Postprint: Cloud Workflow Scheduling Strategy under DAG Partitioning Model

Authors: Xue Fan

Date: 2018-10-11T00:00:00+00:00

Abstract

Workflow scheduling is an economically effective optimization approach in the field of engineering management. To optimize the economic cost and execution efficiency of cloud workflow scheduling, we propose a workflow scheduling algorithm PBWS based on Directed Acyclic Graph (DAG) partitioning. Aiming at the simultaneous optimization of workflow scheduling efficiency and cost, the algorithm divides the scheduling solution process into three stages: workflow DAG structure partitioning, partition structure adjustment, and resource allocation. The workflow DAG structure partitioning stage solves the initial task partition graph while ensuring execution order dependencies among tasks; the partition structure adjustment stage aims to reduce the makespan by reassigning tasks among different partitions; the resource allocation stage aims to select the most cost-effective task-to-resource mapping relationships to ensure minimal total idle time of resources. Simulation experiments were conducted on the algorithm using five scientific workflow DAG models. Results demonstrate that the PBWS algorithm significantly reduces workflow execution cost at the expense of only a small increase in makespan, achieving simultaneous optimization of scheduling efficiency and scheduling cost, and its overall performance is superior to similar algorithms.

Full Text

Preamble

Cloud Workflow Scheduling Strategy Based on DAG Partition Model

Xue Fan
(Innovation & Entrepreneurship College, Huanghuai University, Zhumadian, Henan 463000, China)

Abstract: Workflow scheduling is an economical and effective optimization method in engineering management. To optimize both the economic cost and

execution efficiency of cloud workflow scheduling, this paper proposes a workflow scheduling algorithm based on Directed Acyclic Graph (DAG) partitioning called PBWS. With the goal of simultaneously optimizing workflow scheduling efficiency and cost, the algorithm divides the scheduling solution process into three stages: workflow DAG structure partitioning, partition structure adjustment, and resource allocation. The DAG structure partitioning stage obtains an initial task partition graph while ensuring execution order dependencies between tasks. The partition structure adjustment stage reassigns tasks across different partitions to reduce execution makespan. The resource allocation stage aims to select the most cost-effective task-to-resource mapping relationships to minimize total resource idle time. Simulation experiments were conducted using five scientific workflow DAG models. The results demonstrate that the PBWS algorithm significantly reduces workflow execution cost with only a small overhead in makespan, achieving simultaneous optimization of scheduling efficiency and cost, with overall performance superior to comparable algorithms.

Keywords: cloud computing; scientific workflow; scheduling optimization; DAG partition; execution makespan

0 Introduction

Workflow scheduling is fundamentally an NP-hard problem, with most research focusing on single-objective optimization functions. For instance, authors in [2–4] designed scheduling algorithms to find the lowest-cost scheduling solution that meets predefined workflow execution deadlines. In contrast, algorithms in [5–8] seek the fastest scheduling solution under predefined budget constraints. Meanwhile, [9–11] focus on reducing execution time without considering cost constraints. Although these methods introduce budget or deadline constraints, they remain essentially single-objective optimizations that fail to adequately address workflow scheduling characteristics in cloud environments.

Scientific workflow applications typically exhibit explosive and complex large-scale data growth, demanding higher computational capabilities. Most application tasks in these domains manifest as workflows [1], with structures shown in five conventional scientific domain workflows in [Figure 1: see original paper]. In a workflow, each task functions as part of the whole—either as a transmitter of initial data or as an intermediary relay for data—forming inter-task sequential dependencies to collaboratively complete an application task.

This paper simultaneously considers the minimization of both execution makespan and cost, designing a DAG partition-based workflow scheduling algorithm. The algorithm divides the workflow DAG structure into multiple groups (partitions) containing multiple tasks, thereby achieving optimized matching between partitions and cloud resource dynamic capabilities. Through this three-stage scheduling optimization, the algorithm ultimately achieves a balance between workflow scheduling efficiency and cost.

1 System Model

A workflow is represented as a Directed Acyclic Graph (DAG) $G = \langle V, E \rangle$, where V is the task set and E is the edge set. Each edge connects two tasks, representing sequential constraints between them. Given a resource set $R = \{R_1, R_2, \ldots, R_m\}$, the resource sets are configured based on Amazon EC2 instance types, including m4.xlarge, c4.2xlarge, and r3.4xlarge. Each resource set $R_i \in R$ consists of homogeneous resources with varying numbers of processor cores, processing capabilities, memory, and storage space. The set R is sorted based on resource capability: if $j > i$, then resources in set R_i have lower capability than those in R_j.

A child task v_i can only begin execution after receiving data from its parent tasks. Among all parent tasks, the one that sends data latest is called the Most Influential Parent (MIP) task. Let $c(i, j)$ denote the communication cost between tasks v_i and v_j. The workflow completion time is defined as the execution makespan.

For each task $v_i \in V$, the Earliest Start Time (EST) and Earliest Finish Time (EFT) are calculated as follows:

$$EST(v_i) = \begin{cases} 0 & \text{if } v_i \text{ is an entry task} \\ \max_{v_p \in MIP(v_i)} \{EFT(v_p) + c(p, i)\} & \text{otherwise} \end{cases}$$

$$EFT(v_i) = EST(v_i) + w_i$$

where w_i is the execution time of task v_i. The Actual Start Time (AST) and Actual Finish Time (AFT) differ from EST and EFT when the actual completion time of a task scheduled on the same resource is later than $EST(v_i)$.

The workflow makespan is primarily affected by the execution time of tasks on the critical path, which has the highest cost (communication and computation) and ends at the exit task. The computation cost (execution time w_i) of task $v_i \in V$ depends on the resource capability where the task is scheduled—higher resource capability results in shorter execution time. Let w_i^j denote the computation cost (time), where i is the index of task $v_i \in V$ and j is the index of resource set $R_j \in R$.

To simplify calculations, a constant factor $cv_i > 0$ is introduced for each resource set $R_i \in R$. Given task v_i, each resource set constant $cv_i (R_i \in R)$ represents the approximate relative execution time of the task. For example, if $cv_3 = 4$, then the task execution time $w_i^3 = (1/4)w_i^1$. Let t_i denote the cost per hour of utilizing resources in resource set $R_i \in R$, which depends on resource capability (if $i < j$, then $t_i < t_j$).

The workflow scheduling objective is to assign tasks of the given workflow to resource set S in R while minimizing both the total resource utilization cost and the task execution makespan.

2.1 Algorithm Overview

The PBWS algorithm aims to partition workflow tasks into task groups to generate effective resource-to-task mappings, with the partitioning goal of simplifying the resource allocation process. Based on data dependencies between partition regions, the algorithm determines the resource capability allocated to each partition and ensures minimization of workflow makespan and cost. The PBWS algorithm consists of three steps: (a) DAG partitioning step; (b) partition adjustment step; and (c) resource allocation step.

The DAG partitioning step divides the given workflow DAG structure such that the resulting DAG partitions simplify resource allocation while ensuring task dependencies. Each partition is subsequently treated as a single node. In the partition adjustment step, certain tasks are reassigned and re-partitioned into different partitions to further reduce makespan. The resource allocation step aims to achieve the most cost-effective mapping between tasks and resources, ensuring minimal total resource idle time.

2.2 DAG Partitioning Step

Algorithm 1 presents the pseudocode for the DAG partitioning step. The algorithm first determines the number of tasks allocated to each partition while considering the execution time of the workflow' s critical path (CP). Since the sum of task execution times on the critical path (i.e., the critical path length, denoted as $l(CP)$) represents the lower bound (optimal solution) of makespan, DAG partitioning must ensure that the total execution time of tasks belonging to the same partition is less than or equal to $l(CP)$ (the while loop in step 8). Because tasks on the critical path are scheduled on the same resource, the critical path length includes only execution time, whereas task scheduling length in any other partition (denoted as P' in Algorithm 1) includes both execution and communication time.

The length (cost) of partition P is defined as:

$$l(P) = \sum_{v_i \in P} w_i^1 + \sum_{e(v_i, v_j) \in P} c(i, j)$$

where the first term on the right side represents the sum of computation costs for tasks in partition P on the lowest-cost resource w_1, and the second term represents the sum of communication costs between tasks in partition P.

The DAG partitioning step begins execution from the lowest-level tasks in the workflow structure (such as different levels in the workflow shown in Figure 1(b)). This step adds tasks to the currently constructed partition until no new tasks can be added without violating the $l(CP)$ constraint (step 12). Once all tasks at the same level are allocated to partitions, tasks at the next level (higher level) continue partitioning until all tasks are assigned to different partitions.

Algorithm 1: DAG Partitioning Process

```
Procedure Partitioning(G, l(CP))
Input: G=<V,E>, l(CP)
Output: P=<p1,p2,...> (collection of partitions)
1.  N ← order tasks based on level
2.  while N is not empty do
3.    create new partition P'
4.    while l(P') < l(CP) do
5.      add first task in N to P'
6.      remove the task from N
7.    end while
8.    if l(P') > l(CP) then
9.      undo last step
10.   end if
11.   add P' to P
12. end while
13. return P
14. end procedure
```

2.3 Partition Adjustment Step

To ensure that the task partitions generated in the DAG partitioning step have optimal granularity for the final resource allocation process, tasks from different partitions need to be reassigned and adjusted. Finding optimal partitions primarily involves considering the number of tasks in each partition and their allocated resource types.

Since task partitioning and inter-task data dependencies are interrelated, reassigning tasks across different partitions requires recalculating EST and EFT values. For each task, two values are computed: *sest* and *pest*. The *sest* value of task v_i represents the average EST of v_i' s child tasks that are not in the same partition as v_i. The *pest* value of task v_i represents the average EST of tasks in the same partition as v_i.

Based on these two values, for each task $v_i \in V$, we can determine whether to adjust v_i' s partition by reassigning it to another partition (with the smallest $EST(v_i)$) to reduce makespan. This can be achieved by finding tasks where $sest \leq pest$. This partition adjustment significantly impacts both makespan

and cost performance, as it reduces the waiting time of child tasks caused by the execution of parent tasks in different partitions. If a reassigned task has child tasks within its partition, those child tasks must also be considered for reassignment to the new partition. A task is only considered for reassignment if its execution time is less than the average execution time of tasks in the partition where it resides. This criterion ensures that tasks execute as early as possible in the reassigned partition.

The result of the partition adjustment step is represented as DAG $G' = \langle V', E' \rangle$, where V' represents the set of partitions and E' represents the set of data dependencies between partitions. Any two partitions $v'_j \in V'$ are connected by one or more directed edges, with each directed edge $e'(v'_j) \in E'$ indicating a task $v_p \in v'_j$ and its parent task in v'_i. The edge also shows the minimum EST of tasks belonging to v'_j, denoted as $eest$.

2.4 Resource Allocation Step

Algorithm 2 presents the pseudocode for the resource allocation step. This step consists of two phases: resource set identification and resource allocation. In the resource set identification phase, we determine the resource set type $(i, R_i \in R)$ allocated to each partition. In the resource allocation phase, each task v_j is assigned to a resource $r \in R_i$ such that r belongs to the same resource set type as the partition containing v_j.

The resource set identification phase aims to find the resource set type allocated to each partition to minimize data dependencies between partitions. The resource allocation phase aims to determine the fastest scheduling scheme based on the slack parameter β and the identified resource set types.

Algorithm 2: Resource Allocation Process

```
Procedure Reallocation((G', P,  , R))
Input: G'=<V',E'>, P partitions,  , R=R1,...,Rm resources
1.  C ← group P based on successor partition ID
     each group Ci has one or more partitions
2.  order C based on the dependency between the tasks
     Ci, Cj (i<j), Ci depends on Cj data
3.  for each Ci C do
4.    for each Pj Ci do
5.      h ← (et + cPi) / eest
6.      call AssignResource(h, Pj, R)
7.      for each vi Pj do
8.        call UpdateESTEFT(vi, h, G')
9.      end for
10.   end for
11. end for
```

```
12. for each vi V do
13.   AST(vi) ←   × (UBi - LBi) + LBi
14.   AFT(vi) ← AST(vi) + wi
15. end for
16. end procedure
```

Detailed Algorithm Explanation: The resource set identification phase begins by dividing partitions into different partition groups (step 4). A partition group includes a group leader P_i and partitions connected to P_i. A partition with more than one dependency relationship can belong to multiple groups. For each group (the for loop in step 8), starting from the group containing the exit partition as a leader group (the exit group includes the exit task), partitions belonging to this group (except the leader) begin identifying their resource set types (the for loop in step 9).

In each group, each partition' s resource set type ($i, R_i \in R$) depends on the *eest* value of the group leader. To determine the resource set type, each partition must identify its execution start time, i.e., EST, denoted as *et*. Additionally, each partition must calculate its tasks' computation time cP_i. After obtaining these two values, divide each partition' s $et+cP_i$ by its group leader' s *eest* value (step 10). This division result indicates the required computation time reduction for that partition to decrease makespan. This reduction can be achieved by assigning to the partition the resource set type with the cv_i value closest to this division result (step 11).

If any partition contains only a single partition, that partition is assigned to the lowest resource set type R_1. Once a partition' s resource set type is determined, the EST and EFT for each task in that partition can be recalculated. The EST and EFT of child tasks in that partition are also recalculated (steps 12-14). After a partition identifies its resource set type, if it is a group leader, it informs its group members/partitions to begin the resource set identification phase. When a partition belongs to multiple groups, it is assigned to the highest resource set type.

After all partitions determine their resource set types, the resource allocation phase begins assigning resources to tasks based on each partition' s identified resource set type (steps 17-20). In this phase, based on the slack parameter β value, each task' s AST and AFT are determined. Parameter β ($0 \leq \beta \leq 1$) controls the number of resources used for task execution. Increasing this parameter reduces the number of resources used, favoring cost optimization over makespan. Decreasing the parameter favors makespan optimization.

This phase begins by calculating the upper bound UB and lower bound LB for each task' s AST. These two values represent the allowable delay during each task' s execution. Each task' s AST is set according to parameter β (step 18). If $\beta = 1$, the task AST is set to UB, and the algorithm uses the minimum number of resources. If $\beta = 0$, the task AST is set to LB, and the algorithm uses the maximum number of resources. The LB for task v_i is $(MIP_i) + c(MIP_i, i)$,

representing the earliest time to execute task v_i without resource constraints.

The UB for task v_i is $AFT(v_j) + c_{j,i}$, where v_j is an intermediate task inserted into the partition and $c_{j,i}$ is the communication time between tasks v_i and v_j. In each iteration, the unassigned task with the lowest AST is considered for allocation. This task is assigned to a resource in the resource set type matching its partition that can ensure timely execution. If no such resource exists, a new instance in the resource set type is activated.

2.5 Example Illustration

[Figure 2: see original paper] illustrates the PBWS algorithm concept. For simplicity, computation and communication costs are not shown in the example. Figure 2(b) shows the result of the DAG partitioning step, where each task is assigned to a partition. Figure 2(c) shows the result of the partition adjustment step, where task t_9 meets the condition of having the smallest EST and is reassigned to partition p_2.

In the resource allocation step shown in Figure 2(d), each partition (except the root partition) marks its $et + eP_i$ as H and its $eest$ value. In this example, $H_1 = 40$, $eest_1 = 20$, $H_2 = 20$, $eest_2 = 20$. Since p_3 is a child partition of both p_1 and p_2, these two partitions identify their required resource set types through division of their H values by $eest$ values, indicating that p_1' s resource set type is $2(R_2)$ and p_2' s resource set type is $1(R_1)$. This means all tasks in p_1 are assigned to resources in R_2, and all tasks in p_2 are assigned to resources in R_1.

Once partitions in the same group (with the same descendants) have identified their resource set types, the execution order of tasks in each partition and each partition' s $eest$ can be obtained. The leader partition then informs its group members to begin the resource set identification phase. In this example, p_1 is the parent partition of p_2. However, p_1 remains unchanged since the identified resource set type $2(R_2)$ is higher than $1(R_1)$. Task execution order depends on the level, such as in p_1 where the order is t_{10}, t_7, t_8, t_4. Each task' s AST in a partition depends on the β value. If a single resource is assigned to each partition ($\beta = 1$), the AST values for tasks in p_1 are $0, 10, 20, 30$ respectively. Assuming each task' s computation cost in p_1 is 10, if $\beta = 0$, the AST values become $0, 10, 10, 20$, with t_7 and t_8 assigned to different resources.

2.6 Algorithm Time Complexity Analysis

PBWS algorithm first calculates EST and EFT for each task with time complexity $O(|V|)$. Then it calculates the critical path with time complexity $O(|V|)$. The DAG partitioning step converges with at most $O(|V|^2)$ time. In the parti-

tion adjustment step, calculating the average EFT of child tasks for each task has time complexity $O(|V|^2)$. Additionally, task reassignment has time complexity $O(|V|(|P|-1))$. Therefore, the overall time complexity of the partition adjustment step is $O(|V|^2+|V|(|P|-1))$.

In the resource allocation step, sorting partitions has time complexity $O(|P|^2)$. Grouping partitions has time complexity $O(|P|)$, and the resource allocation process has time complexity $O(|P||R|)$. Therefore, the overall time complexity of the resource allocation step is $O(|P|^2+|P|(1+|R|))$.

In the worst case, the number of DAG partitions equals the number of tasks (one task per partition). Therefore, the worst-case time complexity of the PBWS algorithm is $O(|V|^2+|P|^2+(|V||P|)+|P||R|)=O(|V|^2+|V||R|)$.

---

3 Simulation Experiments

This chapter analyzes the performance of the PBWS algorithm through simulation experiments. The IC-PCP [4] and HEFT [11] algorithms are selected as baseline algorithms. The cloud computing environment simulation tool is CloudSim, and input workflows are generated by the Pegasus workflow generator (https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator). Experiments are conducted using five real-world scientific workflows shown in Figure 1: CyberShake, Epigenomics, LIGO, Montage, and SIPHT, with 1,000 tasks configured for each workflow type. The resource sets are $R=\{R_1,R_2,R_3\}$, where the fastest resource R_3 costs three times the slowest resource R_1. Inter-resource bandwidth is set to 1 Gbps, and the number of available resources is assumed unlimited. The β values are set to $\{0,0.5,1\}$.

3.1 Performance Metrics

The experiments primarily observe algorithm performance in terms of makespan and normalized cost. Cost includes task execution cost, resource initialization cost, and data transfer cost. Obviously, increasing resource quantity in a scheduling scheme significantly impacts total cost. Therefore, normalized cost N_c is adopted to compare this metric performance, which considers both task execution cost and the number of resources used. For each schedule, the obtained cost is divided by the scheduling cost on the lowest-cost resource, representing the cost ratio compared to the lowest-cost schedule. This value is then multiplied by the number of resources used in the obtained schedule:

$$N_c=\frac{c}{sc}\times r$$

where c is the number of tasks executed on resources, r is the number of resources used in the schedule, and sc is the cost of scheduling tasks to the lowest-cost resource.

3.2 Experimental Results

[Figure 3: see original paper] and [Figure 4: see original paper] compare the performance of the three algorithms in terms of makespan and normalized cost. The results show that PBWS consistently outperforms other algorithms in cost, while HEFT outperforms others in makespan, indicating that PBWS sacrifices some execution efficiency to reduce cost. Specifically, we discuss the results based on the structural characteristics of the five workflow types.

a) CyberShake Workflow: For CyberShake, when β values are 0.5 and 0 (i.e., the algorithm favors makespan minimization), PBWS and HEFT have very similar makespan (Figure 3(a)), with only a small increase in execution cost (Figure 4(a)). Specifically, since reducing β leads to more resource usage and the CyberShake workflow (Figure 1(d)) has no bottleneck tasks, PBWS constructs multiple partitions that can execute in parallel to reduce makespan. Although IC-PCP achieves the overall smallest makespan, this better performance comes at the cost of more resource usage and higher cost (Figure 4(a)). Additionally, HEFT has higher cost when $\beta = 0$ because it uses higher-cost resources to achieve better makespan performance.

b) Epigenomics and LIGO Workflows: For these workflows, PBWS achieves makespan close to IC-PCP (Figures 3(b) and 3(c)), but with lower cost than IC-PCP (Figures 4(b) and 4(c)). For normalized cost, since IC-PCP achieves the same makespan independent of β, we expect the resulting schedules to also have the same cost. For PBWS, the obtained makespan depends on both β value and workflow structure. Reducing β can increase the number of resources used and affect the final makespan. However, this effect only occurs when the reduction leads to increased resource allocation to each partition. The structural consistency of these two workflows allows partitions to execute with better continuity while satisfying data dependencies. Therefore, changing β values does not significantly impact PBWS performance. In experiments, HEFT uses the same number of resources as PBWS. For PBWS, changing β brings smaller variations in resource usage. Thus, reducing β has minimal impact on HEFT' s makespan. Overall, HEFT, as a greedy algorithm, can achieve smaller makespan but at higher cost compared to PBWS.

c) Montage and SIPHT Workflows: For these workflows, results show that IC-PCP has the worst normalized cost performance in most cases, which is related to workflow structure. In Montage, tasks in the final level form a pipeline structure, which has a significant negative impact on IC-PCP, primarily because the length of the longest critical path has a more important influence on algorithm efficiency in this case. In SIPHT, a similar effect occurs due to the left-side path emergence.

provides specific performance values for the two metrics. The results again prove that PBWS produces schedules with lower cost compared to other algorithms, achieved by sacrificing only a small makespan.

Table 1: Scheduling Performance

Workflow Type	Normalized Cost
CyberShake	IC-PCP: 0.85
Epigenomics	IC-PCP: 0.92
LIGO	IC-PCP: 0.88
Montage	IC-PCP: 1.15
SIPHT	IC-PCP: 1.25

Overall, PBWS demonstrates optimized comprehensive performance across different scientific workflow structures. Compared to similar algorithms, PBWS achieves balanced optimization of execution efficiency and cost by dividing the optimization objective into two stages: the first stage improves execution efficiency and reduces makespan through initial task DAG partitioning and task reassignment after partition structure adjustment; the second stage optimizes execution cost during resource mapping, resulting in less resource idle time and more efficient resource utilization. Although the algorithm cannot achieve optimal performance on individual metrics, it provides better balance in comprehensive multi-metric performance, which better aligns with actual cloud scheduling environments and workflow optimization requirements.

---

4 Conclusion

To achieve workflow scheduling optimization in cloud computing, this paper proposes a DAG partition-based workflow scheduling algorithm. Through a three-stage workflow scheduling pattern comprising task partitioning, partition adjustment, and resource allocation, the algorithm achieves balanced scheduling of task execution makespan and cost. Experimental results demonstrate that the proposed algorithm can achieve lower execution cost at the expense of only a small execution efficiency overhead, with performance superior to similar algorithms.

Future research will focus on: incorporating energy consumption of cloud resources executing tasks into the optimization objective, providing workflow execution deadlines and user cost constraints, establishing multi-objective optimization scheduling models under multi-constraint conditions, and designing corresponding DAG partition algorithms under the new scheduling model to achieve multi-objective scheduling.

References

[1] Pietri I, Malawski M, Juve G, et al. Energy-constrained provisioning for

scientific workflow ensembles [C]// Proc of the 3rd International Conference on Cloud and Green Computing. 2015: 34-41.

[2] Med M, Master H. Auto-scaling to minimize cost and meet application deadlines in cloud workflows [C]// Proc of High Performance Computing, Networking, Storage and Analysis. [S. l.]: IEEE Press, 2014: 1-12.

[3] Rodriguez M A, Buyya R. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds [J]. IEEE Trans on Cloud Computing, 2014, 2(2): 222-235.

[4] Abrishami S, Naghibzadeh M, Epema D H J. Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds [J]. Future Generation Computer Systems, 2013, 29(1): 158-169.

[5] Wu C Q, Lin Xiangyu, Yu Dantong, et al. End-to-end delay minimization for scientific workflows in clouds under budget constraint [J]. IEEE Trans on Cloud Computing, 2015, 3(2): 169-181.

[6] Zeng Lingfang, Veeravalli B, Li Xiaorong. ScaleStar: budget conscious scheduling precedence-constrained many-task workflow applications in cloud [C]// Proc of IEEE 26th International Conference on Advanced Information Networking and Applications. Piscataway, NJ: IEEE Press, 2012, 11(1): 534-541.

[7] Arabnejad H, Barbosa J G. A budget constrained scheduling algorithm for workflow applications [J]. Journal of Grid Computing, 2014, 12(4): 665-678.

[8] Zheng Wei, Sakellariou R. Budget-deadline constrained workflow planning for admission control [J]. Journal of Grid Computing, 2013, 11(4): 633-651.

[9] Lee Y C, Zomaya A. Stretch out and compact: workflow scheduling with resource abundance [C]// Proc of IEEE//ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE, 2013: 219-226.

[10] Sakellariou R, Zhao Hao. A hybrid heuristic for DAG scheduling on heterogeneous systems [C]// Parallel and Distributed Processing Symposium. Proceedings. International. [S. l.]: IEEE Press, 2014: 111-114.

[11] Topcuoglu H, Hariri S, Wu Minyou. Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Trans on Parallel & Distributed Systems, 2012, 13(3): 260-274.

Note: Figure translations are in progress. See original paper for figures.