Scheduling Workflows with Limited Budget to Cloud Server and Serverless Resources

Jinquan Zhang, Xiaoping Li, *Senior Member, IEEE,* Long Chen, and Rubén Ruiz

Abstract

Serverless functions (SFs) and on-demand virtual machines (VMs) are common cloud resources for scientific workflow applications, which are widespread in many fields (e.g., biology, astronomy, and agricultural sciences). SFs are paid by actual running time with higher unit costs and higher resource utilization, whereas VMs are paid by billing time units with lower unit costs and lower resource utilization. Generally, each application is executed on a limited budget. Since there are a large number of task topological orders in each application, it is challenging to schedule a budget-limited workflow application to SFs and VMs in a hybrid cloud environment. In this paper, we study the cloud workflow scheduling problem with limited budgets to minimize makespan in a hybridization of SFs and on-demand VMs for which the BCWS (Budget Constrained Workflow Scheduling) algorithm is proposed. Methods are developed to determine the task execution order, rent cloud resources and map tasks to resources respectively. Along with initial schedule construction and schedule improvement policies, these methods are repeatedly conducted in BCWS. The proposed algorithm is evaluated by comparing it to existing algorithms for similar problems over a comprehensive set of workflow instances. Experimental results show that the proposed algorithm significantly reduces the makespan with a hybrid configuration of VMs and SFs compared to the only server case or the only serverless configuration one and outperforms the compared algorithms which are the best existing ones for similar problems.

Index Terms

Scientific workflow, Scheduling, Budget, Makespan, Serverless function, Hybrid configuration, Critical path.

Jinquan Zhang, Xiaoping Li and Long Chen work at the School of Computer Science and Engineering, Southeast University, Nanjing, China, 211189; and also at the Key Laboratory of Computer Network and Information Integration, Southeast University, Ministry of Education, 211189, Nanjing, China.

Email:{zhangjq, xpli, chen_long}@seu.edu.cn.

Rubén Ruiz works at the Departmento de Estadística e Investigación Operativa Aplicadas y Calidad, Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain.

E-mail:rruiz@eio.upv.es.

Manuscript received ; revised ..

Scheduling Workflows with Limited Budget to Cloud Server and Serverless Resources

I. Introduction

Scientific workflow applications are widespread in analyzing the data of scientific experiments, each of which consists of hundreds or even thousands of precedence-constrained tasks [1]. These tasks are commonly executed by rented cloud resources whereas customers have a limited budget for each application. Makespan is one of the most concerned objectives which is significantly effected by task execution orders, available resources, and task-resource mapping policies as well as the budget constraint [2]–[6]. Therefore, it is important to minimize the makespan of budget-limited scientific workflow applications.

In cloud computing, VMs (Virtual machines) and SFs (serverless functions) are common provisioned resources. SFs are charged by their actual running time with a higher unit cost and higher resource-utilization whereas VMs are paid by BTUs (billing time units, e.g. hours for the on-demand manner) with lower unit costs and lower resource-utilization. Because of the complex task precedence constraints (tasks could be executed in either parallel or serial), the resource requirements for executing a scientific workflow application are uneven. It is desirable to use a hybridization of SFs and VMs for such uneven resource requirements. An illustrative example is shown in Figure 1. Suppose that the speeds of SFs and VMs are the same. In terms of Amazon AWS EC2[1] and AWS Fargate[2], the unit costs of VMs and SFs are 0.102\$/h and 0.19748\$/h, respectively. For the computation-intensive workflow containing 3 tasks as shown in Figure 1(a), the computation time of tasks $t_1 \sim t_3$ is 10, 20 and 30 minutes, respectively. Data transmission time among tasks is ignored. The budget B is 0.170\$. By enumerating all possible schedules, the minimal makespans for the server, the serverless, and the hybrid configurations are 60, 40, and 40 minutes, respectively. Corresponding rental costs are 0.102\$, 0.197\$, and 0.168\$, respectively, which implies no feasible solution is obtained in the serverless configuration. Results shows that the optimal schedule is obtained by the hybrid configuration. Therefore, it is necessary to schedule budget limited workflow applications using a hybridization of VMs and SFs rather than by a single server configuration or a single serverless configuration.

[1]https://aws.amazon.com/cn/ec2/pricing/

[2]https://aws.amazon.com/cn/fargate/pricing/

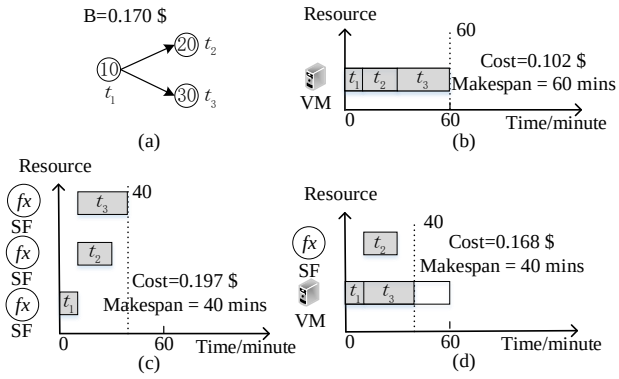

Fig. 1. The optimal schedules of the workflow instance (a) in different configurations: (b) server configuration, (c) serverless configuration, and (d) the hybrid configuration.

In this paper, we consider the scientific workflow scheduling problem (WSP) with a limited budget to minimize the makespan in a hybridization of on-demand VMs and SFs. SFs are assumed to be heterogeneous while VMs are homogeneous for simplifying the complex problem under study. A central shared storage is adopted for data exchange among different tasks. It is much difficult to schedule complex precedence-constrained tasks of a budget-limited workflow to heterogeneous SFs and on-demand VMs. Since WSPs with only server configurations are NP-hard [7], it is natural that the considered problem with a hybridization configuration is also NP-hard with the following challenges: i) For each application, the complicated precedence constraints among tasks result in a large number of topological orders. Different topological orders could lead to different makespans. It is challenging to determine the most appropriate task topological order for makespan minimization. ii) SFs and VMs have different executing and pricing manners. Each application has a limited budget and uneven resource requirements. It is great challenging to rent the most appropriate number of SFs and VMs for a budget-limited workflow application with makespan minimization. iii) The heterogeneity of SFs and different executing manners of SFs and VMs result in multiple executing durations for the same task. There are a large number of allocation candidates for all the tasks of an application to the rented SFs and VMs. It is challenging to determine the optimal task-resource allocation scheme to minimize makespan.

For the challenges mentioned above, we propose a heuristic scheduling algorithm BCWS (Budget Constrained Workflow Scheduling) for the problem under study. The main contributions

are as follows:

1) Considering a hybrid configuration of VMs and SFs as well as the limited budget constraint of a workflow application, the problem under study is mathematically modeled.
2) A resource configuration strategy is developed for renting an appropriate number of SFs and VMs for the uneven resource requirements by workflow tasks.
3) Since makespan is closely related to the critical path of the workflow application, we present the BCWS algorithm which shortens the length of the critical path by iteratively changing its task-resource allocation.

The rest of this paper is organized as follows: Section II reviews the related works of the problem under study. The problem is described and formulated in Section III. Section IV introduces the proposed BCWS algorithm. Experimental results are shown in Section V, followed by conclusions and future works in Section VI.

II. Related Works

WSPs with various objectives, constraints, and resource configurations have been extensively studied. A brief review of related works on makespan minimization of workflow applications with a server configuration is presented in Section II-A. Details on WSPs with a serverless configuration and with a hybrid configuration are further discussed in Section II-B and Section II-C, respectively, which are closely related to our work.

A. Makespan Minimization of Workflow Applications with Servers

Makespan minimization seems to be the most concerning objective in the existing works on WSPs. Classic methods have been proposed for makespan minimization of workflows in multiprocessor systems, grids, and clusters, e.g., DLS (Dynamic Level Scheduling) [8], MH (Scheduling Heuristic) [9], and HEFT (Heterogeneous Earliest Finish Time) [10], HBMCT (Hybrid Balanced Minimum Completion Time) [11]. Kwok et al. [23] summarised and compared 27 static scheduling algorithms for minimizing the makespan of workflows in multiprocessor systems with limited resources. However, these algorithms are not suitable for cloud computing, where the resources are provided geographically and elastically. Since cloud workflow tasks are executed on resources rented from cloud providers, a trade-off between the rental cost and the makespan is necessary. Wu et al. [5] proposed a CG (Critical-Greedy) algorithm for WSPs with limited budgets. An initial schedule is generated by allocating the budget to each task fairly. The

TABLE I

COMPARISON OF RELATED LITERATURE.

Related works	Objective				Constraint			Resource		BTU	
	Makespan	Cost	Success rate	Resource consumption	Deadline	Budget	Limited VMs	VM	SF	Fine-grained	Coarse-grained
[8]–[11]	✓						✓	✓			
[5], [12]	✓					✓		✓			
[4], [13]	✓					✓		✓		✓	
[6], [14], [15]	✓					✓		✓			✓
[16]		✓			✓				✓		
[17]	✓	✓			✓	✓			✓		
[18]			✓		✓	✓			✓		
[19], [20]	✓			✓					✓		
[21]		✓					✓	✓	✓		
[22]	✓	✓					✓	✓	✓		
This work	✓					✓		✓	✓		✓

remaining budget is utilized to speed up the execution of the tasks with the maximal ratio of the time difference to the cost difference on the critical paths. By transferring the budget constraints of workflow applications into budget constraints of tasks, Chen et al. [12] developed a method to generate feasible schedules in which each task is allocated to the processor with the earliest finish time. In both [5] and [12], the rental cost is calculated without considering the BTU of VMs. In terms of the relative size of BTUs to the average execution time of tasks, VM BTUs can be divided into fine-grained BTUs (such as one minute) [4], [13] and coarse-grained BTUs (such as one hour) [6], [14], [15]. A task might need several BTUs for execution in a fine-grained BTU manner, whereas several tasks might be executed within only one BTU in the case of the coarse-grained manner. Arabnejad et al. [4] investigated WSPs constrained by deadlines and budgets with fine-grained BTUs. The budget and deadline constraints of workflow applications are divided into levels to promote the success rate of finding feasible schedules. Faragardi et al. [6] developed the GRP-HEFT (Greedy Resource Provisioning and modified HEFT) algorithm for minimizing makespan of budget-limited workflow applications with coarse-grained BTUs. The number and type of VM instances are specified and adjusted iteratively. Tasks are allocated to available VM instances with a modified HEFT algorithm. Zhou et al. [24] studied a multi-objective WSP to minimize the rental cost and makespan without specific billing models and

BTU manners.

B. Workflow Scheduling with Serverless Configurations

SF selection and SF deployment are two main factors that significantly influence the makespan of workflows in serverless computing. The heterogeneity of SFs with different prices, CPU cores, and memory sizes always results in different tradeoffs [16]–[18], [25], [26] between the rental cost and makespan by selecting different SFs for tasks of a workflow application. Wen et al. [16] analyzed the main factors (memory size, inter and intra-function parallelism) impacting makespan and cost of a workflow. A workflow scheduling algorithm was proposed based on critical paths to reduce the rental cost while satisfying the deadline constraint. Jarachanthan et al. [17] also explored the factors impacting on SFs which further influence the cost and makespan of a workflow. Two subproblems were considered: i) minimizing makespan with a given budget constraint, ii) minimizing the rental cost with a given deadline constraint. Majewski et al. [18] developed the SMOHEFT (Multi-Objective Heterogeneous Earliest Finish Time for Serverless Architectures) algorithm for maximizing the success rate of executing scientific workflows with budget and deadline constraints in a serverless configuration. A Pareto set is generated by adopting the crowding distance measure [27] to explore different tradeoffs between makespan and the rental cost. However, the above works did not consider the influence of SF deployments on the makespan of workflows. Generally, SFs are deployed on containers. Cold starts of SFs cannot be neglected if most tasks can be finished within several seconds. Though cold starts can be avoided effectively by maintaining containers warm, resource consumption would be incurred. It is important to get a tradeoff between the resource consumption and makespan for workflow applications [19], [20]. Xu et al. [19] mitigated cold starts of SFs by a pre-warm policy and a container pool policy. Since an SF could be invoked many times in a workflow, a pre-warm policy can improve its scheduling performances, e.g., reducing the average latency of workflows [20], reducing the energy consumption by packing workflow tasks to fewer containers as many as possible [20].

C. Workflow Scheduling with Hybrid Configurations

It is natural to execute workflow applications with hybridization of SFs and servers (e.g., private clusters, on-demand VMs) [21], [22]. Jiang et al. [21] implemented a workflow management system in which the available resources include SFs in a public cloud and VMs in

a private cluster. Short tasks are executed by SFs, while long tasks are executed by a local cluster. Roy et al. [22] combined on-demand VMs and SFs to minimize makespan and the rental cost of scientific workflows. However, the number of rented VMs is fixed without meeting the unbalanced resource requirement of workflows. Raza et al. [28] adopted SFs and on-demand VMs to reduce SLA (Service Level Agreement) violations and save rental costs only for scheduling independent tasks rather than workflow scheduling.

The above existing studies are summarized in Table I. Most workflow scheduling problems with budget constraints were studied in the server configuration case. Little attention has been paid to the serverless configuration case. No existing work considers scheduling workflow applications with limited budgets in hybridizing servers and serverless functions. Different from existing studies, we study a budget-limited workflow scheduling problem in the hybridization of SFs and on-demand VMs in this paper. To the best of our knowledge, this problem has not been studied yet, although it is crucial for minimizing the makespan of workflow applications. In addition, the number of rented VMs is dynamically determined by the resource requirement of workflows rather than a fixed number as in [21] and [22].

III. Problem Formulation

For the complex problem under study, we make the following assumptions:

- VMs are homogeneous whereas SFs are heterogeneous, which means only one type of VM and multiple types of SFs can be selected for each task.
- Each task is assigned to only one resource instance, and each resource instance executes only one task at a time.
- The instances of SFs and VMs can be set up and released quickly with 0 time.
- Instructions of each workflow task and the size of intermediate data are known in advance.

Notations to be used are listed in Table II.

A. Problem Description

A workflow application can be described as a directed acyclic graph (DAG) $G = (T, E)$. $T = \{t_1, \ldots, t_N\}$ is the task set, and E is the edge set. Task t_i has w_i instructions. t_1 and t_N are the source task and the sink task which are dummy tasks with 0 instructions. Edge $e_{ij} = (t_i, t_j)$ indicates that task t_j cannot start until the data d_{ij} from t_i has been received. In addition, the workflow application G is constrained by a budget B.

TABLE II

NOTATIONS TO BE USED.

Notation	Definition
G	Workflow application
T	Task set, $T = \{t_1, \ldots, t_N\}$
N	Number of tasks in workflow application G
w_i	Instructions of task t_i
E	Edge set, $E = \{(t_i, t_j) \mid \exists t_i \rightarrow t_j\}$
d_{ij}	Data transferred from t_i to t_j
B	Budget of the considered workflow application
K	Number of SF types
M	Number of rented VMs
f_{ik}	k^{th} function of t_i
p_k	Speed of f_{ik}
c_k	Unit cost of f_{ik}
p_{vm}	Speed of VM
c_{vm}	Unit cost of VM
BTU	Length of VM billing time unit
b	Bandwidth between the COS and resource instances
v_m	m^{th} rented VM
x_{ik}	Whether t_i is executed on f_{ik}
y_{im}	Whether t_i is executed on v_m
$CT(t_i)$	Computation time of t_i
$pr(t_i)$	Predecessors of t_i
$su(t_i)$	Successors of t_i
$In(t_i)$	Data input time of t_i
$IP(t_i)$	Input data of t_i (not intermediate data)
$Out(t_i)$	Data output time of t_i
$ET(t_i)$	Execution time of t_i
$Av(v_m)$	Earliest available time of v_m
$ST(t_i)$	Start time of t_i
$FT(t_i)$	Finish time of t_i
$ST(v_m)$	Start time of v_m
$FT(v_m)$	Finish time of v_m
MS	Makespan of the considered workflow application
C_F	Cost of renting SFs
C_V	Cost of renting VMs

In this paper, the provided computing resources include heterogeneous SFs and homogeneous VMs. K types of SFs with different speeds and unit costs can be selected for each task t_i. f_{ik} is the k^{th} function to be selected of which the processing speed is p_k and the unit cost is c_k. For simplicity, we assume that $p_1 < p_2 < \ldots < p_K$ and $c_1 < c_2 < \ldots < c_K$. Only one type of on-demand VM is considered with speed p_{vm} and the unit cost c_{vm}. VMs are paid by time units BTU. In addition, a shared COS (cloud object storage) is adopted for data exchange among resource instances. The network bandwidth between the COS and any resource instance is b. Each task reads its input data from the COS and writes its output data back. However, if two tasks are deployed on the same VM instance, the intermediate data can be utilized directly, i.e., the communication time is 0.

Suppose that the number of rented VMs is M, v_m is the m^{th} rented VM. The binary variable y_{im} takes 1 if and only if task t_i is executed on VM v_m and takes 0 otherwise. Similarly, the binary variable x_{ik} takes 1 if and only if task t_i is executed on SF f_{ik} and takes 0 otherwise.

$$y_{im} = \begin{cases} 1, \text{ if } t_i \text{ is executed on } v_m, i \in \{1, \ldots, N\}, \\ \quad m \in \{1, \ldots, M\}, \\ 0, \text{ otherwise.} \end{cases} \tag{1}$$

$$x_{ik} = \begin{cases} 1, \text{ if } t_i \text{ is executed on } f_{ik}, i \in \{1, \ldots, N\}, \\ \quad k \in \{1, \ldots, K\}, \\ 0, \text{ otherwise.} \end{cases} \tag{2}$$

For any task t_i, the computation time $CT(t_i)$ can be calculated by

$$CT(t_i) = w_i/p_k \times x_{ik} + w_i/p_{vm} \times y_{im}. \tag{3}$$

The time of reading input data $In(t_i)$ can be calculated by

$$In(t_i) = IP(t_i)/b + \sum_{t_j \in pr(t_i)} d_{ji}/b \prod_{m=1}^{M} (1 - y_{im}y_{jm}) \tag{4}$$

where $IP(t_i)$ is the input data of task t_i stored into the COS before workflow execution, and $pr(t_i)$ is the predecessor set of t_i. $1 - y_{im}y_{jm} = 0$ if and only if $y_{im} = 1$ and $y_{jm} = 1$, i.e., t_i and t_j are allocated to the same VM instance v_m. Suppose that $Output(t_i)$ is the size of total output data of t_i. Since all intermediate data is stored into the shared COS, the time of writing output data back $Out(t_i)$ can be calculated by

$$Out(t_i) = Output(t_i)/b. \tag{5}$$

The total execution time $ET(t_i)$ of task t_i is the sum of computation time and data transmission time, i.e.,

$$ET(t_i) = CT(t_i) + In(t_i) + Out(t_i). \tag{6}$$

Let $Av(v_m)$ be the earliest available time of v_m. The start time $ST(t_i)$ and the finish time $FT(t_i)$ of task t_i satisfy

$$ST(t_i) \geq \max\Big\{ \max_{t_j \in pr(t_i)} \{FT(t_j)\}, \sum_{m=1}^{M} Av(v_m) y_{im} \Big\}, \tag{7}$$

$$FT(t_i) = ST(t_i) + ET(t_i). \tag{8}$$

The objective of the considered problem is to minimize the makespan MS

$$\min \ \ MS = \max_{t_i \in T} \{FT(t_i)\}. \tag{9}$$

As the workflow is constrained by budget B, the following inequality should be satisfied, i.e.,

$$C_F + C_V \leq B. \tag{10}$$

where C_F is the rental cost of SFs, and C_V is the rental cost of VMs.

$$C_F = \sum_{t_i \in T} \sum_{k=1}^{K} x_{ik} c_k \times ET(t_i), \tag{11}$$

$$C_V = \sum_{m=1}^{M} c_{vm} \times BTU \times \lceil \frac{FT(v_m) - ST(v_m)}{BTU} \rceil, \tag{12}$$

$FT(v_m)$ is the finish time of v_m, and $ST(v_m)$ is the start time of v_m. They can be determined by

$$FT(v_m) = \max\{FT(t_i) | t_i \in T, y_{im} = 1\}, \tag{13}$$

$$ST(v_m) = \min\{ST(t_i) | t_i \in T, y_{im} = 1\}. \tag{14}$$

IV. Proposed Algorithm

In this section, a heuristic scheduling algorithm Budget Constrained Workflow Scheduling (BCWS) is proposed for workflow application G with a limited budget B in a hybrid configuration of on-demand VMs and SFs to minimize makespan. Suppose that the minimum rental cost of executing G with only serverless functions is C_{SF}^{min} and that with only VMs is C_{VM}^{min}.

The budget B is reasonable if and only if $B \geq \min\{C_{SF}^{min}, C_{VM}^{min}\}$. However, this condition is too general to grantee to find a feasible solution for the considered problem [6], [14], [15]. In this paper, we just consider the case $B \geq \max\{C_{SF}^{min}, C_{VM}^{min}\}$. C_{SF}^{min} can be calculated by

$$C_{SF}^{min} = \sum_{t_i \in T} \min_{k} \left\{ c_k \times (\frac{w_i}{p_k} + MDTT(t_i)) \right\}, \tag{15}$$

where $MDTT(t_i)$ is the maximal data transmission time of task t_i which is determined by

$$MDTT(t_i) = IP(t_i)/b + \sum_{t_j \in pr(t_i)} d_{ji}/b + Out(t_i). \tag{16}$$

C_{VM}^{min} is calculated by

$$C_{VM}^{min} = \left\lceil \frac{\sum_{t_i \in T} \frac{w_i}{p_{vm}} + IP(t_i)/b + Out(t_i)}{BTU} \right\rceil \times c_{vm}. \tag{17}$$

There are three algorithmic components in BCWS: ISC (Initial Schedule Construction), RR (Resource Replacement) and BR (Budget Reallocation). The framework of BCWS is shown as Algorithm 1. Makespan is determined by the length of the critical path. Different SF or VM allocations to a task result in different processing times which could further change the critical path. The numbers and types of rented SFs and VMs are dynamically adjusted in the three components of BCWS. ISC allocates the cheapest SFs to all the tasks in a workflow application from which an initial critical path is constructed. If the remaining budget $RB = B - C_F - C_V > 0$ (here $C_V = 0$), it is desirable to allocate a faster function to a task on the critical path to reduce the makespan (a new critical path could be generated). ISC iteratively selects and speeds up the execution of a task on the critical path and further updates the critical path unless $RB > 0$ is no longer met. RR tries to replace the rented SFs with one non-slower VM each time to guarantee makespan not increase. Decrease of C_F while increase of C_V increase could lead to an increase of the remaining budget RB since VMs are much cheaper than SFs. The replacement process is conducted if the remaining budget RB increases and repeated until no increase on RB. Since a much bigger RB could be obtained by RR, BR tries to select faster SFs using the similar way as that in ISC.

A. Initial Schedule Construction (ISC)

An initial schedule is constructed by selecting the cheapest and slowest functions for all tasks satisfying the budget constraint with the total rental cost of C_{SF}^{min}. The remaining budget RB is utilized to speed up the execution of the tasks on the critical path. The ISC procedure speeds up

Algorithm 1: Budget Constrained Workflow Scheduling Algorithm (BCWS)

Input: Graph G=(T, E); Budget B

Output: Makespan MS

```
begin
    Initial Schedule Construction (ISC) ;
    Resource Replacement (RR) ;
    Budget Reallocation (BR) ;
    return.
```

one task t_i each time by moving it to a slightly more expensive and faster function (from f_{ik} to $f_{i(k+1)}$), followed by a recalculation of the critical path CP. The above process repeats until no more adjustment is possible, i.e., the remaining budget RB cannot support any SF replacement. In this paper, several strategies are proposed to select the next task to be adjusted.

- **METF (Maximal Execution Time First)**: Since the length of the critical path is the summarization of the execution times of the tasks on it, METF selects the task with maximal execution time, i.e., $t_\star = \arg\max_{t_i \in CP}\{ET(t_i)\}$.
- **MTDCIRF (Maximal Time Decrease to Cost Increase Ratio First)**: Suppose that t_i is a task on the critical path CP. The time decrease $\triangle ET(t_i)$ of adjusting task t_i is

$$\begin{aligned} \triangle ET(t_i) &= ET(t_i, f_{ik}) - ET(t_i, f_{i(k+1)}) \\ &= w_i/p_k - w_i/p_{k+1}, \end{aligned} \tag{18}$$

  where $ET(t_i, f_{ik})$ is the execution time of t_i on f_{ik}. Similarly, the cost increase $\triangle C(t_i)$ of adjusting task t_i is

$$\triangle C(t_i) = C(t_i, f_{i(k+1)}) - C(t_i, f_{ik}), \tag{19}$$

  where $C(t_i, f_{ik})$ is the cost of t_i on f_{ik}. $r_{tc}(t_i)$ is the ratio of the time decrease $\triangle ET(t_i)$ to the cost increase $\triangle C(t_i)$, i.e.,

$$r_{tc}(t_i) = \triangle ET(t_i)/\triangle C(t_i). \tag{20}$$

  The task $t_\star = \arg\max_{t_i \in CP}\{r_{tc}(t_i)\}$ is selected.

- **MCETRF (Maximal Computation to Execution Time Ratio First)** : Suppose that t_i is a task on the critical path CP. The ratio of computation to execution time for task t_i is calculated by

$$r_{ce}(t_i) = CT(t_i)/ET(t_i), \tag{21}$$

A larger $r_{ce}(t_i)$ implies that the execution time of t_i could be more reduced if f_{ik} is replaced by $f_{i(k+1)}$. Therefore, the task $t_\star = \arg\max_{t_i \in CP}\{r_{ce}(t_i)\}$ is selected.

Details of ISC with the MTDCIRF strategy are shown in Algorithm 2.

B. Resource Replacement (RR)

RR tries to replace the rented SFs with non-slower VMs one-by-one to increase the remaining budget RB as much as possible while not increasing makespan, i.e., the replacement process is conducted if it leads to an increase in the remaining budget. The replacement process is repeated until RB cannot be increased.

In each replacement, a task queue Q is constructed by collecting all tasks allocated to SFs with speeds slower than or equal to p_{vm}. Obviously, the task execution times in Q do not increase if the replacements are conducted only on these tasks. Moreover, makespan does not increase without delaying the start times of these tasks during the process of replacing SFs with VMs in terms of Theorem 1. Assume the SF allocated to $\forall t_i \in Q$ is replaced by a rented VM instance v. MS and $MS^{'}$ are the makespans and $ST(t_i)$ and $ST(t_i)^{'}$ are the start times of t_i before and after the replacement.

Theorem 1. *$MS^{'} \leq MS$ if $ST^{'}(t_i) \leq ST(t_i)$.*

Proof. If $\forall t_i \in Q$ is allocated to v, $CT^{'}(t_i) \leq CT(t_i)$ according to Eqn. (3) and $In^{'}(t_i) \leq In(t_i)$ according to Eqn. (4). In addition, $Out^{'}(t_i) = Out(t_i)$. $ET^{'}(t_i) = CT^{'}(t_i) + In^{'}(t_i) + Out^{'}(t_i) \leq CT(t_i) + In(t_i) + Out(t_i) = ET(t_i)$. $FT^{'}(t_i) = ST^{'}(t_i) + ET^{'}(t_i) \leq ST(t_i) + ET(t_i) = FT(t_i)$. If $\forall t_j \notin Q$ or $t_i \in Q$ is not allocated to v, $ET^{'}(t_j) = ET(t_j)$. $ST^{'}(t_j) \leq ST(t_j)$ according to Eqn. (7). $FT^{'}(t_j) = ST^{'}(t_j) + ET^{'}(t_j) \leq ST(t_j) + ET(t_j) \leq FT(t_j)$. Therefore, $MS^{'} = \max_{t_i \in T}\{FT^{'}(t_i)\} \leq \max_{t_i \in T}\{FT(t_i)\} = MS$. □

Algorithm 2: Initial Schedule Construction (ISC)

begin
  $CostLeft \leftarrow B$;
  while $CostLeft > 0$ **do**
    Calculate the critical path CP;
    $CPTask \leftarrow \emptyset$;
    foreach $t_i \in CP$ **do**
      $k \leftarrow$ Current function type of t_i;
      if $k+1 \leq K$ **then**
        $\triangle ET(t_i) \leftarrow$ Calculate time decrease;
        $\triangle C(t_i) \leftarrow$ Calculate cost increase;
        $r_{tc}(t_i) \leftarrow \triangle ET(t_i)/\triangle C(t_i)$;
        if $\triangle C(t_i) \leq CostLeft$ **then**
          $CPTask \leftarrow CPTask \bigcup \{t_i\}$;
    if $CPTask = \emptyset$ **then**
      break;
    else
      $t_\star \leftarrow \arg\max_{t_i \in CPTask} r_{tc}(t_i)$;
      $k \leftarrow$ Current function type of $t_\star$;
      Move $t_\star$ to the function with type $k+1$;
      $CostLeft \leftarrow CostLeft - \triangle C(t_\star)$;
  $RB \leftarrow CostLeft$;
  return.

Since the price of an SF is usually much higher than that of a VM, a new VM instance v is rented to some tasks belonging to Q to maximize the remaining budget RB. The binary variable z_i takes 1 if and only if t_i is allocated to VM v and takes 0 otherwise.

$$z_i = \begin{cases} 1, \text{ if } t_i \text{ is allocated to } v, t_i \in Q, \\ 0, \text{ otherwise.} \end{cases} \tag{22}$$

The objective of the replacement is to maximize the increment of the remaining budget $\triangle RB$,

$$\max \ \triangle RB = C_F + C_V - C_F^{'} - C_V^{'}, \tag{23}$$

where $C_F^{'}$ and $C_V^{'}$ are the rental costs of SFs and VMs after the replacement, respectively. To increase the remaining budget, $\triangle RB$ should be not less than 0, i.e.,

$$\triangle RB \geq 0. \tag{24}$$

According to Theorem 1, for $\forall t_i \in Q$ allocated to v, the start time of t_i after the replacement $ST^{'}(t_i)$ should be no later than that before the replacement $ST(t_i)$ to keep makespan non-increased, i.e.,

$$z_i \cdot \left\{ ST^{'}(t_i) - ST(t_i) \right\} \leq 0. \tag{25}$$

For VM instance v, the start time $ST(v)$, the release time $REL(v)$ and the number of rented BTUs n meet the following conditions:

$$REL(v) = ST(v) + n \cdot BTU, \tag{26}$$

$$z_i \cdot \left[ST^{'}(t_i) - ST(v) \right] \geq 0, \tag{27}$$

$$z_i \cdot \left[ST^{'}(t_i) + ET^{'}(t_i) - REL(v) \right] \leq 0. \tag{28}$$

The task order in Q and the number of rented BTUs are crucial for the performance of the replacement. Three strategies are proposed to determine Q.

- **ESTF (Earliest Start Time First)**: The tasks in queue Q are sorted in a non-decreasing order of their start times before the replacement. If multiple tasks have the same start time, the tie is broken by a random order.
- **MRBIF (Maximal Remaining Budget Increment First)**: The asks in Q are sorted by a non-decreasing order of their start times. If multiple tasks have the same start time, they are sorted by a non-increasing order of their estimated remaining budget increments $ERBI(t_j) = c_k \cdot ET(t_j) - c_{vm} \cdot ET^{'}(t_j)$ if the function f_{jk} allocated to t_j is replaced.
- **EFTF (Earliest Finish Time First)**: The tasks in queue Q are sorted in a non-decreasing order of their finish times before the replacement. Since different tasks commonly have different numbers of instructions, it almost impossible to finish them simultaneously.

To determine the rented number of BTUs, three strategies are developed:

- **MRN (Minimal Renting Number)**: Let $t_\star$ be the first task in Q. The number of rented BTUs n is determined by the execution time of $t_\star$ on v, i.e.,

$$n = \lceil ET^{'}(t_\star)/BTU \rceil. \tag{29}$$

- **FRN (Fixed Renting Number)**: To guarantee the first task in Q can be allocated to v, n should not be too small. Let g is a given number in advance. n is determined by $\max\Big\{\lceil ET^{'}(t_\star)/BTU \rceil, g\Big\}$. Obviously, MRN is a special case of FRN ($g = 1$).
- **URN (Unlimited Renting Number)**: The rented number of BTUs n is assumed to be non-limited ($n = +\infty$) and the release time of VM v is infinite ($REL(v) = +\infty$) correspondingly. The actual number of rented BTUs can be obtained after the replacement is conducted. The URN strategy can be regarded as a special case of FRN ($g = +\infty$).

Once Q and n are determined, the start time of v is determined, i.e., $ST(v) = ST(t_\star)$. Let t_i be the last task which can be allocated to v before t_j. The RR procedure sequentially checks whether the following conditions are true or not for each task t_j in Q:

$$\begin{cases} ST(t_j) \geq max\Big\{ST(v), ST(t_i) + ET^{'}(t_i)\Big\}, \\ ST(t_j) + ET^{'}(t_j) \leq REL(v). \end{cases} \tag{30}$$

If Eqn. (30) is satisfied, t_j is selected, and not otherwise . Eqn. (24) is checked after all the selected tasks are determined. If Eqn. (24) is satisfied, the replacement is performed. Otherwise, no task is allocated to v, v is released, Q is updated by removing its first task, and a new VM is rented for the next replacement. The RR procedure is repeated until $Q = \emptyset$. RR with the ESTF and MRN strategies is formally described in Algorithm 3.

C. Budget Reallocation (BR)

The remaining budget RB is reallocated by BR to speed up the tasks on the critical path CP. Though BR allocates the remaining budget using a strategy similar to ISC, they have the following differences:

- Critical path: The critical path CP in ISC contains only tasks allocated to SFs and is calculated based on task execution time, and the edge set E. However, CP in BR contains tasks allocated to both SFs and VMs, and the tasks in the same VM are also precedence constrained according to their execution orders when calculating CP.

- Rental cost increment: In ISC, the cost increment of C is the same as the selected task t_i. Therefore, t_i can be adjusted if $\triangle C(t_i) \leq RB$. However, the budget constraint may still be violated when adjusting t_i in BR since the number of rented BTUs might be changed.
- Makespan: In ISC, if task t_i is moved from a slower SF to a faster one, the makespan does not increase. However, if task t_i is reallocated to a faster SF from a VM in BR, makespan may increase because the data transmission time of t_i and its successors could increase.

Since tasks in the critical path CP are allocated to two kinds of resources (SFs and VMs), two strategies are proposed for reallocating RB:

- **PTA (Partial Task Adjustment)**: Only tasks allocated to SFs on the critical path CP can be reassigned to faster functions from slower ones.
- **GTA (Global Task Adjustment)**: All tasks on CP can be allocated to faster functions, no matter whether they are currently executed on VMs or SFs. In addition, if task t_i allocated to a faster SF $f_{i(k+1)}$ from a VM, the increment of $C(t_i)$ is calculated by $\triangle C(t_i) = C(t_i, f_{i(k+1)}) - c_{vm} \cdot ET(t_i, v)$.

The budget constraint is checked in the above two strategies when adjusting the selected task t_i. GTA also keeps makespan non-increased. Details of RR with the GTA strategy are described in Algorithm 4.

D. Computational Time Complexity Analysis

Each task in RR is adjusted at most $K - 1$ times. The computational time complexity of calculating the critical path is $O(N^2)$. The computational time complexity of selecting tasks is $O(N)$. Therefore, the overall computational time complexity of ISC is $O(N^3K)$, which is the same as BR. For the RR procedure, the computational time complexity of sorting tasks is $O(N \log(N))$. The computational time complexity of each replacement is $O(N^2)$. Since the replacement process may be conducted at most N times, the computational time complexity of RR is $O(N(N \log(N) + N^2)) = O(N^3)$. Therefore, the computational time complexity of the BCWS algorithm is $O(N^3) + O(N^3K) = O(N^3K)$.

V. EXPERIMENTAL RESULTS

To the best of our knowledge, the considered WSP with budget constraints in the hybridization of on-demand VMs and SFs has not been studied yet. To obtain the best algorithm for the considered problem, all algorithmic component candidates of the BCWS algorithm framework

are calibrated over a large number of random instances. To test the performance of the calibrated BCWS algorithm, several existing methods for similar problems are compared. All concerned algorithms are coded in Python and run on an Intel Core i7-7500U CPU @ 2.70GHZ with 12 GBytes of RAM.

All algorithms are compared by RPD (Relative Percentage Deviation):

$$RPD(\%) = \frac{MS(A) - MS^{\star}}{MS^{\star}} \times 100\% \tag{31}$$

$MS(A)$ is the makespan obtained by algorithm A, while $MS^{\star}$ is the minimal makespan obtained among all compared algorithms on the same workflow instance.

The ANOVA (multi-factor analysis of variance) technique is adopted to analyze results. The three main hypotheses (normality, homoscedasticity, and independence of the residuals) are checked from the residuals of the experiments. All three hypotheses are acceptable from the analysis.

In terms of AWS EC2 and AWS Fargate, we conduct experiments by setting one type of on-demand VM and multiple types of SF. Details of these resources are listed in Table III.

A. Parameter & Component Calibration

There are three components (ISC, RR, and BR) and a parameter (g) in the proposed BCWS algorithm framework which need calibration. In ISC, three candidate strategies (METF, MTD-CIRF, MCETRF) are used for task selection on the critical path. In RR, three candidate strategies (ESTF, MRBIF, EFTF) are adopted to sort tasks in the queue Q. FRN^g denotes the FRN strategy with parameter g. Five strategies (MRN, FRN^2, FRN^3, FRN^4, URN) are proposed to determine the number of rented BTUs. In addition, two strategies (PTA, GTA) are considered in BR to reallocate the remaining budget. Therefore, there are $3 \times 3 \times 5 \times 2 = 90$ combinations for the proposed BCWS algorithm framework.

A large number of workflow instances are randomly generated to calibrate all the combinations with different configurations. The number of tasks N of each workflow application takes values from $\{100, 200, 300, 400\}$. The link density LD is the ratio of the number of edges to the number of tasks which takes value from $\{2, 3, 4, 5\}$. The computation time (measured in seconds) of each task executed with an average SF speed is subject to a uniform distribution of $U(1, 1800)$. The CCR (communication computation ratio) takes values from $\{0.5, 0.67, 1.0, 1.5, 2.0\}$. The budget of the workflow instances is defined by $B = (1 + \alpha) \times \max\{C_{SF}^{min}, C_{VM}^{min}\}$, where α is a budget

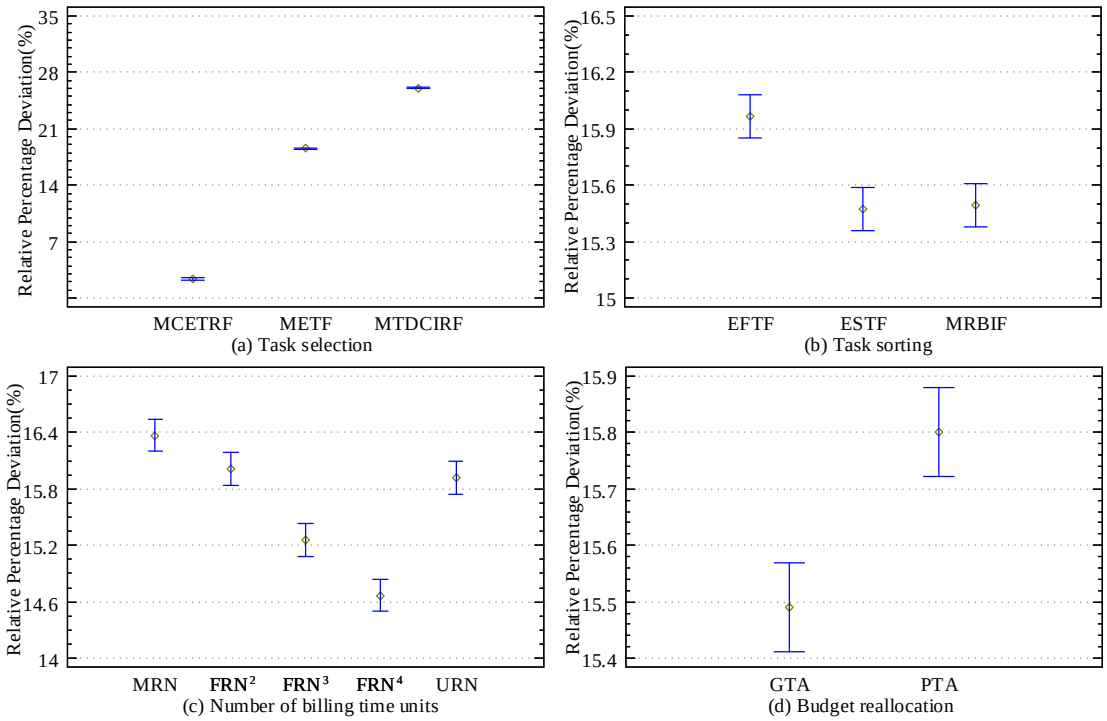

Fig. 2. Mean plots of the different components and parameters of the BCWS algorithm framework with 95.0% Tukey HSD confidence level intervals.

factor with five candidates $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ which demonstrates how tight the budget is. Three instances are generated for each combination of N, LD, CCR, and α. In addition, there are four types of virtual machines. Therefore, $4 \times 4 \times 5 \times 5 \times 3 \times 4 = 4800$ instances are generated in total. The total number of tests conducted is $4800 \times 90 = 432000$. All the resulting p-values are less than 0.05, meaning that all the studied factors significantly affect the RPD response variable at the 95.0% confidence level.

The mean plots of the components and parameters with 95.0% Tukey honest significant difference (HSD) intervals are shown in Figure 2. Figure 2(a) shows that MCETRF statistically outperforms the other two strategies, MTDCIRF and MCETRF. MCETRF has the lowest RPD. It can be observed from Figure 2(b) that ESTF and MRBIF perform similarly with almost the same RPDs though ESTF is a little better than MRBIF. Both of them statistically outperform the EFTF strategy. In Figure 2(c), the RPD of FRN^g firstly decreases with the increase of parameter g and increases with the increase of g afterwards since the MRN and URN strategies can be regarded as special cases of the FRN strategy (FRN^1 and FRN$^\infty$). The lowest RPD is obtained

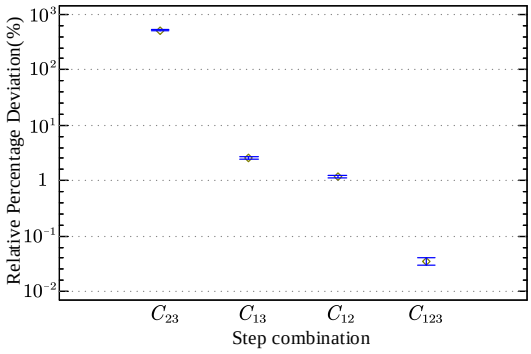

Fig. 3. Mean plots of different step combinations with 95.0% Tukey HSD confidence level intervals: C_{12} (ISC, RR), C_{13} (ISC, BR), C_{23} (RR, BR), C_{123} (ISC, RR, BR).

when $g = 4$. For the two remaining budget reallocating strategies shown in Figure 2(d), GTA statistically outperforms PTA. Therefore, MCETRF, ESTF, FRN[4], and GTA are chosen for the BCWS algorithm in the following comparisons.

Makespan of workflow applications is optimized by the combination of the three steps (ISC, RR, BR) (which is called C_{123} (ISC, RR, BR) for convenience here). To further verify the effectiveness of each step, three new algorithms C_{12} (ISC, RR), C_{13} (IC, BR), C_{12} (ISC, RR), are constructed by removing one step each time. The mean plots of the step combinations with 95.0% Tukey honest significant difference (HSD) intervals are shown in Figure 3. It can be observed that the RPD of C_{123} is the lowest, which indicates that the combination of ISC, RR, and BR obtains the best performance. Therefore, all three steps are necessary for effectively reducing the makespan of workflow applications. Removing any of them results in a performance decrease. C_{23} is statistically outperformed by the other three combinations, which demonstrates ISC has the most significant effect on the performance of BCWS. In addition, the RPD of C_{12} slightly outperforms that of C_{13}, which means RR can slightly improve the schedule generated by ISC. The main reason lies in that RR increases the remaining budget rather than directly decreases makespan.

B. Algorithm evaluation

To the best of our knowledge, there is no existing algorithm for the problem under study. To evaluate the performance of the calibrated BCWS algorithm, several algorithms for similar problems are compared. The baselines include CG [5], SMOHEFT [18], and GRP-HEFT [6].

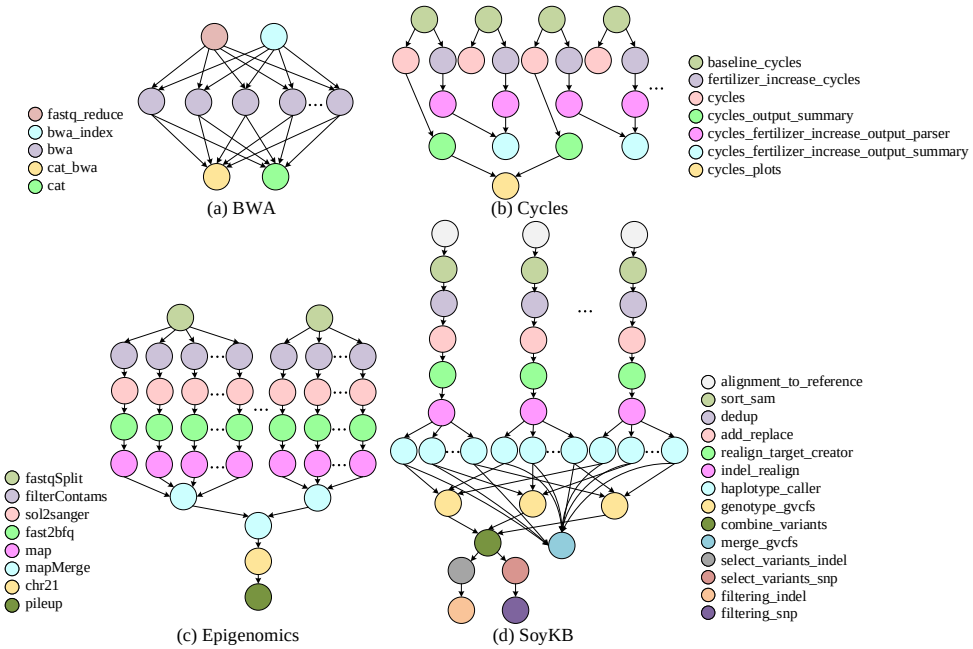

Fig. 4. The structures of four workflow applications: a) BWA, b) Cycles, c) Epigenomics, d) SoyKB.

- Baseline1 (CG): Though CG is designed for renting VMs, the rented VMs are charged by actual usage of time. Therefore, CG is compared in the serverless configuration.
- Baseline2 (SMOHEFT): Both deadline and budget are considered in SMOHEFT. Since no deadline is considered in the problem under study, the deadline is set as infinite for fair comparison. The schedule with the minimal makespan in the Pareto solution set is adopted.
- Baseline3 (GRP-HEFT): GRP-HEFT is developed to minimize the makespan of budget-limited workflow applications by renting heterogeneous VMs charged by a coarse-grained BTU. In this paper, any type of VM can be rented to generate a schedule.

In addition, the calibrated BCWS algorithm attempts to rent one type of VM and multiple types of SF at a time to generate a set of schedules. The minimum makespan of all schedules in the set is selected for comparing to other algorithms.

Four workflow applications: BWA, Cycles, Epigenomics, and SoyKB, are used to evaluate the performance of the calibrated BCWS. BWA is a software package for mapping low-divergent sequences against a large reference genome, such as the human genome. Cycles is a user-friend, multi-crop, multi-year, process-based Agroecosystem model with daily time-step simulations of crop production and the water, carbon, and nitrogen cycles in a soil-plant-atmosphere continuum.

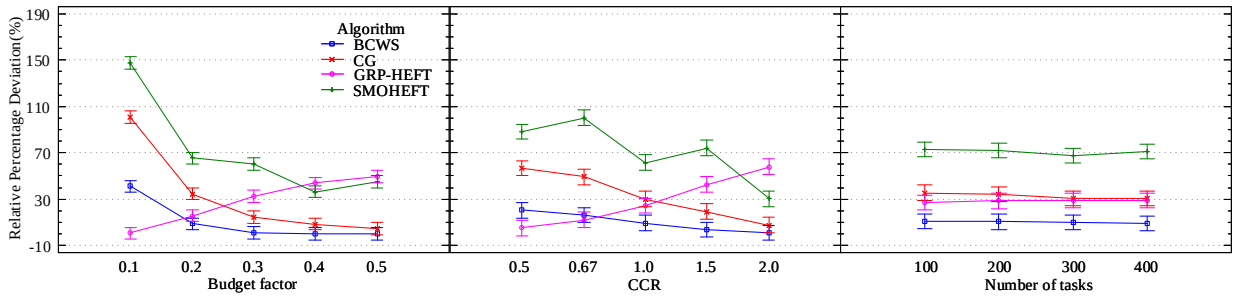

Fig. 5. Interaction plots of the compared algorithms and instance parameters on BWA applications with 95.0% Tukey HSD confidence level intervals.

The Epigenomics workflow is a data processing pipeline for executing various genome sequencing operations. SoyKB is a genomic pipeline that re-sequences soybean germplasm lines selected for desirable traits such as oil, protein, soybean cyst nematode resistance, stress resistance, and the root system architecture. Different workflow applications have different structures, which are shown in Figure 4. The nodes with different colors in each subgraph mean different types of tasks.

WfGen[3] [29] is adopted to generate a lot of synthetic realistic workflow instances by analyzing actual workflow executions. The number of tasks N of each workflow application takes values from $\{100, 200, 300, 400\}$. The budget factor α is selected from $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. The computation communication ratio CCR takes value from $\{0.5, 0.67, 1.0, 1.5, 2.0\}$. For each combination of N, α and CCR in every workflow application, 10 workflow instances are generated. Therefore, $4 \times 4 \times 5 \times 5 \times 10 = 4000$ instances are generated in total. The total number of conducted tests is $4 \times 4000 = 16000$.

To demonstrate the effectiveness of the compared four algorithms in detail, the influence of three factors (N, α, CCR) is evaluated over the four workflow applications, respectively. Interactions between each factor and the compared algorithms on different scientific workflow instances with 95% Tukey HSD intervals are shown in Figures 5 to 8.

In Figures 5 to 8, BCWS statistically outperforms the other three algorithms in almost all cases of all the four workflow applications. This implies that that the hybridization of SFs and VMs is helpful for reducing the makespan within a given budget. CG statistically outperforms

[3]https://wfcommons.org/generator

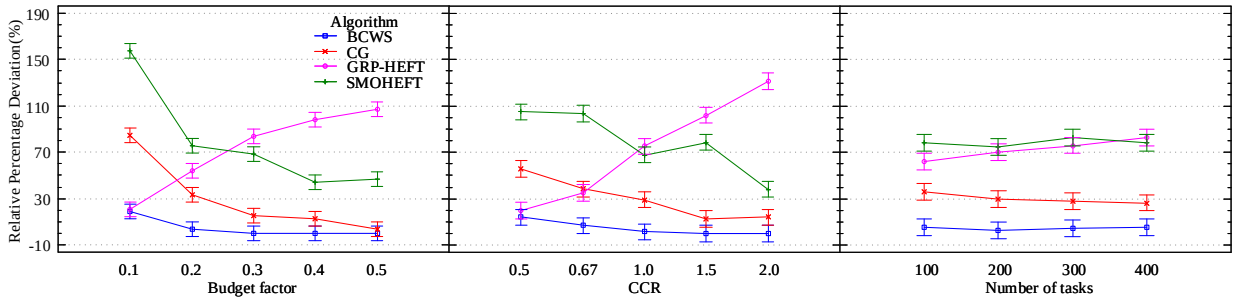

Fig. 6. Interaction plots of the compared algorithms and instance parameters on Cycles applications with 95.0% Tukey HSD confidence level intervals.

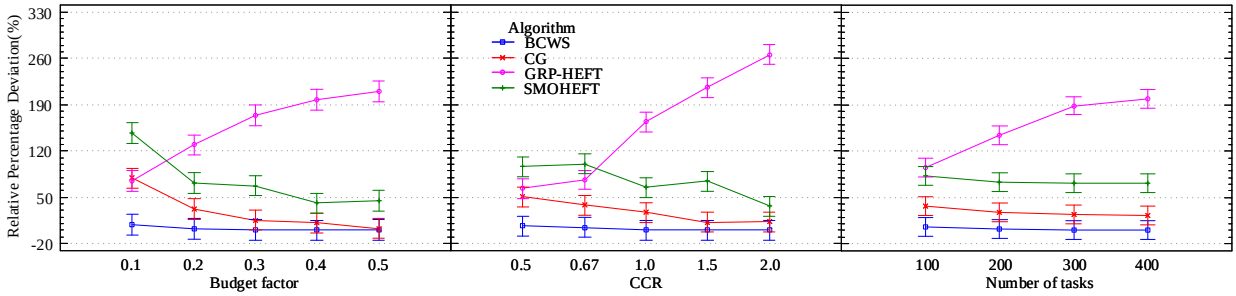

Fig. 7. Interaction plots of the compared algorithms and instance parameters on Epigenomics applications with 95.0% Tukey HSD confidence level intervals.

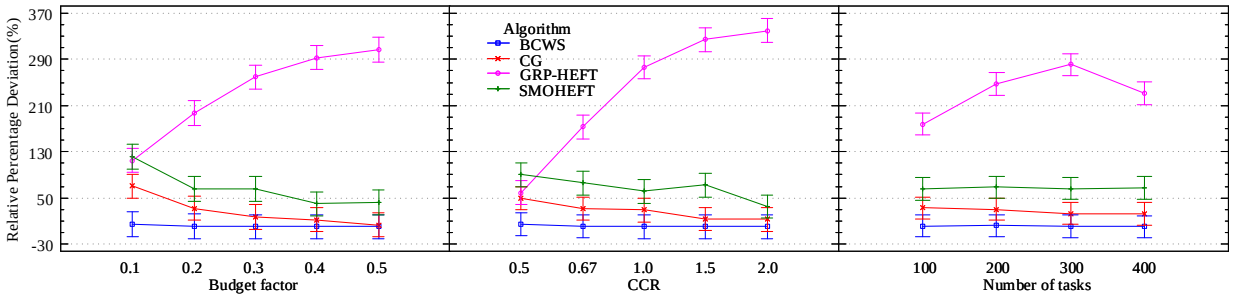

Fig. 8. Interaction plots of the compared algorithms and instance parameters on SoyKB applications with 95.0% Tukey HSD confidence level intervals.

SMOHEFT in all cases since the RPD of CG is always lower than SMOHEFT. The reason lies in that SMOHEFT aims to increase the diversity of schedules instead of minimizing the makespan. GRP-HEFT outperforms CG and SMOHEFT in some cases it is outperformed by CG and SMOHEFT in the other ones. The above results suggest that renting VMs is not always better than renting severless functions and vice visa. In addition, it can be observed that all algorithms have similar RPD tendencies on different workflow applications. Therefore, the structure of the workflow application has slight influence on the performance of the compared algorithms.

Figures 5 to 8 show that the RPDs of CG, SMOHEFT approach to that of BCWS with the increase of the budget factor α of which the reason lies in that more and more tasks are allocated to the fastest SFs. In an extreme case, the RPDs of CG, SMOHEFT, and BCWS are identical if all tasks are allocated to the fastest functions. When the budget factor is small, the RPD of GRP-HEFT is lower than those of CG and SMOHEFT and even lower than that of BCWS. This phenomenon is resulted by two reasons: i) When the budget factor α is too small, most tasks are allocated to the slowest functions and a big makespan is resulted. However, the price of VMs is much lower than that of SFs, some tasks may be allocated to high speed VMs. ii) GRP-HEFT attempts to schedule workflows with a given budget. However, the budget constraint is not strictly guaranteed. The total rental cost in GRP-HEFT may be larger than the budget B when the budget factor α is small.

Figure 5 shows that BCWS outperforms the other three compared algorithms when the communication computation ratio CCR is larger than 0.67 on BWA applications and it is outperformed by GRP-HEFT otherwise. The data transmission time among tasks is a bottleneck for minimizing makespan when CCR is small. However, the data transmission time becomes 0 if two tasks are allocated to the same VM. In such case, the data transmission time is saved in GRP-HEFT and a short makespan is resulted. With the increase of CCR, the performance of GRP-HEFT becomes worse and it is outperformed by the other three algorithms finally. Similar phenomena can be observed in the Cycles, Epigenomics, and SoyKB applications as shown in Figures 6 to 8. In these applications, GRP-HEFT never outperforms BCWS.

RPDs of all the compared algorithms keep robust with the increase in the number of tasks on all applications. That means the performance of the compared algorithms is not closely related to the number of tasks. BCWS always performs the best for cases with any number of tasks. GRP-HEFT performs a little better than CG on BWA applications and performs worse on the others, which indicates that the performance of the compared algorithms is influenced by the

structure of workflow applications to some extent.

VI. Conclusions

In this paper, the WSP problem with budget constraints in a hybrid configuration of servers and serverless functions is studied for which an algorithm framework BCWS is proposed to allocate tasks with the appropriate numbers of SFs and VMs. After calibrating the components and parameters of BCWS over a comprehensive set of random instances, the calibrated algorithm is compared to three existing effective algorithms for only server or serverless configurations over a set of instances generated by realistic workflow patterns. According to the experimental results, we can conclude that the hybridization of VMs and SFs is helpful for reducing the makespan of budget-limited workflows, while neither VM renting nor SF renting is always better than the other. The proposed algorithm minimizes makespan effectively by integrating the advantage of SFs in resource utilization with that of VMs in price. The budget factor and the communication computation ratio significantly influence the effectiveness of the compared algorithms, while the number of tasks and the structure of workflow applications have slight influences on the effectiveness.

There are still many topics worth studying in the future. For example, homogeneous VMs are considered in this paper whereas heterogeneous VMs are more common in actual cloud clusters. In addition, there are many uncertainties during workflow executing.

Acknowledgment

This work is supported by the National Key Research and Development Program of China (No. 2022YFB3305500), the Key-Area Research and Development Program of Guangdong Province (No. 2021B0101200003), the National Natural Science Foundation of China (Nos. 62273089, 62102080), Natural Science Foundation of Jiangsu Province (No. BK20210204), the Fundamental Research Funds for the Central Universities (No. 2242022R10017), and Collaborative Innovation Center of Wireless Communications Technology.

References

[1] X. Xu, W. Dou, X. Zhang, and J. Chen, "Enreal: An energy-aware resource allocation method for scientific workflow executions in cloud environment," *IEEE Transactions on Cloud Computing*, vol. 4, no. 2, pp. 166–179, 2016.

[2] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," *Future Generation Computer Systems*, vol. 48, pp. 1–18, 2015.

[3] W. Zheng and R. Sakellariou, "Budget-deadline constrained workflow planning for admission control," *Journal of grid computing*, vol. 11, no. 4, pp. 633–651, 2013.

[4] V. Arabnejad, K. Bubendorfer, and B. Ng, "Budget and deadline aware e-science workflow scheduling in clouds," *IEEE Transactions on Parallel and Distributed systems*, vol. 30, no. 1, pp. 29–44, 2018.

[5] C. Q. Wu, X. Lin, D. Yu, W. Xu, and L. Li, "End-to-end delay minimization for scientific workflows in clouds under budget constraint," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 169–181, 2014.

[6] H. R. Faragardi, M. R. S. Sedghpour, S. Fazliahmadi, T. Fahringer, and N. Rasouli, "Grp-heft: A budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1239–1254, 2019.

[7] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: a survey," *The Journal of Supercomputing*, vol. 71, no. 9, pp. 3373–3418, 2015.

[8] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE transactions on Parallel and Distributed systems*, vol. 4, no. 2, pp. 175–187, 1993.

[9] H. El-Rewini and T. G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *Journal of parallel and Distributed Computing*, vol. 9, no. 2, pp. 138–153, 1990.

[10] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[11] R. Sakellariou and H. Zhao, "A hybrid heuristic for dag scheduling on heterogeneous systems," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* IEEE, 2004, p. 111.

[12] W. Chen, G. Xie, R. Li, Y. Bai, C. Fan, and K. Li, "Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems," *Future Generation Computer Systems*, vol. 74, pp. 1–11, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X16304411

[13] M. A. Rodriguez and R. Buyya, "Budget-driven scheduling of scientific workflows in iaas clouds with fine-grained billing periods," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 12, no. 2, pp. 1–22, 2017.

[14] S. Qin, D. Pi, and Z. Shao, "Ails: A budget-constrained adaptive iterated local search for workflow scheduling in cloud environment," *Expert Systems with Applications*, vol. 198, p. 116824, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422002809

[15] P. Sun, Z. Cai, and D. Liu, "Budget constraint bag-of-task based workflow scheduling in public clouds," in *Computer Supported Cooperative Work and Social Computing*, Y. Sun, T. Lu, Z. Yu, H. Fan, and L. Gao, Eds. Singapore: Springer Singapore, 2019, pp. 243–260.

[16] Z. Wen, Y. Wang, and F. Liu, "Stepconf: Slo-aware dynamic resource configuration for serverless function workflows," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1868–1877.

[17] J. Jarachanthan, L. Chen, F. Xu, and B. Li, "Astrea: Auto-serverless analytics towards cost-efficiency and qos-awareness," *IEEE Transactions on Parallel and Distributed Systems*, 2022.

[18] M. Majewski, M. Pawlik, and M. Malawski, "Algorithms for scheduling scientific workflows on serverless architecture," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 782–789.

[19] Z. Xu, H. Zhang, X. Geng, Q. Wu, and H. Ma, "Adaptive function launching acceleration in serverless computing platforms," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2019, pp. 9–16.

[20] J. R. Gunasekaran, P. Thinakaran, N. C. Nachiappan, M. T. Kandemir, and C. R. Das, "Fifer: Tackling resource underutilization in the serverless era," in *Proceedings of the 21st International Middleware Conference*, 2020, pp. 280–295.

[21] Q. Jiang, Y. C. Lee, and A. Y. Zomaya, "Serverless execution of scientific workflows," in *International Conference on Service-Oriented Computing*. Springer, 2017, pp. 706–721.

[22] R. B. Roy, T. Patel, V. Gadepally, and D. Tiwari, "Mashup: making serverless computing useful for hpc workflows via hybrid execution," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2022, pp. 46–60.

[23] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys (CSUR)*, vol. 31, no. 4, pp. 406–471, 1999.

[24] X. Zhou, G. Zhang, J. Sun, J. Zhou, T. Wei, and S. Hu, "Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based heft," *Future Generation Computer Systems*, vol. 93, pp. 278–289, 2019.

[25] C. Lin and H. Khazaei, "Modeling and optimization of performance and cost of serverless applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 615–632, 2020.

[26] T. Elgamal, "Costless: Optimizing cost of serverless computing through function fusion and placement," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 300–312.

[27] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[28] A. Raza, Z. Zhang, N. Akhtar, V. Isahagian, and I. Matta, "Libra: An economical hybrid approach for cloud applications with strict slas," in *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2021, pp. 136–146.

[29] T. a. Coleman, H. Casanova, L. Pottier, M. Kaushik, E. Deelman, and R. Ferreira da Silva, "WfCommons: A Framework for Enabling Scientific Workflow Research and Development," *Future Generation Computer Systems*, vol. 128, pp. 16–27, 2022.

Jinquan Zhang received his B.Sc. in College of Science from Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2017. He is currently a Ph.D. candidate at the School of Computer Science and Engineering, Southeast University, Nanjing, China. His research interests focus on reinforcement learning, combinatorial optimization, and cloud resource scheduling.

Xiaoping Li (M09-SM12) received his B.Sc. and M.Sc. degrees in Applied Computer Science from the Harbin University of Science and Technology in 1993 and 1999, respectively, and the Ph.D. degree in Applied Computer Science from the Harbin Institute of Technology in 2002. He is a distinguished professor at the School of Computer Science and Engineering, Southeast University, Nanjing, China. He is the author or co-author over more than 100 academic papers, some of which have been published in international journals such as *IEEE Transactions on Computers*; *IEEE Transactions on Parallel and Distributed Systems*; *IEEE Transactions on Services Computing*; *IEEE Transactions on Cybernetics*; *IEEE Transactions on Automation Science and Engineering*; *IEEE Transactions on Cloud Computing*; *IEEE Transactions on Systems, Man and Cybernetics: Systems*; *Information Sciences*; *Omega*, *European Journal of Operational Research*; *International Journal of Production Research*; *Expert Systems with Applications* and *Journal of Network and Computer Applications*. His research interests include Scheduling in Cloud Computing, Scheduling in Cloud Manufacturing, Service Computing, Big Data and Machine Learning.

Long Chen received his B.Sc. and Ph.D. degrees in Computer Science and Engineering from Southeast University, Nanjing, China, in 2009 and 2018 respectively. He is currently a lecture in the Department of Computer Science and Engineering at Southeast University, Nanjing, China. He has published more than 20 papers in international journals and conferences, such as *IEEE Transactions on Services Computing*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Automation Science and Engineering,*. His main interest includes task scheduling in cloud computing, service-oriented computing, evolutionary computation, data normalization in smart building.

Rubén Ruiz is a full professor of Statistics and Operations Research at the Universitat Politècnica de València, Spain. He is co-author of more than 100 papers in International Journals and has participated in presentations of more than a hundred papers in national and international conferences. He is editor of the Elsevier journal *Operations Research Perspectives (ORP)* and co-editor of the JCR-listed journal *European Journal of Industrial Engineering (EJIE)*. He is also associate editor of other important journals like *TOP* as well as member of the editorial boards of several journals most notably *European Journal of Operational Research* and *Computers and Operations Research*. He has been principal investigator of several public research projects as well as privately funded projects with industrial companies. His research interests include scheduling and routing in real life scenarios.

Algorithm 3: Resource Replacement (RR)

Input: Schedule S

Output: Adjusted schedule S

```
begin
    Flag ← True;
    CP ← Calculate the critical path;
    while Flag do
        Flag ← False; TS ← ∅;
        foreach t_i ∈ T do
            if t_i is allocated to a SF then
                k ← Function type of t_i;
                if p_k ≤ p_vm then
                    TS ← TS ⋃ {t_i}
        Q ← Sort TS according the start time of tasks;
        while Q ≠ ∅ do
            t_⋆ ← Get the first task in Q ;
            v ← Rent a new VM instance ;
            Rented BTUs n ← ⌈ET(t_⋆)/BTU⌉ ;
            ST(v) ← ST(t_⋆) ;
            REL(v) ← ST(t_⋆) + n * BTU ;
            foreach t_i ∈ Q do
                if Eqn. (30) is satisfied then
                    Allocate t_i to v;
            C_old ← Rental cost before replacement;
            C_new ← Rental cost after replacement;
            if C_old > C_new then
                Flag ← True;
                RB ← RB + C_old − C_new ;
                break;
            else
                Release v;
                Q ← Q − {t_⋆};
    return.
```

Algorithm 4: Budget Reallocation (BR)

begin
  $CostLeft \leftarrow RB$; $Flag \leftarrow True$;
  while $CostLeft > 0$ *and* $Flag = True$ **do**
    Calculate the critical path CP;
    $CPTask \leftarrow \emptyset$;
    $MS \leftarrow$ Makespan of the current schedule;
    foreach $t_i \in CP$ **do**
      if t_i *is executed by a SF* **then**
        $k \leftarrow$ Current function type of t_i;
      else
        $k \leftarrow \max\{s|p_{is} \leq p_{vm}\}$;
      if $k + 1 \leq K$ **then**
        $\triangle ET(t_i) \leftarrow$ Calculate time decrease;
        $\triangle C(t_i) \leftarrow$ Calculate cost increase;
        $r_{tc}(t_i) \leftarrow \triangle ET(t_i)/\triangle C(t_i)$;
        $CPTask \leftarrow CPTask \bigcup \{t_i\}$;
    if $CPTask = \emptyset$ **then**
      break;
    else
      $t_\star \leftarrow \arg\max_{t_i \in CPTask} r_{tc}(t_i)$;
      if t_i *is executed by a SF* **then**
        $k \leftarrow$ Current function type of t_i;
      else
        $k \leftarrow \max\{s|p_{is} \leq p_{vm}\}$;
      Move $t_\star$ to the function with type $k + 1$;
      $C' \leftarrow$ Recalculate the total rental cost ;
      $MS' \leftarrow$ Recalculate the makespan ;
      if $C' > B$ *or* $MS' > MS$ **then**
        Move $t_\star$ back to the resource it allocated to before adjusting ;
        $Flag \leftarrow False$;
      else
        $CostLeft \leftarrow B - C'$;
  $RB \leftarrow CostLeft$;

TABLE III

RESOURCES CAN BE RENTED IN THIS PAPER.

Number	Type	CPU cores	Memory size (GB)	Price ($/h)
0	VM	1	2	0.0255
1	VM	2	4	0.051
2	VM	4	8	0.102
3	VM	8	16	0.204
4	SF	0.25	0.5	0.01234
5	SF	0.5	1	0.02469
6	SF	1	2	0.04937
7	SF	2	4	0.09874
8	SF	4	8	0.19748
9	SF	8	16	0.39496
10	SF	16	32	0.78992