special cases for which there exist optimal polynomial-time algorithms. They are (1) scheduling tree-structured task graphs with uniform computation costs on arbitrary number of processors [73]; (2) scheduling arbitrary task graphs with uniform computation costs on two processors [44]; and (3) scheduling an interval-ordered task graph with uniform node weights to an arbitrary number of processors [63]. However, even in these cases, communication among tasks is assumed to take zero time unit [43]. Given these observations, the general workflow scheduling problem cannot be solved in polynomial-time unless $P = NP$.

The ever-growing data and computing requirements demand higher performance computing environment in order to execute these workflows in reasonable amount of time. Traditional HPC systems are mostly dedicated and statically partitioned per administration policy. This kind of system is not only very expensive, but also inefficient in adapting to the surge of resource demand. With the development of cloud, large-scale workflow applications are able to leverage the dynamically provisioned resources from cloud instead of using a dedicated supercomputer. As the cloud provides resources as services, users can access resources at anytime and anywhere. They only need to identify what type of resource to lease, how long to lease resources, and how much their applications will cost. Different from traditional system, where the scheduling methods are usually classified as best effort approach, some new features are added to the workflow scheduling problem. A common scenario in cloud system is that users want to run their applications as fast as possible with the minimum cost. However, if applications are not time-critical, users may tolerate a little delay of execution for more cost saving. These characteristics bring new research challenges on workflow scheduling technology. Novel approaches that address the particular challenges and opportunities of these technologies need to be developed.

There are various types of cloud providers [109], each of which has different product offerings. They are classified into a hierarchy of as-a-service terms: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). This survey focuses on IaaS clouds and PaaS clouds. The IaaS clouds offer the user a virtual pool of unlimited, heterogeneous resources that can be accessed on demand. To turn the long-held dream of computing as the 5th utility (after water, electricity, gas,

and telephony) into reality [30], virtualization technology is adopted to encapsulate infrastructure resources. It enables infrastructure resources being possessed in fine-grained pricing unit [64]. The PaaS clouds concern more logical scheduling than resource scheduling. The combination of IaaS and PaaS clouds forms a complete workflow scheduling structure. This makes it different from other utility paradigm such as utility grid. However, it also introduces new challenges:

1. Previous works developed for clusters and grids mainly focus on meeting application deadlines or minimizing the makespan without considering the cost of the utilized infrastructure. New scheduling algorithms are needed to be developed for cloud environment considering the pay-as-you-go model in order to avoid prohibitive and unnecessary costs.
2. In cloud environment, there are two main stages, resource provisioning and task-resource mapping, prior to the execution of a workflow. While in grids or clusters environment, most works focus only on the task-resource mapping stage, since these environments provide a static pool of resources whose configuration is known in advance. A critical issue in resource provisioning is to determine the amount and types of resources that workflow application would request, which affects both the total execution time (makespan) of workflow and the financial cost. It should also be able to dynamically and automatically accommodate unpredictable workload changes.
3. Virtual Machines (VMs) provided by current cloud infrastructures do not exhibit a stable performance [12,51,76,140]. This has a significant impact on scheduling policies rely on the estimation of task runtimes for different VMs in order to make a mapping decision. Moreover, the VM boot time and stop time [135] occupy an un-ignorable part of the whole VM life time, which is another aspect to be considered for cloud workflow scheduling.

Mapping performance requirements to the underlying resources in the cloud is the key problem. Resource under-provisioning will inevitably hurt performance while resource over-provisioning may result in idle instances, thereby incurring unnecessary costs. The goal of the mapping is to achieve "auto-scaling" in resource provisioning. Fundamentally, it should attain an optimal trade-off between performance and cost. This can be expressed in two primary optimization problems as: minimizing cost under deadline constraint or minimizing scheduling length (makespan) under budget constraint.

In this paper, we first propose a taxonomy for workflow scheduling algorithms based on available information of both resource and workflow. Then we make a comprehensive survey on workflow scheduling issues in cloud environment in a *problem–solution* manner. All issues are summarized into 10 problems of:

– Cloud resource accessing and pricing;
– Best-effort workflow scheduling;
– Deadline-constrained workflow scheduling;
– Budget-constrained workflow scheduling;
– Multi-criteria workflow scheduling;
– WaaS: workflow-as-a-service;
– Robust workflow scheduling;

– Hybrid resource scheduling;
– Data-intensive workflow scheduling;
– Energy-aware workflow scheduling.

The remainder of the paper is organized as follows. Section 2 describes the workflow model and definitions. Section 3 categorizes workflow scheduling algorithms into three taxonomies according to completeness of workflow and resource information. The workflow scheduling problems and state-of-the-art solutions are comprehensively surveyed in Sect. 4. The future research directions are discussed in Sect. 5. And Sect. 6 concludes the paper.

2 Modeling and definition

2.1 Workflow model

A popular representation of a workflow application is the directed acyclic graph (DAG): $G(T, E)$, where T is a set of $v = |T|$ tasks[1] $\{t_1, t_2, \ldots, t_v\}$, and E is a set of $e = |E|$ directed edges $\{e_i^j | (t_i, t_j) \in E\}$ representing inter-task data dependencies. Each task t_i represents an individual application task with a certain amount of computation workload wl_i, which usually takes million instructions (MI) as unit of measurement. Each edge e_i^j represents a precedence constraint which indicates that task t_i should complete executing before task t_j can start. If there is data transmission from t_i to t_j, the t_j can start only after all the data from t_i has been received. A number of d_i^j is associated with edge e_i^j to denote the amount of data to be transferred from t_i to t_j. The source task and the destination task of an edge is called the $parent$ task and the $child$ task, respectively. A task which does not have parent task is called an $entry$ task, whereas a task which does not have child task is called an $exit$ task. To generalize the DAG structure with only one $entry$ task and one $exit$ task, two dummy tasks t_{entry} and t_{exit} are usually adopted and added to the begin and end of the workflow, respectively. These dummy tasks have zero computation workload and zero communication data to actual $entry$ tasks and $exit$ tasks.

2.2 Definition

Table 1 summarizes the common notations used throughout this paper. Due to the NP-hard nature of DAG scheduling problem, it is hard to get the optimal values of parameters in Table 1. Hence, most of them are usually set with local optimal values derived from given local schedules. Moreover, precise definitions of the same notation may vary in different algorithms. Taking $rank$ for example, the communication cost is considered in [153], while not considered in [144].

Assuming each resource is associated with a property of computation capability, as million instructions per second (MIPS) for example. The communication startup

[1] Task is also referred to as node, subtask, activity, stage, job, or transformation in different works.

Table 1 Notation

Symbol	Definition
CCR	The communication to computation ratio
$makespan$	The overall schedule length of the workflow
C	The total cost of the execution of the workflow
D	The deadline of the workflow execution
B	The budget for the workflow execution
$ET_{t_i}^{r_l}$	The execution time of task t_i on resource r_l
$TT_{t_i}^{t_j}$	The transmission time from task t_i to task t_j
CP	The longest necessary path from t_{entry} to t_{exit}
$cp(t_i)$	The critical parent of task t_i
$cc(t_i)$	The critical child of task t_i
$pred(t_i)$	The parents of t_i
$succ(t_i)$	The children of t_i
$r(t_i)$	The resource where t_i executes on
ST_{t_i}	The start time of task t_i
FT_{t_i}	The finish time of task t_i
$rank(t_i)$	The attribute that denotes the priority of task t_i
$rank_u(t_i)$	The upward rank of task t_i traversing graph upward, starting from t_{exit}
$rank_d(t_i)$	The downward rank of task t_i traversing graph downward starting from t_{entry}
$EST_{t_i}^{r_l}$	The earliest start time of task t_i on resource r_l
$EFT_{t_i}^{r_l}$	The earliest finish time of task t_i on resource r_l
$LST_{t_i}^{r_l}$	The latest start time of task t_i on resource r_l
$LFT_{t_i}^{r_l}$	The latest finish time of task t_i on resource r_l
$RST(r_l)$	The leasing start time of resource r_l
$RET(r_l)$	The leasing end time of resource r_l
MET_{t_i}	The minimum execution time of task t_i on available resources
AET_{t_i}	The average execution time of task t_i on available resources

time (delay) of resource r_l is l_{r_l}, and the data transmission bandwidth between two resources r_l and r_k is $bw_{r_l}^{r_k}$. The computation and communication times are calculated as:

$$ET_{t_i}^{r_l} := \frac{wl_i}{\text{capability}(r_l)} \tag{1}$$

$$TT_{t_i}^{t_j} := \begin{cases} l_{r(t_i)} + \frac{d_i^j}{bw_{r(t_i)}^{r(t_j)}}, & r(t_i) \neq r(t_j); \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Then, the general form of workflow scheduling problem is to map tasks onto resources and order their executions so that task-precedence requirements are sat-

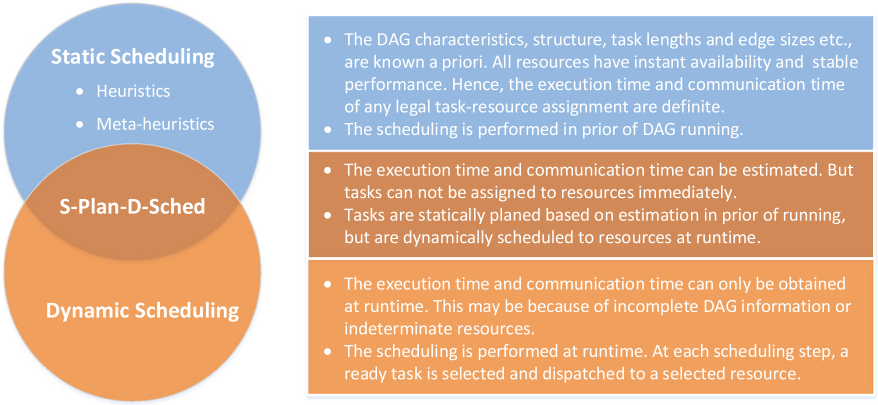

Fig. 1 Taxonomy of workflow scheduling

isfied. A schedule for a workflow represents the assignment of tasks to resources. It is defined as a $S =< R, M, makespan >$, where $R = \{r_1, r_2, \ldots, r_{|R|}\}$ is a set of used resources, and M consists of all task-resource mappings, with $m_{t_i}^{r_l} =< t_i, r_l, ST_{t_i}^{r_l}, FT_{t_i}^{r_l} >$, where $ST_{t_i}^{r_l}$ is the start time of task t_i on assigned resource r_l and $FT_{t_i}^{r_l}$ is the finish time of task t_i on resource r_l. The $makespan$ is the overall schedule length of S, which is defined as:

$$makespan = \max_{m_{t_i}^{r_l} \in M} \{FT_{t_i}^{r_l}\} \tag{3}$$

3 Taxonomy of workflow scheduling

To distinguish a variety of workflow scheduling methods, we classify them into three categories according to available information of workflow and resource and when tasks are assigned to resources, as shown in Fig. 1. Comparative studies in [25,101,165] show that static scheduling outperforms dynamic scheduling in most cases from different perspectives. The reason is that static scheduling is able to globally search from solution space at workflow level. Even at task level, static scheduling obtains more information than dynamic scheduling. However, static scheduling assumes that the accurate task execution and communication time can be obtained before scheduling, which is not always true in real systems. The task execution and communication time are determined by both resource and workflow information. Among these information, the workflow information include workflow structure, task execution workload, and communication data size, etc. And the resource information include availability, processing capability, and communication bandwidth, etc. In cloud environment, the performance fluctuation caused by multi-tenant resource sharing and failure caused by hardware and software should also be taken into consideration. In real production systems, all of them are proved to be incomplete and dynamic. Dynamic scheduling is able to handle these uncertainties but loses global optimization advantage of static

scheduling. S-Plan-D-Sced scheduling takes advantages of both static scheduling and dynamic scheduling. It makes static plan for all tasks based on approximate estimation [116,145] of task execution and communication time. At runtime, the assignment is adaptively tuned, and is rescheduled if necessary [93,182,189].

3.1 Static workflow scheduling

3.1.1 List scheduling heuristic

The basic idea of list scheduling is to make a scheduling list (a sequence of tasks for scheduling) by assigning them some priorities and sorting them according to their priorities, and then repeatedly execute the following two steps until all the tasks in the DAG are scheduled:

1. *Task selection* Select the first task from the scheduling list;
2. *Resource selection* Allocate the task to selected resource.

The scheduling list can be constructed either statically or dynamically. If all priorities are constructed before any task allocation, it is called static list scheduling. If the priorities of unscheduled tasks are recomputed after each task scheduling step, it is called dynamic list scheduling. No matter static or dynamic list scheduling, a prioritizing attribute and resource selection strategy are needed to decide task priorities and the optimal resource for each task.

Some classic list scheduling heuristics are modified critical path (MCP) [168], mapping heuristic (MH) [57], insertion scheduling heuristic [88] and earliest time first (ETF) [74]. Table 2 summarizes 5 typical List scheduling heuristics, heterogeneous earliest-finish-time (HEFT) [153], critical-path-on-a-processor (CPOP) [153], dynamic level scheduling (DLS) [144], dynamic critical path (DCP) [89], and predict earliest finish time (PEFT) [11].

3.1.2 Clustering heuristic

Both clustering-based scheduling and duplication-based scheduling (in Sect. 3.1.3) are designed to optimize transmission time between data dependent tasks. As tasks are scheduled separately in most list scheduling heuristics, schedule that does not take communication delay seriously would generate unnecessary idle fragments in resources. Figure 2 provides an intuition example of clustering and duplication scheduling. When scheduling t_3, resource r_2 is selected for smaller EFT using HEFT algorithm. After that, the start time of task t_4 will be delayed at 8 no matter it is assigned to r_1 or r_2 [Gant chart (b)]. Considering that communication delay from t_2 to t_4 is larger than ET of t_3, the $makespan$ can be shorten if t_3 is scheduled on r_1 through putting all tasks in a single cluster [Gant chart (c)]. Scheduling algorithm in (c) trades off parallelism in favor of communication delay. Through duplication of t_1 on r_2 [the dashed task in Gant chart (d)], the parallelism is fully exploited, leading to a $makespan$ of 8.

A general clustering heuristic consists of two parts:

Table 2 Overview of 5 typical List scheduling Heuristics

Heuristic	Attribute	Resource selection	F/S	CP	S/D	B	Complexity
HEFT [153]	$rank_u(t_i)$	$r_l : minimize\{EFT(t_i)\}$	F	×	S	√	$O(e \times \|R\|)$
CPOP [153]	$rank(t_i)$[a]	$\begin{cases} r_{cp}^{b} : t_i \in CP \\ r_l \quad : minimize\{EFT(t_i)\} \end{cases}$	F	√	S	√	$O(e \times \|R\|)$
DLS [144]	At each step, the *ready task* and available resource that maximize $dynamic\ level : DL(t_i, r_l) = rank_u(t_i) - EST_{t_i}^{r_l}$ is chosen for scheduling		F	×	D	√	$O(v^3 \times \|R\|)$
DCP [89]	$LST(t_i) - EST(t_i)$[c]	$r_l : minimize\{EST(t_i) + EST(cc(t_i))\}$	S	√	D	×	$O(v^3)$
PEFT [11]	$rank_{oct}(t_i)$[d]	$r_l : minimize\{EFT_{t_i}^{r_l} + OCT_{t_i}^{r_l}\}$	F	×	S	√	$O(\|R\|(e+v) + v^2\|R\|)$

Attribute The prioritizing attribute, *F/S* if the start time is fixed or set to be a sliding window when assigning a task to selected resource, *CP* if tasks on critical path will be scheduled firstly, *S/D* if it is static list scheduling or dynamic list scheduling, *B* if tasks are scheduled on bounded number of resources

[a] $rank(t_i) = rank_u(t_i) + rank_d(t_i)$

[b] r_{cp} all critical path tasks are assigned to a same resource

[c] The highest task with the smallest difference between its LST and EST is chosen

[d] $rank_{oct}(t_i) = \frac{\sum_{k=1}^{|R|} OCT_{t_i}^{r_k}}{|R|}$, $where\ OCT_{t_i}^{r_k} = \max_{t_j \in c(t_i)} \{\min_{r_l \in R} \{OCT_{t_j}^{r_l} + ET_{t_j}^{r_l} + TT_{t_i}^{t_j}\}\}$. The variate of OCT covers the impact of future assignment of tasks on different resources

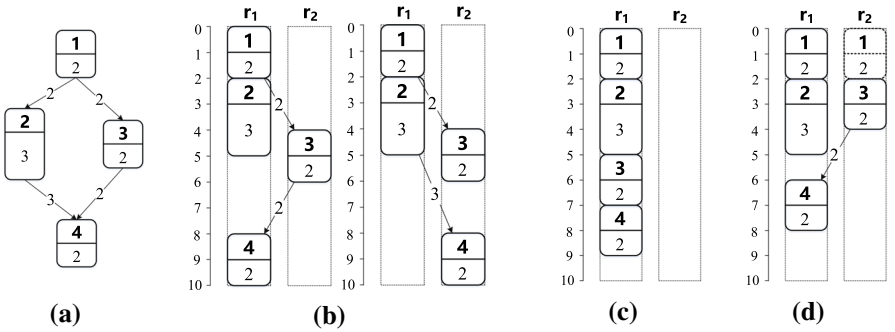

Fig. 2 An intuition example of clustering and duplication scheduling. The *upper* and *bottom* numbers in each *box* denote task id and ET, respectively. And the number at each *edge* denotes TT. **a** The example DAG. **b** Scheduling of task graph with HEFT algorithms. **c** Clustering scheduling. **d** Duplication scheduling

1. *Clustering* Mapping tasks to clusters;
2. *Ordering* Ordering tasks in the same cluster.

Each clustering of a DAG is a mapping of all tasks onto clusters, where each cluster is a subset of T and executes on a separate resource. The basic idea of clustering is to zero TT between tasks through clustering two tasks in a same cluster at the possible sacrifice of parallelism. Hence, there is a trade-off between maximizing parallelism and minimizing communication delay, which is also called a "max-min" problem in [87]. If there are two independent neighbor tasks in a same cluster, it is called nonlinear cluster; otherwise, it is called linear. For nonlinear clustering, an order is imposed for independent tasks. A linear clustering preserves the parallelism presented in original DAG, hence it does not increase the parallel execution time. But nonlinear clustering reduces parallelism by sequentializing parallel tasks, which may increase the parallel execution time. In [69], they further considered the question of linear clustering versus nonlinear clustering. Their analysis showed that the granularity, defined as communication to computation (CCR) ratio of DAG, affects the optimum trade-off point between linear and nonlinear clustering.

A clustering procedure can also be visualized as performing a sequence of clustering refinements. At step i, it merges clusters generated from last step of $(i-1)$ with the goal of reducing makespan. As it is NP-hard to get optimal scheduling in general, some non-backtracking clustering heuristics are proposed to avoid high complexity in the literature. Table 3 compares 4 typical clustering heuristics.

3.1.3 Duplication heuristic

The duplication technique is usually used in together with list scheduling [42] or clustering scheduling [120], either as a pure optimization procedure or as a new algorithm. The underlying concept of duplication heuristic is to properly duplicate task on the same resource with the target task such that TT between these two tasks is avoided. Therefore, the EST of tasks on that resource can be reduced and a better scheduling length can be achieved. The motivation of duplication-based scheduling lies in that, it may happen that some resources are idle during different time periods because the

Table 3 Overview of 4 clustering heuristics

Heuristic	Clustering	Ordering	Linear	CP	M. E. Z.	Complexity
DSC [171,172]	1. Choose the free task with the highest *rank* property	Non-insertion	×	√	√	$O(\log v(v+e))$
	2. Zero its incoming edges(s) in constraint condition of CT1[a] and CT2[b]					
KB/L [85]	Clustering all tasks on the current CP of unscheduled tasks	N/A	√	√	√	$O(v \times (v+e))$
Sarkar's [139]	Zero the highest communication edge if the makespan does not increase	Highest $rank_u$ first	×	×	×	$O(e \times (v+e))$
CASS-II [100]	1. Choose the current task with the highest *rank* property	Non-insertion	×	√	×	$O(e+v \times logv)$
	2. Clustering the task in the cluster containing its critical child[c]					

Clustering the mechanism of mapping tasks to clusters, *Ordering* the mechanism of task ordering in the same cluster, *Linear* if the heuristic is linear clustering or not, *CP* if tasks on CP will be scheduled firstly, *MEZ* if the heuristic does multiple edge clustering at one step

[a] CT1: The zeroing of an incoming edge of task t_i is accepted if it reduces $rank_d(t_i)$

[b] CT2: Zeroing incoming edges of a free node should not affect the strict reduction of task t_j on dominant sequence path at some future step

[c] CASS-II schedules all tasks from bottom to up by default. Hence, current tasks are composed of tasks whose immediate tasks have been examined already

tasks assigned to them are waiting for data from the tasks assigned to some other resources, even using an efficient scheduling algorithm. If these idle time slots can be utilized effectively by identifying the critical tasks and redundantly allocating them in these slots, the execution time of the parallel program can be further reduced. Figure 2d provides an intuition example of duplication scheduling. It assigns t_3 on a separate resource of r_2 to achieve parallelism. Due to precedence constraint of t_1, an idle slot of 4 time units is produced as shown in Fig. 2b. To fully utilize the idle slot and further reduce the start time of t_3, t_1 is selected and duplicated before t_3.

There are two issues to be addressed when designing an effective task duplication algorithm:

1. Which task(s) to duplicate? This concerns the selection of the parent tasks for duplication so that the start time of a child task can be minimized;
2. Where to duplicate the task(s)? This concerns how to locate a proper time slot on a resource to duplicate the parent task(s).

According to the selection of tasks to be duplicated, duplication-based algorithms can be divided into two classes: scheduling with full duplication (SFD) and scheduling with partial duplication (SPD). The SFD allows all tasks from higher levels of target task to be considered for duplication, including DSH [88], BTDH [42], LCTD [37], CPFD [6], PY [122], and TCSD [97]; whereas, SPD impose some restrictions during the task selection process, including TDS [48], LWB [45], PLW [120], DFRN [123]. Such restrictions lead to performance (less application parallel execution time) versus time complexity (longer time to carry out the scheduling algorithm itself) trade-off between these two categories. Duplication-based scheduling generally achieves shorter makespans. However, it also makes the scheduling problem more difficult. The scheduling algorithm not only needs to observe the precedence constraints among tasks but also needs to recognize which tasks to duplicate and how to fit them in the idle time slots. Table 4 makes an overview of those heuristics.

3.1.4 Meta-heuristics

As the DAG scheduling is a NP-complete problem, the optimal schedule cannot be resolved in polynomial time unless $NP = P$ [113]. Some meta-heuristics are proposed in the literature, as they usually provide an efficient way of moving quickly toward a very good solution.

Genetic algorithm is one of the most widely studied meta-heuristic for task scheduling problem. Typical examples for DAG scheduling include: [5,8,47,72,90,159,166,199]. These genetic algorithms differ in string representation ("genes") of the schedules in the search space, genetic operators for generating new schedules, fitness function to evaluate the schedules and stochastic assignment to control the genetic operators. It was shown that the improvement of GA-based solution to the second best solution was not more than 10 % and the GA-based approach required around a minute to produce a solution, while other heuristics required an execution of a few seconds [28].

Blythe et al. [25] investigated greedy randomized adaptive search procedure (GRASP) for workflow scheduling on grids, which gets better performance than the

Table 4 Overview of 10 duplication-based heuristics

Heuristic	Feature	List/clustering	Duplication type	Complexity
TDS [48]	Only the critical parent is duplicated	Clustering	Partial duplication	$O(v^2)$
LWB [45]	Lower bound of start-time is approximated	Clustering	Partial duplication	$O(v^2)$
PLW [120]	Lower bound of start-time is approximated	Clustering	Partial duplication	$O(v(e + v \log v))$
DFRN [123]	Duplication first and reduction next	List	Partial duplication	$O(v^3 \log v)$
DSH [88]	The utilization of idle slot is maximized	List	Full duplication	$O(v^4)$
BTDH [42]	Extension of the DSH	List	Full duplication	$O(v^4)$
LCTD [37]	Optimization of linear clustering	Clustering	Full duplication	$O(v^3 \log v)$
CPFD [6]	task on critical path is considered firstly	List	Full duplication	$O(e \times v^2)$
PY [122]	Lower bound of start-time is approximated	Clustering	Full duplication	$O(v^2(e + v \log v))$
TCSD [97]	Lower bound of start-time is approximated	Clustering	Full duplication	$O(v^3 \log v)$

List/clustering if the duplication is used with list scheduling or clustering scheduling algorithms

Min-Min heuristic for data-intensive applications. Young et al. [175] have investigated performance of simulated annealing (SA) algorithms for scheduling workflow applications in grid environment. Other global search oriented meta-heuristics, such as particle swarm optimization (PSO) and ant colony optimization (ACO) approaches, have also been investigated for grid and cloud environments in the literature. We will discuss them in the following sections.

3.1.5 Comparison of static scheduling

Compared to local search-based heuristics, meta-heuristics are able to search for schedules in larger solution space. Hence, the meta-heuristics perform better than local search-based heuristics in general. However, the overhead of scheduling time of meta-heuristics grows rapidly with the increasing of workflow task number, making them less appropriate for time sensitive workflows containing a large number of tasks.

The common goal of three heuristic categories is to efficiently partition DAG tasks into smaller chunks, and schedule these chunks on target resources to minimize the overall makespan. Most of list scheduling heuristics are for systems with bounded number of resources. This category can usually generate good-quality schedules whose performance are comparable to the other categories at lower scheduling overhead [91,131]. Clustering heuristics usually assume an unbounded number of resources, and assign heavy communicating tasks to the same resource. Although it reduces inter-resource communication, it may also lead to load imbalance or idle time slots when tasks are ordered on resources with respect to precedence constraints. Both list and clustering heuristics can cooperate with a duplication mechanism either as optimization or to form a new duplication algorithm. The duplication heuristics avoid large amount of communication time. Therefore, the ESTs of tasks can be reduced and a better scheduling length can be achieved. Duplication heuristics usually have better quality of schedules in sacrifice of computing time complexity.

Many algorithms in the literature assume unbounded number of available resource to achieve shorter schedule length, but they fail to truly address the practicality issue. In practice, it is not possible for a system to own unbounded number of resources. Even for algorithms designed for bounded number of resources, they do not have a mechanism to minimize the resource requirement without increasing schedule length. If the resource economization requirement is also taken into consideration, a mechanism is needed to minimize the resource requirement. Ahmad and Kwok proposed an Economical CPFD (ECPFD) on basis of CPFD for bounded number of resources [6]. It controls the degree of duplication according to the number of available resources. A more general idea is to add a second phase to merge the task clusters generated by some algorithm, assuming unbounded number of resources, onto bounded number of resources and order the task executions on each resource. Bozdağ et al. [27] firstly addressed the resource requirement minimization problem and proposed a generic algorithm, called schedule compaction (SC) with time complexity of $O(v^3)$. SC can be applied to the output of any scheduling algorithm. It preserves the schedule length of the original schedule and reduces resource requirement by merging resource schedules and removing redundant duplicate tasks.

3.2 Dynamic workflow scheduling

Dynamic scheduling is developed for handling the unavailability of scheduling information and resource contention with other workflow or non-workflow system load. Sonmez et al. [146] presented a taxonomy of dynamic scheduling policies based on three resource information (status, processing speed and link speed) and two task information (task length and communication data size). A particular piece of information can be either unknown, known a priori or obtained at runtime. Then, they map to this taxonomy seven policies across the full spectrum of information use. The performance of these policies are extensively analyzed through simulations and experiments in a real multicluster grid. They found that different scheduling polices are needed for different system conditions and workflow applications in order to attain good application execution performance.

A dynamic scheduling algorithm targets at balancing load among available resource queues. But it is not so easy to accurately evaluate the load of each queue. A dynamic workflow scheduler alone may still dispatch tasks to overcommitted queues, leading to a long waiting time. An easy way is to incorporate queue waiting time prediction technique with workflow scheduling. Accurately predicting the performance of a parallel and distributed applications has been studied extensively in the literature, including analytical [82], simulation-based and hybrid modeling [126] techniques. In [29], Brevik et al. firstly presented a method of binomial method batch predictor (BMBP) to predict bound on queuing delay, the amount of time an individual job will wait in queue before it is initiated for execution. Moreover, it obtains its predictions only from the observed history of previous waiting times, regardless of running environment and scheduling polices, etc. In [117], BMBP is implemented in an on-line service system QBETS. These characteristics make BMBP an attractive technique to select resource for workflow scheduling in [118,184].

In multi-workflow scheduling scenario, overload may occur due to the bursty nature of workload. As a consequence, the execution performance of the applications may deteriorate to unacceptable levels; response times may grow significantly, and throughput may degrade. A common way to achieve efficient overload control in distributed systems is to use admission control mechanisms [40,79]. Instead of dropping or rejecting tasks, *task throttling* is another method [146,152]. It delays the submission of some of the tasks or limits the total number of concurrent tasks in the system.

4 Workflow scheduling in cloud: state of the art

The most distinct feature that makes the workflow scheduling in cloud different from that in multiprocessor or grid system is utility. Scheduling in the later two system focuses on meeting deadlines or minimizing the makespan of a workflow. While the economic cost in the "pay-as-you-go" cloud environment is also important as performance to be considered.

Resources are general provisioned as virtual machines (VMs) in cloud. The virtualization technology is the base to construct an IaaS cloud, and is the most important technology that differs it from utility grid computing [64]. However, current virtual-

ization technology does not provide stable performance guarantee. Schad et al. [140] report an overall CPU performance variability of 24 % on Amazon's EC2 cloud. This is caused by resource sharing and competition between VMs co-scheduled on the same physical machine. The performance instability makes the scheduling methods relying on the estimation of task runtime unsuitable in this environment. Hence, new workflow scheduling methods should be developed for cloud environment.

4.1 Cloud resource model

The resource model describes the parallel computing environment to execute the workflow applications. It defines the component of processing element (PE) and connections between PEs.

4.1.1 Virtual machine resources

Generally, resource is instantiated as processor in traditional parallel machine and service in distributed grid environment. In IaaS cloud, resource is instantiated as virtual machine. Typically, cloud providers offer multiple VM types: $VT = \{vt_1, vt_2, \ldots, vt_{|VT|}\}$. Each type vt is associated with two attributes of capability and cost $< p, c >$, where p represents the overall processing capability of the VM type vt, including CPU speed, RAM size, I/O speed and network bandwidth. And c represents the overall monetary cost of leasing VM type vt per unit time, including all cost components such as VM initialization, program execution, and data communication.

4.1.2 Resource access and pricing modes

To attract different cloud users, IaaS cloud providers offer diverse resource access and pricing options, including reserved, on-demand and spot instances (VMs) with various discounts.

1. Reserved instance For users with predictable and steady demands, the reservation option allows user to prepay a one-time reservation fee and reserve computing instances for a long period (usually weeks, months or even years). During leasing period the usage is either free or charged under a significant discount [1,58,66]. However, whether and how much user can benefit from the reservation option critically depends on the application's resource demand pattern. Due to the prepayment of reservation fees, the cost saving of a reserved instance is realized only when the accumulated instance usage during the reservation period exceeds a certain threshold (varied from 30 to 50 % of the reservation period) [162].

2. On-demand instance For users with sporadic and bursty demands, the on-demand option allows them to lease resource in finer grained manner than that of reservation option. However, on-demand instances usually charge more than reserved instances of the same capability configurations. Moreover, on-demand option adopts coarse grained billing cycles. For example, Amazon Elastic Compute Cloud (EC2) charges

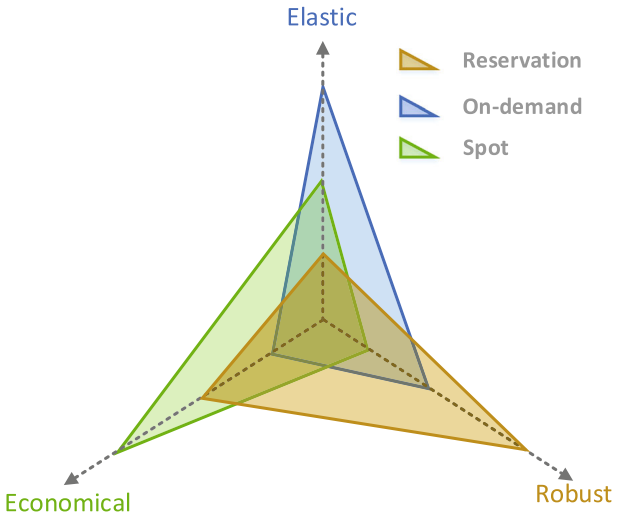

Fig. 3 Comparison of three instance access modes

on-demand instances based on hour billing model [1], whereby partial instance hour usage is rounded up to one hour.

3. Spot instance For market oriented cloud providers, they usually possess large quantities of spare capacity unsold. In order to incentivize clients to purchase spare capacity, spot option allows clients to bid on those capacity and grants resources to bidders if their bids exceed the periodically changing spot price. For example, EC2 [1] publicizes the spot price periodically but does not disclose how it is determined [17]. Spot instance is the product due to the evolution of cloud into economic market [158]. On the one hand, spot instance can be used to reduce monetary cost, because its price is usually much lower than that of on-demand instance of the same type. On the other hand, however, a spot instance may be terminated at any time when the bidding price is lower than the spot price (out of bid event). This may cause excessive long latency due to failures.

Figure 3 uses radar map to compare three instance access methods. They are compared in three dimensions of elastic, economical and robust. The higher value in elastic dimension denotes that the corresponding instance is more powerful in adapting to load changes. Three instance types are ordered in decreasing order of on-demand, spot, and reservation in elastic property. The robust property denotes the performance stability. The reservation instance has the most stable performance property, followed by on-demand instance and spot instance. As reservation instance can be accessed whenever it is needed, while a spot instance may be terminated for out of bid event. Towards building a more market-like cloud environment, spot instance has the highest potential to reduce monetary cost, followed by reservation instance and on-demand instance.

Tables 5, 6, 7 depict the capability configurations and prices of four general purpose instance types under different access modes. All the data are extracted from EC2 instance pricing covering a month of May 2014 in the US East region [1]. In general,

Table 5 Price($/h) and capabilities of four types of on-demand instances in Amazon EC2, May 2014, US East region

Instance type	vCPU	ECU	Mem (GB)	Disk (GB)	Price
m3.medium	1	3	3.75	1×4 SSD	0.070
m3.large	2	6.5	7.5	1×32 SSD	0.140
m3.xlarge	4	13	15	2×40 SSD	0.280
m3.2xlarge	8	26	30	2×80 SSD	0.560

Table 6 Prices($/h) statistics on four types of spot instances in Amazon EC2, May 2014, US East Region

Instance type	Price			
	Average	Stdev	Min	Max
m3.medium	0.018	0.032	0.008	0.452
m3.large	0.094	0.240	0.016	0.896
m3.xlarge	0.130	0.594	0.032	10.0
m3.2xlarge	0.103	0.202	0.064	3.6

spot instances are several times cheaper than on-demand instances of the same type. Based on load prediction, a proper reservation strategy can achieve significant instance discount, especially for high utilization and long reservation term.

4.1.3 Workflow scheduling in cloud

There are two main parts to execute workflow in cloud environment:

1. Resource provisioning. As resource in cloud is used in a "pay-as-you-go" mode, resource provisioning is separated from task scheduling;
2. Task Scheduling. Scheduling tasks to leased resources.

Previous works, developed for grids or clusters, mostly focus on the task scheduling part. The reason is that these environments usually provide a static resource pool where resources are already ready to execute tasks. While this is not the case in cloud environment, both parts are considered in order to generate efficient schedules.

In cloud environment, a schedule is defined as $S = (R, M, makespan, C)$ in terms of a set of resources, task to resource mappings, the schedule length, and the total monetary cost. $R = \{r_1, r_2, \ldots, r_{|R|}\}$ is a set of VMs that need to be leased; each resource r_i is of some VM type in VT and is associated with an resource leasing start time RST_{r_i} and resource leasing end time RET_{r_i}. M consists of all task-resource mappings, with $m_{t_i}^{r_l} =< t_i, r_l, ST_{t_i}^{r_l}, FT_{t_i}^{r_l} >$, where $ST_{t_i}^{r_l}$ is the start time of task t_i on assigned resource r_l and $FT_{t_i}^{r_l}$ is the finish time of task t_i on resource r_l. Hence, the total schedule length $makespan$ and total execution cost[2] C are defined as:

[2] In this paper, cost is referred to monetary cost. While the performance of an algorithm is described as time, such as execution time, etc.

Table 7 The upfront payment ($) and hourly price ($/h) of four types of reserved instances under different reservation options in Amazon EC2, May 2014, US East region

Instance type	Light utilization				Medium utilization				Heavy utilization			
	1-year term		3-year term		1-year term		3-year term		1-year term		3-year term	
	Upfront	Hourly	Upfront	Hourly	Upfront	Hourly	Upfront	Hourly	Upfront	Hourly	Upfront	Hourly
m3.medium	110	0.064	172	0.05	181	0.027	286	0.022	222	0.018	337	0.015
m3.large	220	0.127	343	0.101	362	0.055	571	0.043	443	0.037	673	0.03
m3.xlarge	439	0.254	686	0.201	724	0.109	1142	0.086	886	0.074	1345	0.06
m3.2xlarge	879	0.508	1372	0.401	1448	0.219	2283	0.173	1772	0.146	2691	0.12

$$makespan = \max_{m_{t_i}^{r_l} \in M} \{FT_{t_i}^{r_l}\} \tag{4}$$

$$C = \sum_{i=1}^{|R|} vt(r_i).c \times \left\lceil \frac{(RET_{r_i} - RST_{r_i})}{\tau} \right\rceil, \tag{5}$$

where $vt(r_i)$ is the VM type of r_i and $vt(r_i).c$ is the price of $vt(r_i)$ per billing cycle. The parameter τ denotes the length of billing cycle, which is specified by cloud provider.

4.2 Best-effort scheduling

To schedule workflow in cloud computing environment, application schedulers may have different objectives, including minimizing total execution time, minimizing total monetary cost, balancing the load among resources, and achieving stable performance, etc.

The best-effort scheduling attempts to optimize one objective while ignoring other factors such as various QoS requirements. Most algorithms in Sect. 3 belong to best-effort scheduling. The common objective of them is to minimize makespan. Pandey et al. [121] proposed a particle swarm optimization-based heuristic, that targets at minimizing total cost for data-intensive workflow applications. Both data transfer between resources as well as execution cost are taken into account. They formulated the problem as a Min–Max problem. The cost of each resource is computed separately, and the largest cost for all the resources is minimized. This indirectly ensures that the tasks are not mapped to a single resource and there will be a distribution of cost among the resources. In Bilgaiyan et al. [22] solved the same problem as [121]. Instead of PSO-based optimization, they used cat swarm optimization (CSO) method to obtain schedule in less number of iterations [41].

In contrast to best-effort scheduling, QoS-constrained workflow scheduling is more close to real-world applications. A QoS-constrained schedule tries to optimize some objective with constraints on other objectives. In Chen et al. [38], studied the workflow scheduling problem with various QoS requirements and proposed an ant colony optimization (ACO) approach. They addressed three of the most important QoS requirements: time, cost, and reliability. Their algorithm enables users to specify their QoS preferences as well as define the minimum QoS thresholds for a certain application. The target of their algorithm is to find a schedule that meets all QoS constraints and optimizes the user-preferred QoS objective.To made up a ACO algorithm, they presented 7 types of heuristic information and an adaptive scheme that allows artificial ants to select heuristic information based on pheromone values.

Among all the QoS-constrained workflow scheduling problems for cloud system, deadline-constrained and budget-constrained workflow scheduling are two primary categories that are widely studied in the literature.

4.3 Deadline-constrained workflow scheduling

Other than execution time, economic cost is another important requirement of cloud applications. Generally, faster resources are more expensive than slower ones. Hence,

the scheduler is confronted with a trade-off between execution time and monetary cost in selecting resources. To address this trade-off problem, a general form is to minimize monetary cost under deadline constraint, which can be formulated as:

$$minimize\,\{C\} \tag{6}$$

subject to,

$$makespan \leq D \tag{7}$$

4.3.1 Deadline distribution-based heuristic

The simplest solution for deadline-constrained workflow scheduling is minimum critical path (MCP). It only chooses cheaper resources for non-critical tasks, under the condition that execution of critical tasks is not influenced. Hence, the cost is reduced without increasing the makespan. If the makespan is smaller than deadline, there is potential to further reduce cost by delaying makespan to the deadline. To attain this target, deadline distribution is a main strategy in the literature that can get proper schedule with polynomial time complexity. A general deadline distribution heuristic consists of three phases: task partition, deadline assignment, and resource selection.

DTL [179], DBL [185], and DET [186] are three heuristics implementing the deadline distribution strategy. DTL and DBL partition the tasks into different levels in accordance of their parallel or synchronization properties. The difference between DTL and DBL is partition direction. DTL partitions tasks in top-bottom direction while DBL in the opposite direction. All tasks in the same level have the same sub-deadline. At the deadline assignment phase, DTL divides the overall deadline over task partitions in proportion to their minimum execution time (MET). While DBL divides the deadline by amortizing float time (calculated as: $deadline - makespan$[3]) uniformly to all levels. For the start time of a task, DBL does not require tasks of the same level to start at the same time as DTL. DBL relaxes this limit by starting from the finish time of critical parent. As opposed to leveling-based deadline distribution strategy, the DET algorithm partitions all tasks into different paths of an Early Tree [186]. The whole deadline is divided into time windows for critical tasks, which can be applied to all feasible deadlines. An iterative algorithm is proposed to determine the time windows for non-critical tasks while keeping the precedence constraints among tasks. At resource selection phase, DBL selects the cheapest resource while guaranteeing that the task execution can complete within its sub-deadline. DTL adopts Markov Decision Process (MDP) and DET adopts dynamic programming method to search local optimal resources for tasks.

Abrishami et al. [4] presented a Partial Critical Path (PCP) based task partition method evolved from their previously PCP algorithm [3] for grid environment. They proposed two algorithms of IC-PCP and IC-PCPD2 for scheduling scientific workflow in cloud. IC-PCP distributes deadline to partitions, while IC-PCPD2 further assigns

[3] the makespan is achieved by using fastest resources for all the tasks.

sub-deadline to each task in proportion to their METs. And resources are selected for partitions and tasks, respectively.

4.3.2 Meta-heuristic

Yu et al. [177] presented a genetic algorithm to optimize cost with deadline constraint. They proposed a 2-dimensional schedule representation method, and a fitness function encouraging less cost without violating deadline constraint. For genetic operators, they proposed a crossover operator and two mutation operator of *swapping mutation* and *replacing mutation*. Wu et al. [169] proposed a revised discrete particle swarm optimization (RDPSO) based workflow scheduling algorithm. Their optimizing objective is also to minimize total cost with deadline constraint. Both computation and communication costs are accounted in their algorithm. Rodriguez [135] presented a combined resource provisioning and task scheduling strategy for executing scientific workflows on IaaS cloud. They solved the problem using a meta-heuristic optimization algorithm of PSO. Some usually neglected factors as performance variation and VM boot time are taken into consideration in their algorithm. They designed a deadline constraint aware fitness function using the constraint-handling strategy proposed by Deb et al. [50].

4.4 Budget-constrained workflow scheduling

The intuition of this problem is to finish a workflow as fast as possible at given budget. A general form is to minimize makespan under budget constraint, which is formulated as:

$$minimize\ \{makespan\} \tag{8}$$

subject to,

$$C \leq B \tag{9}$$

4.4.1 One-time heuristic

One-time heuristic schedules workflow tasks only once without back-tracking. GreedyTimeCD [181], BHEFT [191,192], and HBCS [10] are three one-time heuristics. All of them are extended from HEFT [153], hence, the time complexities are the same of $O(e \times r)$. The differences between them exist at the resource selection phase. GreedyTimeCD (*greedy time-cost distribution*) statically distributes the overall user-defined budget to all tasks in proportion to their AETs before scheduling. Then, the resource that enables the task to finish earliest with cost non-larger than assigned sub-budget is selected. Budget-constrained heterogeneous earliest finish time (BHEFT) works just like GreedyTimeCD, except that budget is distributed dynamically. They presented a current task budget (CTB) quantity, which distributes current spare budget to unscheduled tasks in proportion to their AETs. For HBCS algorithm, resource with highest aggregation weight of *worthiness* over normalized execution cost and normalized execution time is selected. If remaining budget is high, time factor has more influence; and if the remaining budget is small, the cost factor becomes dominant one for $worthiness$.

4.4.2 Back-tracking-based heuristic

The novel idea of back-tracking heuristic is to start from an assignment which has good performance under one of the two optimization criteria considered (that is, makespan and budget) and swap tasks between resources trying to optimize as much as possible for the other criterion. There are two approaches implementing this idea. The first one is "LOSS" which starts with an assignment of optimal makespan, meaning the cost is over budget. And then, it keeps swapping tasks between resources by choosing tasks with smallest loss weight in (10). The intuition in $LossWeight$ is to minimize execution time loss for the largest execution cost savings. The "LOSS" approach terminates when the execution cost is less than budget. Conversely, the second one, named "GAIN", starts with the cheapest assignment of tasks onto resources (that is, the one that requires the least money), and then keeps swapping tasks between resources by choosing tasks with largest gain weight in (11). The intuition in $GainWeight$ is to get the biggest benefit in makespan for the smallest money cost. It terminates when there is no more money available (budget exceeded). Sakellariou et al. [138] observed that "LOSS" approach gets better makespan than "GAIN" approach, because "LOSS" approach starts with assignment that is generated by efficient DAG scheduling heuristic. The "GAIN" approach only performs better in makespan when the available budget is close to the cheapest budget.

$$\text{LossWeight}(t_i, r_l) = \frac{ET_{\text{new}} - ET_{\text{old}}}{EC_{\text{old}} - EC_{\text{new}}} \tag{10}$$

$$\text{GainWeight}(t_i, r_l) = \frac{ET_{\text{old}} - ET_{\text{new}}}{EC_{\text{new}} - EC_{\text{old}}} \tag{11}$$

Lin et al. [99] formulated the task scheduling problem with target of minimizing end-to-end delay under budget constraint, referred to as minimum end-to-end delay under cost constraint (MED-CC) problem. The MED-CC problem requires that each task is assigned to a separate VM. The scheduling target is to minimize end-to-end delay, which is calculated as: $MED = \min\{\sum_{t_i \in T} ET_{t_i}\}$. They proposed a Critical-Greedy algorithm using "GAIN" approach. It only imposes rescheduling on dynamic critical tasks. Zeng et al. [188] also proposed a backtracking algorithm, referred to as *ScaleStar*. The resources are selected based on the value of a proposed metric of comparative advantage (CA), including CA1 for initial scheduling and CA2 for backtracking scheduling. In addition, an adjustment policy, referred to as *DeSlack*, is proposed to remove part of slack without adversely affecting the overall makespan and the total monetary cost. *ScaleStar* is able to generate efficient schedule at the sacrifice of computing complexity, as it needs to recompute the makespan and cost for every rescheduling and adjustment step.

4.4.3 Meta-heuristic

The genetic algorithm approach presented by Yu et al. [177] (see, Sect. 4.3.2) and [176] is also able to search for time optimal solutions with budget constraint. The only

difference is the fitness function that encourages the algorithm to choose schedules with earliest completion time without violating budget constraint.

4.5 Multi-criteria workflow scheduling

Multi-criteria workflow scheduling means to take into account several objectives, which are often conflicting, simultaneously.

4.5.1 Aggregation approach

The most straightforward solution for multi-objective optimization problem is to transform it to a mono-objective optimization problem. The major problem with this approach is to define a proper aggregation weight. In Li et al. [98] proposed a CCSH heuristic, extended from HEFT. Their objectives are to minimize makespan and minimize cost. They introduced a cost-conscious weight factor between the EST and task execution cost for selecting resources. Garg et al. [67] presented three novel heuristics through aggregating conflicting objectives of makespan and cost into single objective.

Fard et al. [59] proposed a list scheduling algorithm for multi-objective DAG scheduling. They presented a constraint vector partitioning mechanism that distributes multi-objectives to tasks. To compose a list scheduling algorithm, tasks are selected using the attribute of $rank_u$, and resource selection is approximated by applying a double strategy: maximizing the distance to the constraint vector for dominant solutions and minimizing it if there is no dominant solution.

Dogan et al. [53] solved a bi-criteria (performance and reliability) optimization problem using aggression approach. They proposed a reliable dynamic level scheduling (RDLS) algorithm extended from DLS [144] algorithm. RDLS adds an incremental cost variate to the dynamic level (DL) metric of DLS. In Dogan et al. [54] improved their approach using a genetic algorithm and proposed an algorithm of bi-objective dynamic level scheduling (BDLS). They used SWGR [18] technique to aggregate two objectives into one fitness function. Hakem et al. [70] used aggregation approach to achieve two conflicting objectives of performance and reliability simultaneously. They proposed a list scheduling heuristic of BSA, and introduced a bi-objective compromise function for resource selection. In Dongarra et al. [55] presented a product metric of $(\lambda_j \times \tau_i^l)$, where λ_j is the failure rate of resource r_j and τ_i^l is an expression of performance as $FT_{t_i}^{r_l}$. They also proposed a reliable-HEFT (RHEFT) algorithm using this product metric.

Lee et al. [94] proposed an adaptive dual-objective scheduling (ADOS) algorithm as a novel semidynamic scheduling heuristic, which targets at reducing makespan and increasing resource utilization simultaneously. The algorithm generates a random schedule as an initial solution, which undergoes the manipulation process of ADOS repeatedly for further improvement in makespan and/or resource utilization. This schedule manipulation involves a *branch-and-bound-style* technique and two types of mutation. At runtime, it adopts rescheduling technique to tackle performance fluctuation.

4.5.2 ϵ-approach

This approach is usually adopted for bi-criteria scheduling problems, that targets at optimizing two conflicting criteria: a primary criteria and a secondary criteria specified as a sliding constraint function of the primary criteria. The intuition is that the more important criteria c_1 is minimized and let c_1^* be the *obtained optimal value*, and then the second criterion c_2 is minimized under the additional constraint $c_1 \leq c_1^*$, here ϵ is defined as c_1^* or $(c_1^* - c_1^{opt})$. The value of c_1^{opt} is the theoretical optimal value of c_1. The motivation of this approach is that user is usually not able to set a fixed constraint for one criteria, (as he may not know in advance the realistic criteria values), nor is he able to express his preferences by means of aggregation weight which can be assigned to multi-criteria.

Wieczorek and Prodan et al. [129,164] used ϵ-approach to solve a bi-criteria scheduling problem. They described the problem as an extension of the *multiple-choice knapsack problem* and proposed an algorithm called DCA based on dynamic programming method.

4.5.3 Pareto approach

Different from previous two approaches that get only one schedule as the final solution. Pareto approach generates a set of non-dominated schedules, offering more flexibility to users when estimating their QoS requirements.

Bessai et al. [19] used pareto approach to solve a bi-criteria (execution time and cost) workflow scheduling. Their algorithm is implemented on basis of list scheduling. They proposed three pareto algorithms in total in accordance with three resource selection polices, *cost-based*, *time-based*, and *cost-time-based*. For each algorithm, some candidate schedules are searched using three different task selection strategies of *top-down*, *bottom-up*, and *mixed*. The *mixed* strategy start by selecting the tasks belonging to the intermediate level. Then, the tasks belonging to the level smaller or larger than starting level are selected using *bottom-up* and *top-down* strategies, respectively. Among all candidate schedules, non-dominated schedules are maintained as pareto schedules.

Yu et al. [180] used multi-objective evolutionary algorithms (MOEAs) to search for pareto front schedules. Their approach is able to simultaneously minimize two conflicting objectives of execution time and execution cost while meeting deadline and budget constraints. To attain this goal, corresponding fitness functions for the objectives and penalty functions for the constraints, have been developed. Two population-based MOEAs, NSGAII, DEB2002FAST and SPEA2 [198], and one local search-based MOEA, PAES [86] are implemented and compared for different workflow structures and constraint levels. Talukder et al. [150] solved the same problem using *Multi-objective Differential Evolution* (MODE) approach.

4.6 WaaS: workflow-as-a-service

Unlike single workflow instance scheduling, there could be multiple workflow instances submitted to the application, and the workload may change all the time.

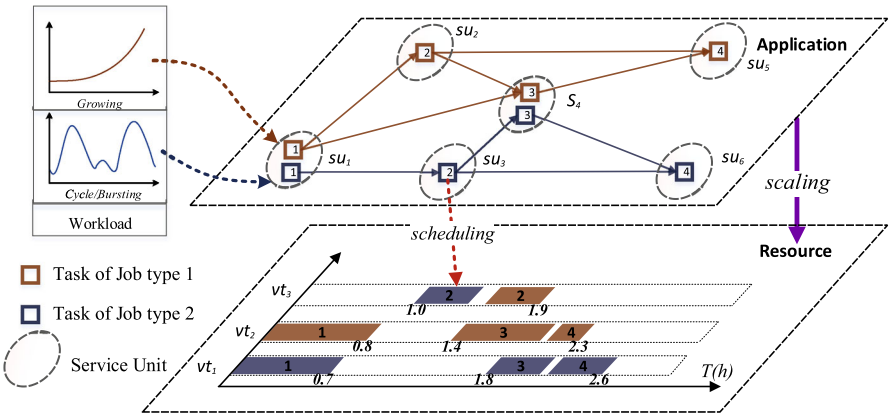

Fig. 4 Architecture of WaaS

We call this kind of workflow scheduling problem in cloud as WaaS problem. Figure 4 depicts a general architecture of WaaS. The application provider deploys application in cloud and provides service for application users. A cloud application consists of a set of functional service units $SU = \{su_1, su_2, \ldots, su_{|SU|}\}$, with each denotes an abstraction of processing components as order submission and data persistence steps for an online shopping site, or data reduction and compression steps of a scientific workflow. The application users submit jobs (workflows) following some workload arrival patterns. An application supports multiple Job types $JT = \{jt_1, jt_2, \ldots, jt_{|JT|}\}$, with each expressed as a DAG consisting of a subset service units of SU. In Fig. 4, the application composes of 6 service units $SU = \{su_1, su_2, su_3, su_4, su_5, su_6\}$. Two Job types $JT = \{jt_1, jt_2\}$ are supported by this application, where $jt_1 = \{< su_1, su_2 >, < su_1, su_4 >, < su_2, su_4 >, < su_2, su_5 >, < su_4, su_5 >\}$ and $jt_2 = \{< su_1, su_3 >, < su_3, su_4 >, < su_3, su_6 >, < su_4, su_6 >\}$. A service unit can be part of different Job types. As the t_3 of jt_1 and t_3 of jt_2 do the same work of su_4. Two workloads of *Growing* workload and *Cycle/Bursting* workload are submitted to this application. The WaaS system scales in/out resources from cloud provider according to workload changes and schedules tasks to leased resources. In Figs. 2, 4 VMs of vt_1, 2 VMs of vt_2 and 1 VM of vt_3 type are leased to execute 2 jobs of jt_1 and jt_2 separately, assuming that VM instances are priced by hour. The *makespan*s are 2.3 and 2.6.

4.6.1 Elastic resource provisioning

A critical challenge to schedule workflow workload in cloud is to determine the right amount of resources required for the execution of workflows in order to minimize the economic cost from the perspective of users and to maximize the resource utilization from the perspective of resource providers. Byun et al. [33] proposed an algorithm of partitioned balanced time scheduling (PBTS), which estimates the minimum number of resources required to execute a workflow within a user-specified finish time. PBTS is extended from their previously proposed approach of BTS [31,32]. The BTS algorithm

is motivated by a simple idea that a task can be delayed as long as its finish time constraint and other tasks can exploit the slack time created by such artificial delay. The BTS targets at reducing the number of resources required to execute the workflow. In PBTS, it improves the idea of BTS to support elastically adjusting the number of computing resources at runtime. It firstly determines the distribution of tasks across per time charge unit in order to determine the resource capacity of each time charge unit. Then the best number of computing resources per time charge unit is computed using BTS algorithm.

4.6.2 Merge-based multiple DAG scheduling

The scheduling of single DAG has been studied extensively in the literature. To handle multiple workflow scheduling, a strait forward method is to merge multiple DAGs into a single DAG, and schedule all tasks in single DAG scheduling manner. Hönig et al. [71] used this method to improve overall parallelism and resource utilization. Zhao et al. [190] proposed 4 composition approaches to merge DAGs and 2 fair scheduling policies to schedule tasks of merged DAG. The merge-based approach is able to utilize static scheduling advantages. However, it is not suitable for dynamic workloads, the arrival of whose job following some special patterns.

4.6.3 Hierarchical multiple DAG scheduling

To handle workload of any arrival pattern, dynamic scheduling, which works in a ready-and-schedule manner, is a natural way. In Stavrinides et al. [147] evaluated several algorithms implemented on basis of this idea. The ready task selection phase is either based on the earliest deadline first (EDF), highest level first (HLF) or the least space-time first (LSTF) policies. For resource selection, first fit (FF), best fit (BF) and worst fit (WF) bin packing techniques are employed to exploit idle time slots. Their results show that the combination of EDF and BF outperforms all the other examined heuristics. Tsai et al. [154] elaborated on the direction of Stavrinides and developed a new approach to further improve multiple workflow scheduling performance through two techniques. The first is clustering technique, proposed by Bittencourt et al. [23]. The tasks within workflows are clustered into groups for allocation instead of allocating individual tasks to reduce inter-task communication time. The second technique makes a balance between task start time and fitness of idle time slots instead of pure *Best Fit* allocation.

Hierarchical scheduling cooperates static and dynamic scheduling to generate promising solutions. Iverson et al. [78] devised a decentralized two-level strategy. At the high-level, it maps task to resource group using list scheduling of DLS [144] algorithm. While at the low-level, it selects the resource that is able to finish task earliest from mapped group. The scheduling of separate DAG is unaware of the others. They are co-scheduled in a competition manner. In Yu et al. [183] designed a planner-guided scheduling strategy made up of three components: DAG Planner, Task Pool and Executor. For each arrived DAG, an instance of DAG Planner locally prioritize tasks using the HEFT [153] mechanism. The Task Pool stores ready tasks of all submitted DAGs. And then, the Executor globally re-prioritizes all tasks in Task Poll

and maps tasks to available resources. The re-prioritization set the DAG which gets closer to completion with higher priority. This method helps to reduce turnaround of the majority, the total time between submission and completion of a DAG.

4.6.4 Auto-scaling of cloud resources

The auto-scaling problem is a basic problem for utility cloud computing. It targets at offering cloud resources in an autonomic way. It has already been studied for simple applications such as batch-queue application [108]. For WaaS applications, it is more complicated due to precedence constraints between tasks and deadline and budget constraints imposed on each DAG.

Malawski et al. [104] addressed a problem concerning the efficient management of ensembles, made up of inter-related workflows. Their target is to complete as many high-priority workflows as possible under budget and deadline constraints in IaaS clouds. They developed three algorithms with different features. The threshold-based dynamic provisioning dynamic scheduling (DPDS) algorithm is an online algorithm. workflow-aware DPDS (WA-DPDS) introduces a workflow admission control mechanism for DPDS. And static provisioning static scheduling (SPSS) takes the advantage of static scheduling. However, their work is unable to handle unpredictable workloads. Mao et al. [106] described an auto-scaling mechanism for WaaS application, with each DAG job specified with a deadline. It firstly preprocesses submitted DAGs and distributes sub-deadlines [138,179] to tasks. And then, they designed a *Load Vector* technique to calculate resource requirement for dynamic resource scaling. After that, tasks are scheduled on resources using earliest deadline first (EDF) algorithm. In Mao et al. [107] studied budget-constrained auto-scaling of cloud workflows. The budget here is for the whole cloud application instead of every single DAG job. They firstly addressed the budget constraint in the form of dollars per time quantum (h), which is consistent with the current prevailing hourly pricing scheme in the cloud [1]. Based on this budget model, they proposed two auto-scaling algorithms of scheduling-first and scaling-first. Scheduling-first algorithm distributes the application-wide budget to each individual job, determines the fastest execution plan and then acquires cloud resources. While scaling-first algorithm determines the size and the type of the cloud resources first and then schedules the workflow jobs on acquired resources. Zhou et al. [195] designed a general framework, named *Dyna* for hosting WaaS in IaaS cloud. When workflow arrives, static optimization is imposed on it. It uses $A^{\star}$ search method to generate a monetary cost minimized resource configuration, which offers probabilistic performance guarantees. Moreover, spot instances are used to refine and get a better configuration. At runtime, dynamic optimization of consolidation and scaling technique are also adopted to further optimize execution cost as in [106].

For WaaS application, submitted workflows may have a diversity of requirements on performance and cost. They are traditionally handled by specific or *ad hoc* strategies. Zhou et al. [194] presented a general framework of *ToF* to support flexible settings. ToF has two major components for performance and cost optimizations: transformation models and planner. The transformation models define a set of transformation operations, and the planner performs the transformation on the workflow according to user's requirement setting.

4.7 Robust workflow scheduling in cloud

4.7.1 Performance variations and failures

Most static scheduling algorithms are based on an assumption that task execution and data transmission time can be estimated precisely. While real cloud systems are faced with performance variation and failure problems, which bring uncertainty on workflow execution.

As VMs are deployed in a shared resource pool, where multiple VMs are consolidated in one physical machine, the performance of VM may be not definite when executing workflows. VMs can share CPUs and main memory surprisingly well in cloud environment, while I/O performance is hard to isolate [39,111]. A performance variation of 4–16 % is observed due to network and disk I/O interferences [12]. Dejun et al. [51] analyzed the EC2 performance for resource provisioning of service-oriented applications. They demonstrated that the CPU and disk I/O performance of small instances are relatively stable from the perspective of a long-running periods. However, the performance behaviors of multiple "identical" small instances differs greatly. Studies [76,140] have also demonstrated significant variances in I/O and network performances for Many-Tasks scientific computing and data-intensive MapReduce applications. Moreover, the performance variations also arise for factors like load changing, etc.

Failure is another kind of uncertainty, including hardware failures (e.g., host crash, network partition, etc.), and software errors (e.g., memory leak, numerical exception, etc.). When spot instances are used, failure also happens due to out-of-bid event.

The performance uncertainty in cloud system affects the overall workflow execution and may increases the makespan. To handle this problem, robust workflow scheduling is needed to absorb some degree of uncertainty, especially for cost-critical and time-critical applications.

4.7.2 Robustness metrics

A schedule is said to be robust if it is able to absorb some degree of uncertainty in workflow execution while maintaining a stable solution. The robustness alone is not a metric but gives an intuitive notion of the stability of the solution with regards to other performance metrics such as makespan. There is no consensus on a single metric, and almost each paper uses its own metric depending on the studied problem. Ali et al. [7] give a good definition of how to measure robustness using the following steps: (1) define the performance features that need to be robust; (2) identify the parameters that impact the robustness; (3) identify how a modification of these parameters impact on the performance features; (4) identifying the smallest collective variation of the parameters that make the performance features to violate acceptable variation. With this methodology, an example of measuring the robustness metric is as following: the performance metric is the makespan, the parameters that impact the robustness are task execution and data communication times. Hence, a schedule is said more robust than another one if it requires a greater change of the task execution or data

Table 8 Robustness metrics

Metric	Definition
Makespan mean	The average makespan of the schedule
Makespan standard deviation	For two schedules, the one with the smaller standard deviation is more likely to have makespans close to the average makespan
Makespan differential entropy	The differential entropy, used in [26], is a measure of the uncertainty for a random variable. If the uncertainty is small, the schedule is robust
Robustness probability	Shestak et al. [142] used the probability that the makespan is within two bounds as robustness metric. If it is high, the makespan of the schedule is likely to be close to the average makespan and the schedule is robust
Tolerance time	The mount of time a workflow can be delayed without violating the deadline
Lateness likelihood	Shi et al. [143] defined the lateness likelihood as the probability of makespan to exceeds a threshold. If the lateness likelihood is large, it means that the makespan tends to be late and the robustness is low
Makespan 0.99-quantile	This metric measures the worst makespan it is possible to have in 99 % of cases
Slack mean	Task slack time represents a time window within which the task can be delayed without extending the makespan. Bölöni et al. [26] also proposed a metric of slack time that gives the sum of spare time in the schedule. If a schedule has a large slack time, it intuitively denotes that the schedule is able to absorb more uncertainty

communication time. However, it is still hard to make comparisons of these changes between schedules.

To model the uncertainty, stochastic method is the most widely used technique. Taking makespan as the performance metric, the robustness of a schedule is the stability of the makespan. For a given stochastic task graph, the makespan of a schedule will also follow some probability distribution. Some easy to calculate and compare metrics are proposed in the literature as shown in Table 8.

Canon et al. [34] provided a comprehensive study of these robustness metrics. It is proven that several of them are equivalent due to the implication of central limit theorem. Consequently, the simplest standard deviation is sufficient to express robustness for most cases. However, the computation of the probability density or cumulative distribution of makespan is non-trivial [52,103]. Two main solutions are *probability theory-based method* and *Monte Carlo method*. The first method uses complicated mathematics, while the second is easier to implement at the expense of a lot of sampling. Moreover, modeling uncertainty by probability distribution presupposes much knowledge (for instance statistical), which is little available in real environment.

Fuzzy methodology is another mathematic tool to model uncertainty. Fayad et al. used fuzzy sets to model the uncertain execution times and proposed a genetic algorithm [62] and tabu search algorithm [61] to search for the best solution that would

minimize the number of tardy tasks. Dubois et al. [56] adopted possibility theory, on basis of fuzzy set [115], to model imprecision in task scheduling. The evaluation of possibility distribution of makespan is much simpler than the probability distribution. However, it cannot give the same insight as stochastic evaluation and misses some stochastic aspects.

4.7.3 Robust workflow scheduling

1. Makespan minimization The intuition for makespan minimization method is that, a schedule with smaller mean value of makespan is considered better in uncertain computing environment. Moreover, Cannon et al. [35] suggested that robustness and makespan are somehow correlated, schedules that perform well statically tend to be the most robust.

To mitigate the impact of uncertainties and still achieve good performance, a straightforward idea is *just-in-time (dynamic) scheduling*. It means that every task is scheduled only when it becomes ready. Malewicz et al. [105] and Cordasco et al. [46] have developed heuristics based on this idea. They consider only the DAG structure to improve application performance. While in the evaluation of Szepieniec et al. [149] showed that the effectiveness is limited without considering task execution time.

Another idea is to adapt static scheduling algorithms to the uncertain environment and keep the advantages of static scheduling. Based on this idea, the first kind of implementation is to overestimate execution times of individual tasks. This results in a waste of resources as it induces a lot of idle time slots during the execution if the task durations are much shorter than the estimation. Another kind of implementation is to dynamically reallocate tasks to idle resources during the execution. The allocated tasks may be rescheduled according to an assessment of variations during run-time. Sakellariou et al. [137] proposed a low-cost rescheduling policy, which considers rescheduling at a few, carefully selected points to reduce rescheduling times. However, rescheduling a task is costly as it implies some extra communication and synchronization overheads.

Study in [137] also indicates that, in addition to rescheduling, it is important to have a static schedule with good properties before the start of the execution. Even for dynamic strategy, a good initial assignment would reduce the possibility of making a bad decision [35]. Hence, a full-ahead scheduling scheme is helpful to address stochastic DAG scheduling problems. Zheng et al. [193] proposed a novel Monte Carlo approach, named MCS. Each MCS scheduling consists of two phases of *producing* phase and *selecting* phase. The first phase generates a number of schedules using a deterministic scheduling heuristic from pre-configured m samples of the stochastic DAG. The second phase selects the best schedule for pre-configured n samples of the stochastic DAG. Experimental evaluation showed that the best schedule always outperforms the schedule that directly applies the deterministic scheduling heuristic.

2. Slack maximization As the mean value of makespan cannot fully describe the distribution of random variable of makespan, other metrics should be adopted as scheduling objectives. Among all metrics in Table 8, slack mean is the only metric that

does not need to compute probability distribution. Hence, it is selected as a complement to makespan with little computing complexity increasing. Then, the problem can be formalized as a multi-objective optimization problem where minimizing the makespan and maximizing the mean value of slack are two objectives.

Poola et al. [127] presented a critical path-based heuristic with deadline and budget constraints. It presented three resource allocation policies with robustness, makespan and cost as its objectives. Users can select from these policies according to their priorities and get schedules that are aligned to their priorities. The heuristic information for selecting robust VMs are defined by the amount of slack time. The robust of the whole schedule is measured by two metrics of *robustness probability* and *tolerance time*. Shi et al. [143] also evaluated the robustness of a schedule as in [127]. They used ϵ-method [157] to solve a bi-objective (makespan and robustness) optimization problem. They proposed a genetic algorithm to search optimal schedules. It takes schedule generated from HEFT [153] as initial generation, and takes *slack mean* as fitness function to select next generation.

3. Fault-tolerant To handle failures, Hwang et al. [75] categorized techniques into two classes of task level and workflow level failure handling. Task-level technique has been widely studied for parallel and distributed systems. They can be divided into 4 types of retry, alternate resource, checkpoint/restart and replication techniques. The simplest task-level techniques are retry and alternate [151], as it simply tries to reexecute the same task on the same or alternate resource after failure. Checkpoint is especially efficient for long-running applications. After checkpoint, failed tasks can restart from the failure point. Among checkpointing techniques, Dome [14], Fail-safe PVM [96] and CoCheck [148] are libraries that enable coordinated checkpoint for parallel computing platforms, while Libckpt [125] is a standalone and portable checkpoint library for uniprocessor platforms. The replication technique [2] runs the same task simultaneously on different resources to ensure at least one of the replicas succeeds. Workflow-level techniques refer to manipulate workflow structure (e.g., modifying execution flows) to deal with erroneous conditions. In [75], two workflow-level techniques of alternative task and redundancy were proposed. The designed a flexible failure handling framework combining these techniques together in [75].

Wang et al. [160,161] used checkpoint and rescheduling techniques to handle task failures for workflows with large volume of data generated during their execution. They introduced a customized risk assessment model, that provides precise risk evaluation by taking the task dependency into consideration and assigning higher weights to tasks close to the end. Thereafter, they presented an output data backup mechanism based on this model. To schedule tasks that maximizes the robustness and minimize the possible of failure, they proposed a risk-aware resource selection method using genetic algorithm.

4. Spot instance Premature termination of spot instance is a new field of robust workflow scheduling. The introduction of spot instance reduces execution cost, up to 70 % [128] against using on-demand instance only on the one hand. It also challenges DAG scheduling for failure handling caused by uncertain spot price on the other hand. To address this problem, Poola et al. [128] proposed a *just-in-time* scheduling algo-

rithm. The key idea is to efficiently map ready tasks to spot instances or on-demand instances based on a parameter of latest-time-to-on-demand (LTO). LTO is the difference between the deadline and the critical path length at current scheduling point. An intelligent bidding strategy is presented to minimize the cost. To handle failures, they adopted the checkpointing technique, which is proven to be effective for spot markets [174]. In Zhou et al. [195] presented a hybrid instance configuration refinement mechanism. Their algorithm firstly uses $A^\star$ search approach to get a cost optimal schedule that meets deadline requirement. Then the refinement mechanism is imposed on the static schedule to get a hybrid instance configuration, under constraint that the deadline is still met.

4.8 Hybrid environment

In a hybrid cloud environment, the cloud usually plays a complement role. The public cloud resources can be elastically aggregated to the private resource pool when necessary. Bittencout et al. [24] proposed a hybrid cloud optimized cost (HCOC) scheduling algorithm, which decides how cloud resources are aggregated to the private cloud to finish workflow with least economic cost under deadline constraint. HCOC starts with an initial schedule generated by path clustering heuristic (PCH) algorithm [23]. Then HCOC iteratively reschedules tasks to cloud resources until the makespan is smaller than deadline. Ostermann et al. [119] solved the same problem of insufficient grid resource through aggregating cloud spot resources. A DCP-C algorithm was extended from dynamic critical path (DCP) [89]. DCP-C implements a resource selection mechanism that enables spot resource bidding. To raise the level of application programming over hybrid grid and cloud system, VGrADS [133] designed a system framework enables managing heterogeneous resources, scheduling of computation and data movement, fault tolerance, and performance tuning. VGrADS separates application development from resource management. Two main components are virtual grid execution system (vgES) and *Workflow Planner*. The vgES component is responsible for providing a uniform virtual grid abstraction atop widely differing resources, and *Workflow Planner* schedules tasks to feasible resources provided by vgES.

For commercial multicloud environment, workflow scheduling is still an open issue to be addressed. In a commercial multicloud environment, individual providers focus on increasing their own revenue and do not care about the utility of users and other providers. In such an environment, we cannot trust the information presented by cloud providers. Frad et al. [60] addressed this problem using algorithmic mechanism design approach, which is an intersection between game theory and computer science. They proposed an truthful mechanism of BOSS that forces the self-interested agents to tell the truth. The mechanism is implemented through a reverse auction. For an auction, a task is the auctioneer that starts an auction to select a proper resource. And then, each cloud bids for the task. The winner is selected based on the minimum product of cost and time across all clouds. Based on BOSS, a list scheduling heuristic is proposed, where BOSS is imposed on each task repeatedly to select proper resource.

4.9 Data-intensive workflow scheduling

Workflows are of loosely coupled parallel applications that consist of a set of computational tasks linked via data- and control-flow dependencies. Workflow tasks typically communicate through the use of files, while tightly coupled applications, such as MPI jobs, communicate directly via the network.

In Juve et al. [81] described data management issues when deploying scientific workflows in Amazon EC2 cloud environment. They examined the performance and cost of 4 storage systems: *Amazon S3*, *NFS*, *GlusterFS*, and *PVFS*. They also presented some meaningful conclusions that: the first write penalty on ephemeral disks is the major factor inhibiting storage performance on EC2; the choice of storage system has a significant impact on workflow execution time; and the cost closely follows performance, etc.

Several researchers have addressed data-aware workflow scheduling problem, in which large data sets associated with scientific workflows are taken into account when scheduling tasks. Park et al. [124] proposed a bandwidth allocation technique to speed up file transfers. Bharathi et al. [20] presented several data staging techniques for data-intensive workflows, and demonstrated that decoupled data staging can reduce the execution time of workflows significantly. Ramakrishnan et al. [132] evaluated a dynamic method that minimizes the storage space by removing data files at runtime when they are no longer needed.

4.10 Energy-aware workflow scheduling

As power is now widely recognized as a first-class design constraint for modern computing systems, the power management is another critical issue for the clouds hosting thousands of computing servers. A single high-performance 300 W server consumes 2628 KWh of energy per year, with an additional 748 KWh in cooling, which also causes a significant adverse impact on the environment in terms of CO_2 emissions [21]. A typical datacenter with 1000 racks can well consume nearly 10 MW of electricity power per day [134]. Reportedly, datacenters now consume about 1.3 % of the worldwide electricity and this fraction will grow to 8 % by 2020 [65]. The energy consumption in cloud datacenter raises many important environmental and economic issues. It attracts high attention of cloud owners and operators.

Due to the importance of energy consumption, it also become a critical issue for workflow scheduling. To attain the target of energy consumption reduction, there are two types of solution. The first one is to reduce energy consumption through improving the utilization of resources [15,16,110]. In [197], they observed that tasks of a given scientific workflow can have substantially different resource requirements, and even the resource requirements of a particular task can vary over time. They used a dimensionality reduction method (KCCA) to relate the resource requirements to performance and power consumption, and developed a pSciMapper framework for task consolidation. PASTA [141] is an two-phase power aware solution to schedule workflows on heterogeneous computing resources. Their contributions are a resource selection mechanism that offers a fair trade-off between power efficiency and overall

makespan, and a new list-based scheduling algorithm for task scheduling on selected resource.

The other solution is to use dynamic voltage scaling (DVS) technique [167]. Venkatachalam et al. [156] investigated and developed various techniques including DVS, resource hibernation, and memory optimizations. Among these techniques, DVS has been proven to be a very promising technique for energy savings [68,136,196]. Pruhs et al. [130] theoretically analyzed the dual objectives of makespan and energy minimization problem using DVS technique. They showed that the search space can be restricted to those with constant power schedules (i.e., the sum of power at which machines run is constant over time). They also showed how to reduce this problem to obtain $ln(v)$ approximation algorithms. However, the data transmission is not considered in their theory. In Kim et al. [84] leveraged DVS technique to optimize energy consumption in deadline-constrained Bag-of-Task applications. Both space-shared and time-shared resource provisioning policies were also exploited in DVS-enabled cluster. In Kim et al. [83] extended their work to handle the problem of power aware Virtual Machine provisioning for real-time cloud services.

For workflow scheduling problem, the most straight forward method is to generate a schedule using makespan best-effort algorithm firstly, and then adopt DVS technique to tune slack times of generated schedule [13,36]. Lee et al. [95] devised a novel objective function and a variant that effectively balance two goals of makespan and energy consumption minimization. And then they proposed a list scheduling heuristic based on those two objective functions and an tuning mechanism. Mezmaz et al. [112] and Yassa et al. [173] formalized the same problem in [95] as a multi-objective optimization problem, and solved it using meta-heuristic algorithms of genetic algorithm (GA) and particle swarm optimization (PSO) algorithm separately.

Apart from computation-intensive workflows, data-intensive workflows also attracts plenty of attention in the big data processing context. MECDS [170] addressed the data-intensive workflow scheduling in virtualized cloud systems. It addressed the energy consumption problem with a large volume of intermediate data to store and retrieve. In [114], they focused on workflows with deadline constraint and studied the energy consumption problem. They proposed a static power management scheme based on the degree of parallelism for an already generated schedule. To consider the run-time behavior of tasks, they also devised an on-line dynamic power management technique to further explore the idle periods of processors.

5 Discussion on research directions

Based on the related literature, we find that the following issues have not been sufficiently studied towards programming the cloud using workflow paradigm. These are the gaps in the reviewed work that would prove to be directions for future works.

5.1 Cost/performance programming

Performance oriented scheduling has been extensively studied in the past on both homogeneous and heterogeneous systems. In utility cloud computing environment,

however, performance heterogeneous is replaced by cost/performance heterogeneous. Cloud providers offer resources of different capabilities and prices to satisfy various requirements. It also challenges the workflow scheduling on cost/performance heterogeneous resources.

The pricing model of most current commercial clouds charges users on basis of the number of time intervals they have occupied, which makes it different from utility grid and brings new challenges. The key problem is resource fragment optimization. Most cost minimizing methods are designed for utility grid environment, where charging unit is ignored. The cost can be further reduced by treating the problem as a bin-packing problem, in which charging units are considered as bins. The problem is to determine the minimal number of bins to execute all the tasks. The instance consolidation [107] is also an option to handle the time quantum fragmentation problem which has not been sufficiently discussed in the literature.

5.2 Auto-scaling for WaaS

A critical challenge in integrating workflow technologies with resource provisioning technologies in cloud is to determine the right amount of resources required. An efficient resource provisioning method can minimize the economic cost from the perspective of users and maximize the resource utilization from the perspective of resource providers. The aim of automatic resource scaling for workflow execution is to deliver workflow applications as a service.

Executing dynamic concurrent workflows together on shared resources will improve resource utilization and reduce cost for the users. In analogy to the real world, a service level agreement (SLA), which can be regarded as a bilateral contract between user and cloud provider, is usually specified to capture user's QoS requirements and acts as a guarantee of the expected QoS. If the terms of the SLA are fulfilled, user is expected to pay some fee to the provider. Otherwise, the provider may have to pay some penalty to the user. To meet the user's SLA without risking its failure, an efficient planning with quick admission control [191] is able to decides whether or not a submitted workflow is acceptable, and enhances the user experience.

Workload prediction is an efficient technique that enables the auto-scaling component to prepare earlier for the incoming workload bursting. This helps to reduce the influence of delayed instance acquisitions and avoid VM churn. If the workload information can be known in advance using prediction techniques, better budget allocation, resource acquisition and task scheduling decisions can be made. However, there are still few works about workload prediction of workflow in the literature.

5.3 Robust workflow scheduling

5.3.1 Performance fluctuation

The most important problem when implementing algorithms in real environment is the uncertainties of task execution and data transmission time. It will be interesting to study how to guarantee performance for workflow using cloud resources. Moreover,

most works assume that a workflow has definite DAG structure. However, a workflow can have loops or conditional branches as in BPEL [9] or Spark [187], and dynamical adding task may also occur at runtime. These situations make the performance guarantee even harder.

5.3.2 Failure handling

In a super large scale cloud computing center, failure becomes a common phenomenon due to both hardware and software reasons. Failure handling is an important direction for robust workflow scheduling. Another kind of failure stems from the market nature of cloud computing. Previous studies consider only fixed pricing scheme. Cloud computing is evolving towards an economic market, where resources are dynamically priced. Take the Amazon EC2 spot instance for example, the prices are determined by market demand and supply. On the one hand, spot instances can be used to reduce monetary cost, because the price of spot instances are usually much lower than on-demand instances of the same type. On the other hand, however, a spot instance may be terminated at any time when the bidding price is lower than the spot price. The use of spot instances may cause excessive long latency due to failures. Robust scheduling that can absorb some degree of this kind of failure uncertainty is more likely to meet the bilateral contract of SLA.

5.4 Data-intensive workflow scheduling

In most studies, data transfer between tasks is not directly considered, data uploading and downloading are assumed as part of task execution. However, this may not always be the case, especially in the big data era. For a data-intensive application, data movement activities can dominate both the execution time and cost. In cloud system, storage resources are also virtualized like computing resources. For example, cloud storage such as S3 can be used for more efficient data broadcasting and higher data availability. It is important to consider the intermediate data transfer performance explicitly, and design the data placement strategies for resource provisioning decision-making. To employ VMs that are deployed on different regions, the data transfer cost between data centers is a complicated problem to be addressed.

5.5 Others

It is necessary to investigate hybrid environments, including private cloud and public cloud, that provide heterogeneous resources. Flexible resource selection is important to take advantages of all kinds of resources.

Although there are a lot of work on energy optimization for cloud datacenter, the energy consumption problem of workflow scheduling is still a relative new field in the literature. Both software and hardware methods can be explored for energy consumption optimization of cloud workflow scheduling. The software method includes resource utilization improving, space and time sharing of resources. Based on DVS and virtualization hardware technologies, there is still promise of energy consumption

optimization. Moreover, most of already exist researches either do not take energy consumption of data transmission into consideration or use straight forward method for data transmission. For energy generating sources, little work considered storing energy, and networking energy other than computing energy. As an ecosystem, the CO_2 emission problem and leveraging renewable energy will also be important directions in the future.

6 Conclusions

Cloud computing aims at delivering hardware infrastructure and software applications as services. The users can consume these services based on a SLA which defines their required QoS levels, on a pay-as-you-go basis. Although workflow scheduling has been extensively studied in multiprocessor and grid systems, the cloud workflow scheduling still remains a challenging field in the literature. In this paper, we survey this scheduling problem in three dimensions of: economic, elastic, and robust scheduling.

We firstly introduce the DAG model and definitions, followed by a general problem statement of workflow scheduling. To better understand the design of algorithms for workflow scheduling, we make a taxonomy on scheduling algorithms based on the availability of workflow and resource information. They are categorized into three types: *Static Scheduling*, *S-Plan-D-Sched*, and *Dynamic Scheduling*. *Static Scheduling* is again divided into heuristic algorithm and meta-heuristic algorithm. In turn, heuristic algorithms consist of *List Heuristic*, *Cluster Heuristic*, and *Duplication Heuristic*. *Static Scheduling* outperforms *Dynamic Scheduling* in general, as it is able to make workflow level scheduling. However, *Static Scheduling* needs to make precise estimation beforehand, which is not always attainable in real production system, especially for cloud environment. *Dynamic Scheduling* handles these uncertainties with the loss of global scheduling advantage. Hence, *S-Plan-D-Sched* is proposed to take both advantages of *Static Scheduling* and *Dynamic Scheduling* in real systems.

Then, we survey cloud workflow scheduling issues in a *problem–solution* manner. We consider 9 problems in all. The cloud resource accessing and pricing problem is firstly discussed using our three-dimensional model. To optimize cloud workflow scheduling, best-effort scheduling algorithms attempt to complete execution at the earliest finish time or with the least monetary cost. QoS-constrained scheduling is developed for applications with some QoS requirements. Among QoS constraints, deadline and budget are two mostly concerned constraints in the literature. For deadline-constrained scheduling, deadline-distribution-based scheduling algorithms are main proposed heuristics. For budget-constrained scheduling, one-time scheduling and back-tracking-based algorithms are two primary heuristics. The one-time scheduling maps each task to resource only once. While the back-tracking algorithm usually starts with an schedule generated using best-effort algorithm. And then, the initial schedule is tuned to make use of unused budget and meet the budget constraint. Multi-criteria scheduling attempts to optimize multiple conflicting objectives. Multi-criteria scheduling can generate either one schedule or a number of schedules, named pareto schedules. Aggregating, ϵ-, and pareto scheduling are three multi-criteria scheduling approaches. WaaS attempts to provide workflow application as a service. The target of

WaaS is to elastically provide resources in accordance with workload changes. Performance fluctuation and failure are two major uncertainties in shared cloud environment. Robust scheduling is developed to absorb these uncertainties as well as premature termination of spot instances in market environment. For full marketed environment in the future, there will be heterogeneous resources coexist, as hybrid grid and cloud, private cloud and public cloud, and multi-cloud environments. Flexible resource choosing mechanisms for workflow scheduling should be developed. In face of big data era, the programming paradigm has changed from computing-intensive to data-intensive programming. This challenges workflow scheduling in data accession, transmission, and storing in cloud environment.

Finally, we have discussed future research directions. Although there are already numerous works on workflow scheduling on parallel and distributed systems, workflow scheduling in cloud is still a new and hot academic research field. Moreover, there are few researches on scheduling workflows on real cloud environment, and much fewer cloud workflow management systems, which need further academic study and industrial practice.

Acknowledgments This work is supported by project (Grant No. 2013AA01A212) from the National 863 Program of China, project (Grant No. 61202121) from the National Natural Science Foundation of China, Science and technology project (Grant No. 2013Y2-00043) in Guangzhou of China.

References

1. Amazon ec2 pricing. http://aws.amazon.com/ec2/pricing/
2. Abawajy JH (2004) Fault-tolerant scheduling policy for grid computing systems. In: Proceedings of parallel and distributed processing symposium, 2004, 18th international, IEEE, p 238
3. Abrishami S, Naghibzadeh M, Epema DH (2012) Cost-driven scheduling of grid workflows using partial critical paths. IEEE Trans Parallel Distrib Syst 23(8):1400–1414
4. Abrishami S, Naghibzadeh M, Epema DH (2013) Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Gener Comput Syst 29(1):158–169
5. Ahmad I, Dhodhi MK (1995) Task assignment using a problem genetic algorithm. Concurr Pract Exp 7(5):411–428
6. Ahmad I, Kwok YK (1998) On exploiting task duplication in parallel program scheduling. IEEE Trans Parallel Distrib Syst 9(9):872–892
7. Ali S, Maciejewski AA, Siegel HJ, Kim JK (2004) Measuring the robustness of a resource allocation. IEEE Trans Parallel Distrib Syst 15(7):630–641
8. Ali S, Sait SM, Benten MS (1994) Gsa: Scheduling and allocation using genetic algorithm. In: Proceedings of the conference on European design automation, IEEE, pp 84–89
9. Andrews T, Curbera F, Dholakia H, Goland Y, Klein J, Leymann F, Liu K, Roller D, Smith D, Thatte S et al (2003) Business process execution language for web services
10. Arabnejad H, Barbosa JG (2014) A budget constrained scheduling algorithm for workflow applications. J Grid Comput, pp 1–15
11. Arabnejad H, Barbosa JG (2014) List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans Parallel Distrib Syst 25(3):682–694
12. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I et al (2010) A view of cloud computing. Commun ACM 53(4):50–58
13. Baskiyar S, Abdel-Kader R (2010) Energy aware dag scheduling on heterogeneous systems. Clust Comput 13(4):373–383
14. Beguelin A, Seligman E, Stephan P (1997) Application level fault tolerance in heterogeneous networks of workstations. J Parallel Distrib Comput 43(2):147–155
15. Beloglazov A, Abawajy J, Buyya R (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Gener Comput Syst 28(5):755–768

16. Beloglazov A, Buyya R (2012) Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr Comput Pract Exp 24(13):1397–1420
17. Ben-Yehuda OA, Ben-Yehuda M, Schuster A, Tsafrir D (2013) Deconstructing amazon ec2 spot instance pricing. ACM Trans Econ Comput 1(3)
18. Bentley PJ, Wakefield JP (1996) An analysis of multiobjective optimization within genetic algorithms. Tech Rep 96:1–14
19. Bessai K, Youcef S, Oulamara A, Godart C, Nurcan S (2012) Bi-criteria workflow tasks allocation and scheduling in cloud computing environments. In: Proceedings of IEEE 5th international conference on cloud computing (CLOUD), IEEE, pp 638–645
20. Bharathi S, Chervenak A (2009) Data staging strategies and their impact on the execution of scientific workflows. In: Proceedings of the second international workshop on data-aware distributed computing, ACM, p 41–50
21. Bianchini R, Rajamony R (2004) Power and energy management for server systems. Computer 37(11):68–76
22. Bilgaiyan S, Sagnika S, Das M (2014) Workflow scheduling in cloud computing environment using cat swarm optimization. In: Proceedings of 2014 IEEE international advance computing conference (IACC), IEEE, pp 680–685
23. Bittencourt LF, Madeira ER (2008) A performance-oriented adaptive scheduler for dependent tasks on grids. Concurr Comput Pract Exp 20(9):1029–1049
24. Bittencourt LF, Madeira ERM (2011) Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds. J Internet Serv Appl 2(3):207–227
25. Blythe J, Jain S, Deelman E, Gil Y, Vahi K, Mandal A, Kennedy K (2005) Task scheduling strategies for workflow-based applications in grids. In: Proceedings of cluster computing and the grid, CCGrid 2005, vol 2, IEEE International Symposium on 2005, pp 759–767
26. Bölöni L, Marinescu DC (2002) Robust scheduling of metaprograms. J Sched 5(5):395–412
27. Bozdağ D, Özgüner F, Catalyurek UV (2009) Compaction of schedules and a two-stage approach for duplication-based dag scheduling. IEEE Trans Parallel Distrib Syst 20(6):857–871
28. Braun TD, Siegel HJ, Beck N, Boloni LL, Muthucumaru M, Reuther AI, Robertson JP, Theys MD, Yao B, Hensgen D, Freund RF (1999) A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In: Proceedings of 8'th heterogeneous computing workshop, IEEE, pp 15–29
29. Brevik J, Nurmi D, Wolski R (2006) Predicting bounds on queuing delay for batch-scheduled parallel machines. In: Proceedings of the eleventh ACM SIGPLAN symposium on principles and practice of parallel programming, ACM, pp 110–118
30. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25(6):599–616
31. Byun EK, Kee YS, Deelman E, Vahi K, Mehta G, Kim JS (2008) Estimating resource needs for time-constrained workflows. In: Proceedings of IEEE fourth international conference on eScience, IEEE, pp 31–38
32. Byun EK, Kee YS, Kim JS, Deelman E, Maeng S (2011) Bts: resource capacity estimate for time-targeted science workflows. J Parallel Distrib Comput 71(6):848–862
33. Byun EK, Kee YS, Kim JS, Maeng S (2011) Cost optimized provisioning of elastic resources for application workflows. Future Gener Comput Syst 27(8):1011–1026
34. Canon LC, Jeannot E (2010) Evaluation and optimization of the robustness of dag schedules in heterogeneous environments. IEEE Trans Parallel Distrib Syst 21(4):532–546
35. Canon LC, Jeannot E, Sakellariou R, Zheng W (2008) Comparative evaluation of the robustness of dag scheduling heuristics. In: Proceedings of grid computing, Springer, New York, pp 73–84
36. Cao F, Zhu MM, Wu CQ (2014) Energy-efficient resource management for scientific workflows in clouds. In: Proceedings of services (SERVICES), IEEE World Congress on 2014, IEEE, pp 402–409
37. Chen H, Shirazi B, Marquis J (1993) Performance evaluation of a novel scheduling method: linear clustering with task duplication. In: Proceedings of the 2nd international conference on parallel and distributed systems
38. Chen WN, Zhang J (2009) An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. IEEE Trans Syst Man Cybern Part C Appl Rev 39(1):29–43

39. Cherkasova L, Gardner R (2005) Measuring cpu overhead for i/o processing in the xen virtual machine monitor. In: Proceedings of USENIX annual technical conference, general track, vol 50
40. Cherkasova L, Phaal P (2002) Session-based admission control: a mechanism for peak load management of commercial web sites. IEEE Trans Comput 51(6):669–685
41. Chu SC, Tsai PW (2007) Computational intelligence based on the behavior of cats. Int J Innov Comput Inf Control 3(1):163–173
42. Chung YC, Ranka S (1992) Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors. In: Proceedings of supercomputing '92. IEEE, pp 512–521
43. Coffman EG (1976) Computer and job shop scheduling theory. Wiley, New York
44. Coffman EG, Graham RL (1972) Optimal scheduling for two-processor systems. Acta Informatica 1(3):200–213
45. Colin J, Chretienne P (1991) C.p.m. scheduling with small computation delays and task duplication. In: Proceedings of operations research, pp 680–684
46. Cordasco G, Malewicz G, Rosenberg AL (2010) Extending ic-scheduling via the sweep algorithm. J Parallel Distrib Comput 70(3):201–211
47. Corrêa RC, Ferreira A, Rebreyend P (1996) Integrating list heuristics into genetic algorithms for multiprocessor scheduling. In: Proceedings of eighth symposium on parallel and distributed processing, IEEE, pp 462–469
48. Darbha S, Agrawal DP (1998) Optimal scheduling algorithm for distributed-memory machines. IEEE Trans Parallel Distrib Syst 9(1):87–95
49. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
50. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans Evolut Comput 6(2):182–197
51. Dejun J, Pierre G, Chi CH (2010) Ec2 performance analysis for resource provisioning of service-oriented applications. In: Proceedings of ICSOC/ServiceWave 2009 workshops service-oriented computing, Springer, New York, pp 197–207
52. Dodin B (1985) Bounding the project completion time distribution in pert networks. Op Res 33(4):862–881
53. Dogan A, Ozguner F (2002) Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):308–323
54. Dogan A, Özgüner F (2005) Biobjective scheduling algorithms for execution time? Reliability trade-off in heterogeneous computing systems. Comput J 48:300–314. doi:10.1093/comjnl/bxh086
55. Dongarra JJ, Jeannot E, Saule E, Shi Z (2007) Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In: Proceedings of the nineteenth annual ACM symposium on parallel algorithms and architectures, ACM, pp 280–288
56. Dubois D, Fargier H, Fortemps P (2003) Fuzzy scheduling: modelling flexible constraints vs. coping with incomplete knowledge. Eur J Op Res 147(2):231–252
57. EI-Rewini H, Lewis TG (1990) Scheduling parallel program tasks onto arbitrary target machines. J Parallel Distrib Comput 1(9):138–153
58. Elastichosts. http://www.elastichosts.com/
59. Fard HM, Prodan R, Barrionuevo JJD, Fahringer T (2012) A multi-objective approach for workflow scheduling in heterogeneous environments. In: Proceedings of the 2012 12th IEEE/ACM international symposium on cluster, cloud and grid computing (ccgrid 2012), IEEE Computer Society, pp 300–309
60. Fard HM, Prodan R, Fahringer T (2013) A truthful dynamic workflow scheduling mechanism for commercial multicloud environments. IEEE Trans Parallel Distrib Syst 24(6):1203–1212
61. Fayad C, Garibaldi JM, Ouelhadj D (2007) Fuzzy grid scheduling using tabu search. In: Proceedings of IEEE international fuzzy systems conference, IEEE, pp 1–6
62. Fayad C, Petrovic S (2005) A fuzzy genetic algorithm for real-world job shop scheduling. In: Proceedings of innovations in applied artificial intelligence, Springer, New York, pp 524–533
63. Fishburn PC (1985) Interval graphs and interval orders. Discret Math 55(2):135–149
64. Foster I, Zhao Y, Raicu I, Lu S (2008) Cloud computing and grid computing 360-degree compared. In: Proceedings of grid computing environments workshop 2008, GCE'08, IEEE, pp 1–10
65. Gao PX, Curtis AR, Wong B, Keshav S (2012) It's not easy being green. ACM SIGCOMM Comput Commun Rev 42(4):211–222

66. Gogrid cloud hosting. http://www.gogrid.com/
67. Garg SK, Buyya R, Siegel HJ (2010) Time and cost trade-off management for scheduling parallel applications on utility grids. Future Gener Comput Syst 26(8):1344–1355
68. Ge R, Feng X, Cameron KW (2005) Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. In: Proceedings of the 2005 ACM/IEEE conference on supercomputing, IEEE Computer Society, p 34
69. Gerasoulis A, Yang T (1993) On the granularity and clustering of directed acyclic task graphs. IEEE Trans Parallel Distrib Syst 4(6):686–701
70. Hakem M, Butelle F (2007) Reliability and scheduling on systems subject to failures. In: Proceedings of international conference on parallel processing, IEEE, pp 38–38
71. Hönig U, Schiffmann W (2006) A meta-algorithm for scheduling multiple dags in homogeneous system environments. In: Proceedings of the eighteenth IASTED international conference on parallel and distributed computing and systems (PDCS06)
72. Hou ESH, Ansari N, Ren H (1994) A genetic algorithm for multiprocessor scheduling. IEEE Trans Parallel Distrib Syst 5(2):113–120
73. Hu TC (1961) Parallel sequencing and assembly line problems. Op Res 9(6):841–848
74. Hwang JJ, Chow YC, Lee FDACY (1989) Scheduling precedence graphs in systems with interprocessor communication times. SIAM J Comput 18(2):244–257
75. Hwang S, Kesselman C (2003) Grid workflow: a flexible failure handling framework for the grid. In: Proceeedings of high performance distributed computing, 12th IEEE international symposium on 2003, IEEE, pp 126–137
76. Iosup A, Ostermann S, Yigitbasi MN, Prodan R, Fahringer T, Epema DH (2011) Performance analysis of cloud computing services for many-tasks scientific computing. IEEE Trans Parallel Distrib Syst 22(6):931–945
77. Isard M, Budiu M, Yu Y, Birrell A, Fetterly D (2007) Dryad: distributed data-parallel programs from sequential building blocks. In: Proceedings of ACM SIGOPS operating systems review, vol 41. ACM, pp 59–72
78. Iverson MA, Özgüner F (1999) Hierarchical, competitive scheduling of multiple dags in a dynamic heterogeneous environment. Distrib Syst Eng 6(3):112
79. Iyer R, Tewari V, Kant K (2001) Overload control mechanisms for web servers. In: Proceedings of performance and QoS of next generation networking, Springer, New York, pp 225–244
80. Juve G, Chervenak A, Deelman E, Bharathi S, Mehta G, Vahi K (2013) Characterizing and profiling scientific workflows. Future Gener Comput Syst 29(3):682–692
81. Juve G, Deelman E, Vahi K, Mehta G, Berriman B, Berman BP, Maechling P (2010) Data sharing options for scientific workflows on amazon ec2. In: Proceedings of the 2010 ACM/IEEE international conference for high performance computing, networking, storage and analysis, IEEE Computer Society, pp 1–9
82. Kerbyson DJ, Alme HJ, Hoisie A, Petrini F, Wasserman HJ, Gittings M (2001) Predictive performance and scalability modeling of a large-scale application. In: Proceedings of the 2001 ACM/IEEE conference on supercomputing (CDROM), ACM, pp 37–37
83. Kim KH, Beloglazov A, Buyya R (2011) Power-aware provisioning of virtual machines for real-time cloud services. Concurr Comput Pract Exp 23(13):1491–1505
84. Kim KH, Buyya R, Kim J (2007) Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters. CCGRID 7:541–548
85. Kim SJ, Browne JC (1991) A general approach to mapping of parallel computation upon multiprocessor architectures. In: Proceedings of the 1991 ACM/IEEE conference on supercomputing '91, ACM/IEEE, pp 633–642
86. Knowles J, Corne D (1999) The pareto archived evolution strategy: a new baseline algorithm for pareto multiobjective optimisation. In: Proceedings of the 1999 congress on evolutionary computation, IEEE, vol 1
87. Kruatrachue B, Lewis T (1988) Grain determination for parallel processing systems. In: Proceedings of the twenty-first annual Hawaii international conference on software track, IEEE, pp 119–128
88. Kruatrachue B, Lewis T (1988) Grain size determination for parallel processing. IEEE Softw 5(1):23–32
89. Kwok YK, Ahmad I (1996) Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. IEEE Trans Parallel Distrib Syst 7(5):506–521

90. Kwok YK, Ahmad I (1997) Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. J Parallel Distrib Comput 47(1):58–77
91. Kwok YK, Ahmad I (1998) Benchmarking the task graph scheduling algorithms. In: Proceedings of the international parallel processing symposium, IEEE, pp 531–537
92. Kwok YK, Ahmad I (1999) Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput Surv (CSUR) 31(4):406–471
93. Lee K, Paton NW, Sakellariou R, Deelman E, Fernandes AA, Mehta G (2009) Adaptive workflow processing and execution in pegasus. Concurr Comput Pract Exp 21(16):1965–1981
94. Lee YC, Subrata R, Zomaya AY (2009) On the performance of a dual-objective optimization model for workflow applications on grid platforms. Parallel Distrib Syst IEEE Trans 20(9):1273–1284
95. Lee YC, Zomaya AY (2011) Energy conscious scheduling for distributed computing systems under different operating conditions. Parallel Distrib Syst IEEE Trans 22(8):1374–1381
96. Leon J, Fisher AL, Steenkiste P (1993) Fail-safe pvm: a portable package for distributed programming with transparent recovery, technical report, DTIC Document
97. Li G, Chen D, Wang D, Zhang D (2003) Task clustering and scheduling to multiprocessors with duplication. In: Proceedings of the parallel and distributed processing symposium, IEEE
98. Li J, Su S, Cheng X, Huang Q, Zhang Z (2011) Cost-conscious scheduling for large graph processing in the cloud. In: Proceedings of 13th international conference on high performance computing and communications (HPCC), IEEE, pp 808–813
99. Lin X, Wu CQ (2013) On scientific workflow scheduling in clouds under budget constraint. In: Proceedings of 42nd international conference in parallel processing (ICPP), IEEE, pp 90–99
100. Liou JC, Palis MA (1996) An efficient task clustering heuristic for scheduling dags on multiprocessors. In: Proceedings of multiprocessors, workshop on resource management, symposium of parallel and distributed processing, pp 152–156
101. López MM, Heymann E, Senar MA (2006) Analysis of dynamic heuristics for workflow scheduling on grid systems. In: Proceedings of the fifth international symposium on parallel and distributed computing, IEEE, pp 199–207
102. Ludäscher B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones M, Lee EA, Tao J, Zhao Y (2006) Scientific workflow management and the kepler system. Concurr Comput Pract Exp 18(10):1039–1065
103. Ludwig A, Möhring RH, Stork F (2001) A computational study on bounding the makespan distribution in stochastic project networks. Ann Op Res 102(1–4):49–64
104. Malawski M, Juve G, Deelman E, Nabrzyski J (2012) Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In: Proceedings of the international conference on high performance computing, networking, storage and analysis, IEEE Computer Society Press, p 22
105. Malewicz G, Foster I, Rosenberg AL, Wilde M (2007) A tool for prioritizing dagman jobs and its evaluation. J Grid Comput 5(2):197–212
106. Mao M, Humphrey M (2011) Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: Proceedings of 2011 international conference for high performance computing, networking, storage and analysis, ACM, p 49
107. Mao M, Humphrey M (2013) Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In: Proceedings of 27th international symposium on parallel and distributed processing (IPDPS), IEEE, pp 67–78
108. Mao M, Li J, Humphrey M (2010) Cloud auto-scaling with deadline and budget constraints. In: Proceedings of grid computing (GRID), 11th IEEE/ACM international conference on 2010, IEEE, pp 41–48
109. Mell P, Grance T (2009) The nist definition of cloud computing. Natl Inst Stand Technol 53(6):50
110. Meng X, Pappas V, Zhang L (2010) Improving the scalability of data center networks with traffic-aware virtual machine placement. In: Proceedings of INFOCOM 2010, IEEE, pp 1–9
111. Menon A, Santos JR, Turner Y, Janakiraman GJ, Zwaenepoel W (2005) Diagnosing performance overheads in the xen virtual machine environment. In: Proceedings of the 1st ACM/USENIX international conference on virtual execution environments, ACM, pp 13–23
112. Mezmaz M, Melab N, Kessaci Y, Lee YC, Talbi EG, Zomaya AY, Tuyttens D (2011) A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. J Parallel Distrib Comput 71(11):1497–1508
113. Michael RG, Johnson DS (1979) Computers and intractability, a guide to the theory of np-completeness. WH Freeman Co., San Francisco

114. Mishra R, Rastogi N, Zhu D, Mossé D, Melhem R (2003) Energy aware scheduling for distributed real-time systems. In: Proceedings of parallel and distributed processing symposium 2003, IEEE, p 9
115. Negoita C, Zadeh L, Zimmermann H (1978) Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets Syst 1:3–28
116. Nudd GR, Kerbyson DJ, Papaefstathiou E, Perry SC, Harper JS, Wilcox DV (2000) Pacea toolset for the performance prediction of parallel and distributed systems. Int J High Perform Comput Appl 14(3):228–251
117. Nurmi D, Brevik J, Wolski R (2008) Qbets: queue bounds estimation from time series. In: Proceedings of job scheduling strategies for parallel processing, Springer, New York, pp 76–101
118. Nurmi D, Mandal A, Brevik J, Koelbel C, Wolski R, Kennedy K (2006) Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. In: Proceedings of the 2006 ACM/IEEE conference on supercomputing, ACM, p 119
119. Ostermann S, Prodan R (2012) Impact of variable priced cloud resources on scientific workflow scheduling. In: Proceedings of Euro-Par 2012 parallel processing, Springer, New York, pp 350–362
120. Palis MA, Liou JC, Wei DS (1996) Task clustering and scheduling for distributed memory parallel architectures. IEEE Trans Parallel Distrib Syst 7(1):46–55
121. Pandey S, Wu L, Guru SM, Buyya R (2010) A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: Proceedings of 2010 24th IEEE international conference on advanced information networking and applications (AINA), IEEE, pp 400–407
122. Papadimitriou CH, Yannakakis, M (1988) Towards an architecture-independent analysis of parallel algorithms. In: Proceedings of the twentieth annual ACM symposium on theory of computing, STOC '88, ACM, New York. doi:10.1145/62212.62262
123. Park GL, Shirazi B, Marquis J (1997) Dfrn: a new approach for duplication based scheduling for distributed memory multiprocessor systems. In: Proceedings of 11th international parallel processing symposium, pp 157–166
124. Park SM, Humphrey M (2008) Data throttling for data-intensive workflows. In: Proceedings of IEEE international symposium on parallel and distributed processing, IEEE, pp 1–11
125. Plank JS, Beck M, Kingsley G, Li K (1994) Libckpt: transparent checkpointing under unix. Computer Science Department
126. Pllana S, Fahringer T (2005) Performance prophet: a performance modeling and prediction tool for parallel and distributed programs. In: Proceedings of international conference workshops on parallel processing, IEEE, pp 509–516
127. Poola D, Garg SK, Buyya R, Yang Y, Ramamohanarao K (2014) Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In: Proceedings of the 28th IEEE international conference on advanced information networking and applications (AINA-2014), pp 1–8
128. Poola D, Ramamohanarao K, Buyya R (2014) Fault-tolerant workflow scheduling using spot instances on clouds. Procedia Comput Sci 29:523–533
129. Prodan R, Wieczorek M (2010) Bi-criteria scheduling of scientific grid workflows. IEEE Trans Autom Sci Eng 7(2):364–376
130. Pruhs K, van Stee R, Uthaisombut P (2008) Speed scaling of tasks with precedence constraints. Theory Comput Syst 43(1):67–80
131. Radulescu A, van Gemund AJ, Lin HX (1999) Llb: a fast and effective scheduling algorithm for distributed-memory systems. In: Proceedings of the international parallel processing symposium, IEEE, pp 525–530
132. Ramakrishnan A, Singh G, Zhao H, Deelman E, Sakellariou R, Vahi K, Blackburn K, Meyers D, Samidi M (2007) Scheduling data-intensive workflows onto storage-constrained distributed resources. In: Proceeedings of seventh IEEE international symposium on cluster computing and the grid, IEEE, pp 401–409
133. Ramakrishnan L, Koelbel C, Kee YS, Wolski R, Nurmi D, Gannon D, Obertelli G, Yarkhan A, Mandal A, Huang TM et al (2009) Vgrads: enabling e-science workflows on grids and clouds with fault tolerance. In: Proceedings of the conference on high performance computing networking, storage and analysis, IEEE, pp 1–12
134. Rivoire S, Shah MA, Ranganathan P, Kozyrakis C (2007) Joulesort: a balanced energy-efficiency benchmark. In: Proceedings of the 2007 ACM SIGMOD international conference on management of data, ACM, pp 365–376

135. Rodriguez MA, Buyya R (2014) Deadline based resource provisioning and scheduling algorithmfor scientific workflows on clouds. IEEE Trans Cloud Comput (to be published)
136. Rountree B, Lowenthal D, Funk S, Freeh V, de Supinski B, Schulz M (2007) Bounding energy consumption in large-scale mpi programs, in the ACM. In: Proceedings of IEEE conference on supercomputing, Nov 2007, vol 1
137. Sakellariou R, Zhao H (2004) A low-cost rescheduling policy for efficient mapping of workflows on grid systems. Sci Program 12(4):253–262
138. Sakellariou R, Zhao H, Tsiakkouri E, Dikaiakos MD (2007) Scheduling workflows with budget constraints. In: Proceedings of integrated research in GRID computing. Springer, New York, pp 189–202
139. Sarkar V (1987) Partitioning and scheduling parallel programs for execution on multiprocessors. PhD thesis, Stanford, CA, USA. UMI order no GAX87-23080
140. Schad J, Dittrich J, Quiané-Ruiz JA (2010) Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proc VLDB Endow 3(1–2):460–471
141. Sharifi M, Shahrivari S, Salimi H (2013) Pasta: a power-aware solution to scheduling of precedence-constrained tasks on heterogeneous computing resources. Computing 95(1):67–88
142. Shestak V, Smith J, Siegel HJ, Maciejewski AA (2006) A stochastic approach to measuring the robustness of resource allocations in distributed systems. In: Proceedings of international conference on parallel processing, IEEE, pp 459–470
143. Shi Z, Jeannot E, Dongarra JJ (2006) Robust task scheduling in non-deterministic heterogeneous computing systems. In: Proceedings of cluster computing, IEEE international conference on 2006, IEEE, pp 1–10
144. Sih GC, Lee EA (1993) A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans Parallel Distrib Syst 4(2):175–187
145. Smith W, Foster I, Taylor V (1998) Predicting application run times using historical information. In: Proceedings of job scheduling strategies for parallel processing. Springer, New York, pp 122–142
146. Sonmez O, Yigitbasi N, Abrishami S, Iosup A, Epema D (2010) Performance analysis of dynamic workflow scheduling in multicluster grids. In: Proceedings of the 19th ACM international symposium on high performance distributed computing, ACM, pp 49–60
147. Stavrinides GL, Karatza HD (2011) Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques. Simul Modell Pract Theory 19(1):540–552
148. Stellner G (1996) Cocheck: checkpointing and process migration for mpi. In: Proceedings of the 10th international parallel processing symposium, IEEE, pp 526–531
149. Szepieniec T, Bubak M (2008) Investigation of the dag eligible jobs maximization algorithm in a grid. In: Proceedings of the 2008 9th IEEE/ACM international conference on grid computing, IEEE Computer Society, pp 340–345
150. Talukder A, Kirley M, Buyya R (2009) Multiobjective differential evolution for scheduling workflow applications on global grids. Concurr Comput Pract Exp 21(13):1742–1756
151. Taverna. http://www.taverna.org.uk/
152. Thain D, Tannenbaum T, Livny M (2005) Distributed computing in practice: the condor experience. Concurr Comput Pract Exp 17(2–4):323–356
153. Topcuoglu H, Hariri S, Wu MY (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Syst 13(3):260–274
154. Tsai YL, Huang KC, Chang HY, Ko J, Wang ET, Hsu CH (2012) Scheduling multiple scientific and engineering workflows through task clustering and best-fit allocation. In: Proceedings of IEEE eighth world congress in services, pp 1–8
155. Ullman JD (1975) Np-complete scheduling problems. J Comput Syst Sci 10(3):384–393
156. Venkatachalam V, Franz M (2005) Power reduction techniques for microprocessor systems. ACM Comput Surv (CSUR) 37(3):195–237
157. Vira C, Haimes YY (1983) Multiobjective decision making: theory and methodology. In: System science and engineering, vol 8. North-Holland
158. Wang H, Jing Q, Chen R, He B, Qian Z, Zhou L (2010) Distributed systems meet economics: pricing in the cloud. In: Proceedings of HotCloud'10. USENIX
159. Wang L, Siegel HJ, Roychowdhury VP, Maciejewski AA (1997) Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. IEEE Trans Parallel Distrib Syst 47(1):8–22

160. Wang M, Ramamohanarao K, Chen J (2012) Dependency-based risk evaluation for robust workflow scheduling. In: Proceedings of IEEE 26th international parallel and distributed processing symposium workshops and PhD forum (IPDPSW), IEEE, pp 2328–2335
161. Wang M, Zhu L, Chen J (2012) Risk-aware checkpoint selection in cloud-based scientific workflow. In: Proceedings of second international conference on cloud and green computing (CGC), IEEE, pp 137–144
162. Wang W, Niu D, Li B, Liang B (2013) Dynamic cloud resource reservation via cloud brokerage. In: Proceedings of 33rd international conference on distributed computing systems (ICDCS), IEEE, pp 400–409
163. Wieczorek M, Hoheisel A, Prodan R (2009) Towards a general model of the multi-criteria workflow scheduling on the grid. Future Gener Comput Syst 25(3):237–256
164. Wieczorek M, Podlipnig S, Prodan R, Fahringer T (2008) Bi-criteria scheduling of scientific workflows for the grid. In: Proceedings of 8th IEEE international symposium on cluster computing and the grid, IEEE, pp 9–16
165. Wieczorek M, Prodan R, Fahringer T (2005) Scheduling of scientific workflows in the askalon grid environment. ACM SIGMOD Record 34(3):56–62
166. Wu AS, Yu H, Jin S, Lin KC, Schiavone G (2004) An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Trans Parallel Distrib Syst 15(9):824–834
167. Wu CM, Chang RS, Chan HY (2014) A green energy-efficient scheduling algorithm using the dvfs technique for cloud datacenters. Future Gener Comput Syst 37:141–147
168. Wu MY, Gajski DD (1990) Hypertool: a programming aid for message-passing systems. IEEE Trans Parallel Distrib Syst 1(3):330–343
169. Wu Z, Ni Z, Gu L, Liu X (2010) A revised discrete particle swarm optimization for cloud workflow scheduling. In: Proceedings of 2010 international conference on computational intelligence and security (CIS), IEEE, pp 184–188
170. Xiao P, Hu ZG, Zhang YP (2013) An energy-aware heuristic scheduling for data-intensive workflows in virtualized datacenters. J Comput Sci Technol 28(6):948–961
171. Yang T, Gerasoulis A (1991) A fast static scheduling algorithm for dags on an unbounded number of processors. In: Proceedings of the 1991 ACM/IEEE conference on supercomputing '91, ACM/IEEE, pp 633–642
172. Yang T, Gerasoulis A (1994) Dsc: scheduling parallel tasks on an unbounded number of processors. IEEE Trans Parallel Distrib Syst 5(9):951–967
173. Yassa S, Chelouah R, Kadima H, Granado B (2013) Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. Sci World J 2013. doi:10.1155/2013/350934
174. Yi S, Kondo D, Andrzejak A (2010) Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In: Proceedings of IEEE 3rd international conference on cloud computing (CLOUD), IEEE, pp 236–243
175. Young L, McGough S, Newhouse S, Darlington J (2003) Scheduling architecture and algorithms within the iceni grid middleware. In: Proceedings of UK e-science all hands meeting, Citeseer, pp 5–12
176. Yu J, Buyya R (2006) A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. In: Proceedings of workshop on workflows in support of large-scale science, IEEE
177. Yu J, Buyya R (2006) Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci Program 14(3):217–230
178. Yu J, Buyya R, Ramamohanarao K (2008) Workflow scheduling algorithms for grid computing. In: Proceedings of Metaheuristics for scheduling in distributed computing environments. Springer, New York, pp 173–214
179. Yu J, Buyya R, Tham CK (2005) Cost-based scheduling of scientific workflow applications on utility grids. In: Proceedings of first international conference on e-science and grid computing, IEEE, pp 8
180. Yu J, Kirley M, Buyya R (2007) Multi-objective planning for workflow execution on grids. In: Proceedings of the 8th IEEE/ACM international conference on grid computing, IEEE Computer Society, pp 10–17
181. Yu J, Ramamohanarao K, Buyya R Deadline/budget-based scheduling of workflows on utility grids. In: Proceedings of market-oriented grid and utility computing, pp 427–450
182. Yu Z, Shi W (2007) An adaptive rescheduling strategy for grid workflow applications. In: Proceedings of IEEE international parallel and distributed processing symposium, IEEE, pp 1–8

183. Yu Z, Shi W (2008) A planner-guided scheduling strategy for multiple workflow applications. In: Proceedings of international conference on parallel processing-workshops, IEEE, pp 1–8
184. Yu ZF, Shi WS (2010) Queue waiting time aware dynamic workflow scheduling in multicluster environments. J Comput Sci Technol 25(4):864–873
185. Yuan Y, Li X, Wang Q, Zhang Y (2008) Bottom level based heuristic for workflow scheduling in grids. Chin J Comput Chin 31(2):282
186. Yuan Y, Li X, Wang Q, Zhu X (2009) Deadline division-based heuristic for cost optimization in workflow scheduling. Inf Sci 179(15):2562–2575
187. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on hot topics in cloud computing, p 10
188. Zeng L, Veeravalli B, Li X (2012) Scalestar: Budget conscious scheduling precedence-constrained many-task workflow applications in cloud. In: Proceedings of IEEE 26th international conference on advanced information networking and applications (AINA), IEEE, pp 534–541
189. Zhang Y, Koelbel C, Cooper K (2009) Hybrid re-scheduling mechanisms for workflow applications on multi-cluster grid. In: 9th IEEE/ACM international symposium on cluster computing and the grid, IEEE, pp 116–123
190. Zhao H, Sakellariou R (2006) Scheduling multiple dags onto heterogeneous systems. In: Proceedings of 20th international parallel and distributed processing symposium, IEEE, p 14
191. Zheng W, Sakellariou R (2012) Budget-deadline constrained workflow planning for admission control in market-oriented environments. In: Proceedings of economics of grids, clouds, systems, and services, Springer, New York, pp 105–119
192. Zheng W, Sakellariou R (2013) Budget-deadline constrained workflow planning for admission control. J Grid Comput 11(4):633–651
193. Zheng W, Sakellariou R (2013) Stochastic dag scheduling using a monte carlo approach. J Parallel Distrib Comput 73(12):1673–1689
194. Zhou AC, He B (2014) Transformation-based monetary cost optimizations for workflows in the cloud. IEEE Trans Cloud Comput 2(1):85–98
195. Zhou AC, He B, Liu C (2013) Monetary cost optimizations for hosting workflow-as-a-service in iaas clouds. arXiv:1306.6410
196. Zhu D, Melhem R, Childers BR (2003) Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems. Parallel Distrib Syst IEEE Trans 14(7):686–700
197. Zhu Q, Zhu J, Agrawal G (2010) Power-aware consolidation of scientific workflows in virtualized environments. In: Proceedings of the 2010 ACM/IEEE international conference for high performance computing, networking, storage and analysis, IEEE Computer Society, pp 1–12
198. Zitzler E, Laumanns M, Thiele L, Zitzler E, Zitzler E, Thiele L, Thiele L (2001) Spea 2: improving the strength pareto evolutionary algorithm
199. Zomaya AY, Ward C, Macey B (1999) Genetic scheduling for parallel processor systems: comparative studies and performance issues. IEEE Trans Parallel Distrib Syst 10(8):795–812