Budget-Constrained Contention-Aware Workflow Scheduling in a Hybrid Cloud

Qingliang Zhang, Xinyue Shu, and Quanwang Wu(✉)

Chongqing University, Chongqing, China
zql@stu.cqu.edu.cn, wqw@cqu.edu.cn

Abstract. Private clouds offer high controllability but lack scalability, while public clouds provide high scalability with limited controllability. The hybrid cloud paradigm combining them can well balance controllability and scalability nowadays. Increasing numbers of workflow applications are being deployed on a hybrid cloud and this paper investigates how to effectively minimize workflow makespan within a user-specified budget. It first establishes a practical communication contention-cognizant workflow scheduling model for hybrid clouds, where a queueing mode is used to handle multiple data contending for the scarce cross-cloud bandwidth resources. A Budget-constrained Contention-aware Workflow Scheduling (BCWS) heuristic is proposed to optimize the workflow makespan within a given budget. It chooses a subset of cloud instances and allocates each task from the workflow to computing resources sequentially with its data communications scheduled as well. In experiments, traditional contention-agnostic scheduling techniques such as GRP-HEFT, PSO, IPPTS etc. are adapted to the considered model before comparison with the proposed method BCWS. Experimental results verify the superiority of BCWS as it can always achieve the best makespan.

Keywords: Hybrid Cloud · Communication Contention · Budget Constraint · Workflow Scheduling

1 Introduction

As a new computing paradigm, cloud computing provides highly scalable applications, platforms, and hardware services to end-users through the Internet. In particular, private clouds offer high controllability but lack scalability, while public clouds provide high scalability but limited controllability. Hence, the hybrid cloud paradigm that combines the two provides a superior balance of controllability and scalability. In this paradigm, end-users can use public cloud resources based on a "pay-as-you-go" mode, and keep sensitive data in private cloud, thus significantly reducing costs and operational expenses. In a hybrid cloud environment, workflow scheduling faces two challenges: effective scheduling and communication contention.

H. Gao et al. (Eds.): CollaborateCom 2023, LNICST 561, pp. 111–127, 2024.
https://doi.org/10.1007/978-3-031-54521-4_7

First, effective scheduling under budget constraints refers to the rational arrangement and allocation of tasks resources within a computing system to maximize resource utilization. It includes two inseparable segments [1], namely 1) resource provisioning, and 2) task scheduling. In the context of a limited budget, providing appropriate resources to meet task requirements is crucial. This involves intelligently allocating computing resources, necessitating a balance between performance and cost to achieve optimal resource provisioning. Task scheduling focuses on optimizing task sequences and execution times to minimize the makespan. In task scheduling, tasks involving sensitive information should be allocated within private cloud resources to address privacy issues [2, 3]. This comprehensive approach optimizes performance, resource utilization, and budget adherence, fostering stability and efficiency within hybrid cloud environments.

Second, communication contention refers to the situation where multiple data streams compete for limited bandwidth resources. When workflows are deployed in a hybrid cloud, data transfers may take place between public and private clouds or within a single cloud platform. Since there is only one duplex communication channel connecting the private and public platforms, if multiple senders transmit data across the platforms, multiple data may compete for scarce bandwidth resources, which may cause further delays. Although there are also a few studies concentrating workflow scheduling in hybrid clouds [4, 5], none of them tackle the potential data communication contention for workflow scheduling. Thus, communication contention can easily become a bottleneck in executing workflows.

With the above challenges as motivation, this paper establishes a practical scheduling model to capture workflow deployment in hybrid clouds. We introduce an instance provisioning mechanism and a workflow scheduling algorithm, named budget-constraint contention-aware workflow scheduling (BCWS). BCWS contains two stages: (1) Resource Provisioning (RP) and (2) Task and Data Scheduling (TDS). RP focuses on selecting instances that are billed on an hourly basis when making purchase decisions. TDS schedules tasks with incoming data to computing and communication resources sequentially. Moreover, a look-ahead task selection mechanism is designed. The experiments are conducted with realistic workflows, and the results verify the superiority of BCWS.

The remaining part is structured in the following manner. Section 2 reviews the scheduling methods which have been proposed in literature. Section 3 introduces the problem formulation and system model for hybrid cloud, while Our proposed method BCWS is presented in Sect. 4. Further, Sect. 5 gives experimental results and analysis, and Sect. 6 concludes the whole paper and points out future work.

2 Related Work

We first review the literature on workflow scheduling under budget constraint and then discuss communication contention in workflow execution.

2.1 Budget-Constrained Workflow Scheduling

In [6], the authors introduced two budget-constrained algorithms, namely “Loss” and “Gain,” for scheduling Directed Acyclic Graph (DAG) workflows in grid environments.

In [7], Jia Yu et al. proposed a budget-constraint scheduling approach that utilized genetic algorithms to optimize workflow execution time while meeting the budget. In [8], the Budget-constrained Heterogeneous Earliest Finish Time (BHEFT) was proposed, which is an extension of the HEFT algorithm. BHEFT defines a suitable plan by minimizing the makespan so that the user's budget and deadline constraints are met. With the emergence of hybrid clouds, workflow scheduling has embraced the use of hybrid clouds to harness the benefits of both internal and external resources. The scheduling approach can be described as a cost model aimed at minimizing execution time while considering the rental cost of public resources. This is achieved by providing the user with a list of resources available for execution [9]. Regarding the billing of instance usage time, cloud providers offer various billing options, including billing by seconds and billing by hours. Users can select the billing mode that best suits their specific needs. Billing by seconds is ideal for short-duration usage scenarios, as users are charged only for the actual number of seconds used [10]. On the other hand, billing by hours is more suitable for longer-duration usage, with charges being calculated on an hourly basis, rounding up any partial hour to the nearest whole hour [9].

However, many existing budget-constrained workflow methods consider instance billing based on billing by seconds rather than billing by hours. The integral instance hour increases the difficulty for solving the makespan minimization problem. In [11], Xiaotong Wang et al. introduced an evolutionary algorithm based on Particle Swarm Optimization (PSO). This PSO-based approach considers the hourly-based cost model in the context of budget-constrained workflow scheduling. In [12], the authors propose a new algorithm named GRP-HEFT which includes two steps, a new resource provision strategy and assigns tasks based on the HEFT algorithm, achieving very good results.

2.2 Communication Contention in Workflow Execution

Communication contention refers to the situation that multiple data communications contend for a specific network resource at the same time. Task duplication in contention-aware scheduling models is studied in [13]. Benoit et al. [14] introduced a bidirectional one-port architectural model to capture endpoint contention. Efficient contention-aware and fault-tolerant scheduling algorithms are then designed. Recently, Özkaya et al. [15] described new list-based scheduling heuristics based on clustering for homogeneous processors, based on a duplex single-port communication model.

To avoid network communication contention, Genez et al. [16] introduced an estimation mechanism to address imprecise information regarding the available inter-cloud link bandwidth and how it affects the estimation of makespan and cost. Son et al. [17] proposed priority-aware resource placement algorithms considering both host and network resources for software-defined networking-enabled clouds, where network flows can be reconfigured dynamically and adapted to network traffics. For data-intensive workflow scheduling in a DC, Wu et al. [18] proposed a practical scheduling model that takes into account communication contention between endpoints in the interfaces connecting servers with a communication network. Efficient contention-aware scheduling algorithms are then designed. Mithila propose latency-based vector scheduling of many-task applications, where communication latency is measured for making decisions on task allocation [19].

3 Workflow Scheduling Model in a Hybrid Cloud

In this section, we formally describe the budget-constrained contention-aware workflow scheduling problem in hybrid cloud. We first present the workflow and resource model, and then describe the scheduling model with contention awareness. Finally, we formulate the problem mathematically as a constrained problem.

3.1 Workflow and Resource Model

A directed acyclic graph (DAG) can represent a workflow application, DAG = (N, E, W, D), where, (1) N represents a set of tasks which are indivisible individual applications, $N = \{n_1, n_2, \ldots, n_k\}$. (2) E denotes a set of edges representing precedence dependencies among tasks. A precedence dependence $e_{i,j} \in E$ between tasks $n_i \in N$ and $n_j \in N$ indicates that n_j can start executing only after n_i finishes. (3) W: is a weighing vector whose w_i represents the computation load of $n_i \in N$. (4) D: $E \rightarrow R^+$ is a matrix whose entry $d_{i,j}$ indicates data amount to be transferred from n_i to n_j. To generalize a workflow with one entry and one exit, two dummy tasks n_{entry} and n_{exit} with zero computational workload are added to its start and end, respectively. $succ(n_i)$ and $pred(n_i)$ denote the set of n_i's immediate successors and immediate predecessors, respectively.

In hybrid cloud, a private cloud provides a limited number of computation resources $R = \{r_1, r_2, \ldots, r_m\}$, while a public cloud provides an 'infinite' number of resources $U = \{u_1, u_2, \ldots, u_n\}$. U are used on demand. A hybrid cloud is a set of resources, consisting of resources on the private cloud and resources of the public cloud, which is denoted as $V = R \cup U = \{v_1, v_2, \ldots, v_l\}$. Based on the computational capacity and price of the resources, we can classify V into different types of resources, which are represented by $P = \{p_1, p_2, \ldots, p_k\}$.

We use $s_{i,j}$ to indicate the execution time of the n_i on v_j with the type of p_k. Cloud providers specify the processing power of different types of the provided instances use a metric names Compute Unit (CU) or similar concepts such as mean performance [20]. Hence, we denote the compute unit of p_k by CU_k. A higher value of CU potentially results in greater computing capacity. For several scientific standard workflows, the execution time of tasks on Xeon@2.33 GHz CPUs (CU = 8) has been published [21]. This principle resembles the commonly applied concept for estimating the task execution duration within scientific workflows, especially in the domain of static workflow scheduling [20, 22]. The CU value is inversely correlated with the task execution time. The task n_i execution time for the instance with $CU = 1$ is the reference execution time of the task, denoted by $ref\_time(w_i)$. Accordingly, the execution time of task n_i on v_j with the type of p_k, is

$$s_{i,j} = \frac{ref\_time(w_i)}{CU_k} \tag{1}$$

Furthermore, privacy is a major point of concern for users when executing workflows. Sending all data to a public cloud could pose a potential risk for workflow users. In hybrid clouds, users can keep sensitive tasks, denoted as S ($S \subseteq N$), residing in a private cloud. Remaining tasks (i.e., $O = N\text{-}S$) that do not involve privacy issues can be executed on both R and U.

For data transmission, the external network channel connecting the two platforms goes through the Internet, and its corresponding bandwidth resource, denoted as b_{ext}. Without losing generality, we assume that the internal bandwidth is the same for each platform, which is denoted as b_{int}.

3.2 Contention-Aware Scheduling Model

When deploying workflows in a hybrid cloud, data transfers may take place between public and private clouds or within a single cloud platform. Due to only one duplex communication channel connecting the public and private cloud platforms, if multiple senders transfer data across platforms at the same time, multiple data may compete for scarce bandwidth resources. On the contrary, since the computing resources within the cloud platform are well connected to each other based on advanced data center networking technologies (e.g., software defined networking) [17], the influence of network communication contention can be disregarded in comparison to inter-platform bandwidth resources when multiple data communications occur simultaneously within a cloud platform.

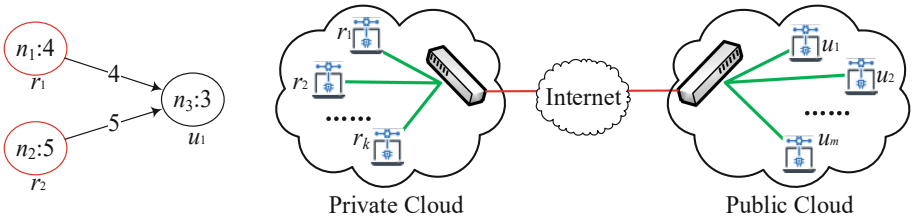

Fig. 1. Simple workflow example and data transmission in a hybrid cloud

Figure 1 illustrates a deployment scheme of a workflow on a hybrid cloud. The green lines indicate the internal data links and red lines indicate the external data links. The communication contention occurs in the red lines. Suppose that both tasks n_1 and n_2 marked in red indicate that they contain sensitive information, and are respectively allocated to resources r_1 and r_2. Task n_3 is allocated to resource u_1. In addition, the bandwidth resources of external data channels and speed of private computation resources and are both 1. According to traditional scheduling model [4, 23, 24], the start time of n_3 on resource u_1 is the maximum value among its predecessors' finish time plus data transfer time, resulting in a start time of 10. However, this scheduling method is not suitable. When n_1 and n_2 try to send data to n_3, communication contention occurs for external data channels.

Specifically, each data communication involved in contention proceeds one after another to avoid contention and the communication order can be determined by policies like FCFS (First Come First Service). For example, in Fig. 1, communication from n_1 to n_3 proceeds first ahead of that from n_2 to n_3. In this way, the time for n_1 to finish sending data to n_3 is 8, that for n_2 is 13, so the start time of n_3 on u_1 is 13.

To address the communication contention, this paper adopts the queuing approach, which means that each data sender is allowed to exclusively occupy the whole bandwidth

resource. Suppose that task n_i is scheduled to execute on the cloud platform $p(n_i)$ and between them there is a precedence dependence $e_{i,j}$. When n_i and n_j are deployed in the same cloud ($p(n_i) = p(n_j)$), the data transfer of $e_{i,j}$ can start when task n_i finishes. If $p(n_i) \neq p(n_j)$, the data transfer $e_{i,j}$ can start when task n_i finishes and b_{ext} is ready for transferring $e_{i,j}$. That is, it additionally requires that the external bandwidth resource is not occupied by other data senders. Let $t_{\diamond}(e_{i,j}, b_{ext})$ be the earliest time when b_{ext} is ready for transferring $e_{i,j}$ and $t_{\bullet}(n_i)$ be n_i's finish time. The data transfer of $e_{i,j}$ can start at

$$t_{\mathrm{o}}(e_{i,j}) = \begin{cases} t_{\bullet}(n_i) & \text{if } p(n_i) = p(n_j) \\ \max\{t_{\bullet}(n_i),\ t_{\diamond}(e_{i,j}, b_{ext})\} & \text{otherwise} \end{cases} \tag{2}$$

Queueing mode stipulates that no contention from other senders emerges during the transmission of $e_{i,j}$. If n_i and n_j reside in the same cloud, b_{int} can be used for calculating $e_{i,j}$'s transfer finish time. Otherwise, b_{ext} can be used for calculating $e_{i,j}$'s transfer finish time. In addition, when n_i and n_j are assigned to in the same instance (i.e. $v_l = v_m$), $e_{i,j}$'s transfer time is 0 because the intermediate data can be direct accessed locally. In summary, $e_{i,j}$'s transfer finish time can be obtained via

$$t_{\bullet}(e_{i,j}) = \begin{cases} t_{\mathrm{o}}(e_{i,j}), & \text{if } p(n_i) = p(n_j),\ v_l = v_m \\ t_{\mathrm{o}}(e_{i,j}) + d_{i,j}/b_{ext}, & \text{if } p(n_i) \neq p(n_j) \\ t_{\mathrm{o}}(e_{i,j}) + d_{i,j}/b_{int}, & \text{otherwise} \end{cases} \tag{3}$$

When all task n_i's input data have arrived at v_l and v_l is ready to execute n_i, n_i can start on v_l. Let $t_{\diamond}(n_i, v_l)$ be the earliest time at which v_l is ready for executing n_i, and $t_{\triangle}(n_i, v_l)$ be the time of all n_i's input data arriving at v_l. Therefore, the start time of n_i on v_l can be calculated as

$$\begin{aligned} t_{\mathrm{o}}(n_i, v_l) &= \max\{t_{\diamond}(n_i, v_l),\ t_{\Delta}(n_i,\ v_l)\} \\ &= \max\{t_{\diamond}(n_i, v_l),\ \max_{n_p \in pred(n_i)}\{\ t_{\bullet}(e_{p,i})\}\} \end{aligned} \tag{4}$$

The finish time of n_i on v_l can be obtained via

$$t_{\bullet}(n_i, v_l) = t_{\mathrm{o}}(n_i, v_l) + s_{i,l} \tag{5}$$

The makespan of a workflow refers to the total execution time required for completing all tasks and it can be gained via

$$\text{makespan} = \max_{n_i \in N}\{t_{\bullet}(n_i)\} \tag{6}$$

To ensure the budget constraint for running workflow, the cost(C) can be obtained via

$$C = \sum_{v_l \in M_V} Pr(p_k) \times \left\lceil \frac{t_{\bullet}(v_l) - t_{\mathrm{o}}(v_l)}{3600} \right\rceil + \sum_{e_{i,j} \in E,\ p(n_i) \neq p(n_j)} d_{i,j} \times Pr(b_{ext}) \tag{7}$$

Where $Pr(p_k)$ represents the cost of execution of the v_l with the type p_k for an hour usage. And $t_{\mathrm{o}}(v_l)$ is the the starting time point of instance v_l's execution, while $t_{\bullet}(v_l)$

is v_l's completion time. We utilize an instance model billed on an hourly basis. As all execution times in our system are specified in seconds, we divide them by 3,600 to determine the hourly cost. $Pr(b_{ext})$ represents the billing of transferring data across clouds. The networking expenses of the intra-cloud are deemed negligible, while the networking expenses of the inter-cloud need to be billed.

3.3 Problem Definition

The solution not only includes assigning tasks to time slices of computational resources, but also assigning data transfers to time slices of communication resources. Specifically, the scheduling solution can be represented as a ternary $\{M_N, M_E, M_V\}$, where:

1) M_N includes all task-resource mappings. Each task mapping is a 4-tuple $[n_i, v_l, t_o(n_i), t_\bullet(n_i)]$, which indicates that task n_i is deployed on the computation resource v_l, during the period from $t_o(n_i)$ to $t_\bullet(n_i)$.
2) M_E includes mappings between data transmission and external bandwidth resources b_{ext}, and each mapping is a 4-tuple $[e_{i,j}, \mu, t_o(e_{i,j}), t_\bullet(e_{i,j})]$, which indicates that if the Boolean variable μ is true, data communication $e_{i,j}$ occupies the upstream channel of b_{ext} during the time from $t_o(e_{i,j})$ to $t_\bullet(e_{i,j})$, and otherwise the downstream channel is busy.
3) M_V represents the public cloud resources used and the start and end times of their operation and each mapping is a 3-tuple $[v_l, t_o(v_l), t_\bullet(v_l)]$.

Based on the above analysis, we aim to minimize the makespan when deploying workflows in a hybrid cloud environment without violating the budget B constrain, which can be formulated as the following constrained optimization problem:

$$\begin{aligned} &\min \ makespan \\ &\text{s.}t.\ C \leq B \end{aligned} \tag{8}$$

4 Methodology

This section describes a Budget-constrained Contention-aware Workflow Scheduling (BCWS) heuristic. It includes two basic stages. 1) Resource Provisioning (RP) strategy, which is responsible for choosing a subset of instances from all instances in the pool of cloud IaaS resources, and 2) Task and Data Scheduling (TDS), which not only allocates each task from the list to computing resources but allocates inter-platform data communications to b_{ext} explicitly.

4.1 Resource Provisioning Strategy

The first stage consists of two steps: 1) selecting the instance type and 2) selecting the corresponding quantity for each type of instance.

Algorithm 1 Resource Provisioning (RP)

1: Assign all non-dominated instance types of public cloud to the set I;
2: Sort I according to the efficiency rate of the instance types;
3: **while** I is not empty **do**:
4: *remain Budget*(R)←B;
5: The set of taken instances $Y = \{\}$;
6: **while** ($R >= p^{Cheapest}$) **do**:
7: pick instance p_i from the I;
8: Take $n = \lfloor \boldsymbol{R/Pr(p_i)} \rfloor$ instances from the type p_i and add them to Y;
9: $R=R - Pr(p_i)\times n$;
10: **end while**;
11: $\{M_N, M_E, M_V\}$ ← run TDS (Algorithm 2) with Y and R;
12: Calculate makespan and cost C of the created assignment $\{M_N, M_E, M_P\}$;
13: **if** B-C can buy more efficient instance type p_j **then**:
14: add instance from type p_j to Y;
15: $\{M_N, M_E, M_V\}$ ← rerun TDS with Y;
16: **end if;**
17: remove the first item from the I;
18: **end while**;
19: return the best assignment $\{M_N, M_E, M_V\}$;

Initially, we begin by examining the list of instance types to identify and eliminating any dominated instance types. An instance type p_j is dominated by instance type p_i if $Pr(p_i) \leq Pr(p_j)$ and $CU_i \geq CU_j$. The remaining non-dominated instances after removing the dominated ones are then appended to a list referred to as I. Subsequently, the list is sorted in descending order based on the efficiency rate (ER) of each instance type (line 2), which is defined as follows:

$$ER_k = \frac{CU_k}{Pr(p_k)} \tag{9}$$

The workflow scheduling gives priority to instances with a higher efficiency rate (Line 1–2). We prioritize selecting as many instances as possible from the most efficient instance type and include them in the Y (Line 7–8). The remaining budget R (i.e., the available budget is insufficient to acquire any additional most-efficient instances) is used to buy the second most efficient instance. This process continues until the remaining budget is not enough even to take one more of the cheapest instances (Line 6–10). $p^{Cheapest}$ denotes the price of the cheapest instance type.

The generated set of instances Y, is given as an input to the TDS algorithm, and the scheduling phase begins (Line 11). The TDS completes the assignment by filling the values of $\{M_N, M_E, M_V\}$. The details of TDS will be explained in Sect. 4.2.

Pay attention to the remaining amount (i.e., Budget B minus total cost C). If a more efficient instance (refer to instance types that have been removed from the I) can be purchased, execute it again after purchase. Compare the new makespan with the previous one and record the better one (Line 13–16). Next, the most-efficient instance type is removed from I, and the algorithm continues to the next iteration with the remaining items in the I. The algorithm terminates when the list is empty. Eventually, the best assignment found across all iterations returns the final solution.

4.2 Task and Data Scheduling

Constructing an ordered list of tasks by a certain attribute is the basis for executing scheduling algorithms.

Here, a task property for ordering tasks is defined which is specific to the considered model. Before presenting it formally, the concept of outbound communication duration for a task n_i, denoted as $\lambda(n_i)$, is first introduced, which represents the sum of all outgoing communication time of n_i. Considering sensitive tasks(S) can only be allocated to the private cloud and it is better to allocate insensitive tasks(O) to the public cloud for faster processing, it is defined based on tasks' privacy sensitivity as,

$$\lambda(n_i) = \max \left\{ \max_{n_k \in X \cap succ(n_i)} \frac{d_{i,k}}{b_{int}}, \sum_{n_j \in (N-X) \cap succ(n_i)} \frac{d_{i,j}}{b_{ext}} \right\} \tag{10}$$

where, X represent tasks in the workflow with the same privacy sensitivity with n_i, that is, if n_i is a sensitive task, X is S, and otherwise it is O. For a sensitive task, if its successor task is also privacy sensitive, the internal bandwidth b_{int} is used for calculation, and otherwise b_{ext} is used instead.

Based on this design, we define a new heterogeneity and contention-oriented upward rank μ for task ordering as follows:

$$\mu(n_i) = \lambda(n_i) \times (1 + \frac{\delta^+(n_i)}{\delta^*}) + \max_{n_j \in succ(n_i)} \{\mu(n_j)\} + \bar{s} \tag{11}$$

$\delta^+(n_i)$ represents n_i's out-degree (i.e., the number of n_i's immediate successor tasks), and δ^* represents the maximum out-degree value of tasks in the given workflow graph. The property μ of task n_i measures its execution time, its outbound communication duration $\lambda(n_i)$, and the maximum μ value among its successors. Note that the impact of $\lambda(n_i)$ is additionally enhanced by multiplying the coefficient $1 + \delta^+(n_i)/\delta^*$. This is because a larger $\delta^+(n_i)$ value implies that n_i has more successor tasks, and the finish of n_i can trigger execution of more tasks in parallel. Hence, the normalized $\delta^+(n_i)$ value is introduced to reflect the priority of a task in this regard. $\bar{s}$ represents the average runtime of n_i on virtual machines. $\mu(n_{exit})$ is 0, and by traversing the workflow graph upward from n_{exit}, the μ property of each task can be recursively calculated.

The whole procedure of TDS is shown in Algorithm 2. In this phase, not only tasks are assigned to instance resources, but data transmissions are also allocated to bandwidth resources. At first, prepare the computation resources ψ of private cloud (Line 1). Then, the task list is constructed by sorting tasks in a descending order of μ values (Line 2). TDS then traverses the tasks in the list in sequence and chooses a computation resource for each task. Additionally, it allocates cross-cloud data communications to external bandwidth resources.

Algorithm 2 Task and Data Scheduling

1: prepare computation resource ψ of private cloud;
2: sort tasks by a decreasing order of $\mu(n_i)$;
3: **for** each task n_i in the task list **do**
4: $best \leftarrow +\infty$, $T_1 \leftarrow \emptyset$, $T_2 \leftarrow \emptyset$, $best^* \leftarrow +\infty$, $T^*_1 \leftarrow \emptyset$, $T^*_2 \leftarrow \emptyset$;
5: prepare *candidates* for n_i;
6: sort $pred(n_i)$ by their finish time increasingly;
7: calculate and prepay $c(n_i)$;
8: **for** each v_l in *candidates* **do**
9: **for** each n_i's predecessor n_j **do**
10: calculate $t_o(e_{j,i})$ via (2) and $t_\bullet(e_{j,i})$ via (3);
11: **if** $p(n_j) \neq p(v_l)$ **then**
12: tentatively allocate $e_{j,i}$ to b_{ext};
13: **end if**
14: **end for**
15: calculate $t_o(n_i, v_l)$ via (4), $t_\bullet(n_i, v_l)$ via (5) and $t_\Delta(n_i, v_l)$ via (12);
16: **if** $t_\Delta(n_i, v_l) < best$ **then**
17: $best \leftarrow t_\Delta(n_i, v_l)$;
18: $T_1 \leftarrow$ record the above tentative operations;
19: $T_2 \leftarrow <n_i, v_l, t_o(n_i), t_\bullet(n_i)>$;
20: **end if**
21: **if** $t_\Delta(n_i, v_l) < best^*$ and $Adding\_Not\_Change\_the\_Cost(n_i, v_l)$ **then**
22: $best^* \leftarrow t_\Delta(n_i, v_l)$;
23: $T^*_1 \leftarrow$ record the above tentative operations;
24: $T^*_2 \leftarrow <n_i, v_l, t_o(n_i), t_\bullet(n_i)>$;
25: **end if**
26: withdraw the above tentative operations;
27: **end for**
28: choose $\{T_1, T_2\}$ or $\{T^*_1, T^*_2\}$ to allocate edges and n_i based on the values of T_2 and T^*_2;
29: recovering a portion of the prepaid amount;
30: **end for**
31: return the generated schedule $\{M_N, M_E, M_V\}$;

When preparing computation resource *candidates* for n_i (Line 5), only private cloud resources ψ are considered if n_i is a sensitive task or certain parent tasks n_p of the n_i didn't prepay $c(n_p)$ ($c(n_p)$ means that the cost for the n_p outbound communication transmission). Otherwise, both private computation resources ψ and public computation cloud resources Y can be used. For task n_i, it sorts n_i's immediate predecessors by their finish time increasingly for data communication allocation (Line 6). Line 7 calculates the cost $c(n_i)$ and makes a prepayment. Prepayment is to consider the worst case, that is, the budget is not enough to use the public cloud resources to execute the n_i's subtasks. The purpose is to have enough transmission budget to transfer data between n_i and its subtasks to the private cloud, thus ensuring that the entire workflow has a feasible solution. It is worth noting that this is divided into three scenarios. 1) $\beta \geq c(n_i)$: directly prepay; 2) $\beta < c(n_i)$ but amount converted from unused instances can prepay the balance: these instances are converted into corresponding amounts and added to β, and the instances are removed from *candidates* and Y; 3) other situations: only the private cloud resources are used as *candidates* (i.e., remove Y from *candidates*).

For each task n_i, we keep two sets of parameters, 1) the minimum $t_\Delta(n_i, v_l)$ of the task on all instances, and 2) the minimum $t_\Delta(n_i, v_l)$ of the task on instance that assigning n_i to them does not change the execution cost of the instance.

Next, each candidate v_l of n_i is traversed to search for the best one (Lines 8–27). For each n_i's immediate predecessor n_j, TDS checks whether n_j is deployed in a different platform from v_l, and if so, the cross-cloud data communication $e_{j,i}$ is tentatively allocated to b_{ext} (Lines 11–13). In fact, if $p(n_j)$ is public and $p(v_l)$ is private, the downstream channel is used and otherwise the upstream one is used.

$t_o(n_i, v_l)$ can be acquired with an insertion-based policy based on data communication allocation (Line 15): n_i is inserted between already scheduled tasks in v_l if the idle time slot between them is enough for executing n_i, and otherwise, it is allocated after the last task on v_l. Accordingly, n_i's finish time can be obtained. TDS not only considers the finish time of n_i, but also the potential cross-cloud data communications from n_i. Considering that sensitive tasks can only be allocated to the private cloud and it is better to allocate insensitive ones to the public cloud for faster processing, the potential cross-cloud data transmission time is estimated as,

$$t_\Delta(n_i, v_l) = t_\bullet(n_i, v_l) + \begin{cases} \sum\limits_{n_j \in succ(n_i) \cap O} d_{i,j}/b_{ext}, \text{ if } v_l \in R \\ \sum\limits_{n_j \in succ(n_i) \cap S} d_{i,j}/b_{ext}, \text{ if } v_l \in U \end{cases} \tag{12}$$

Hence, if allocating n_i onto v_l achieves a lower value of $t_\Delta(n_i, v_l)$, the current *best* value is updated to it. Meanwhile, the above temporary data communication allocation operations are recorded as T_1, and the task allocation operation $(n_i, v_l, t_o(n_i), t_\bullet(n_i))$ is recorded as T_2 (Lines 18–19). In line 21–25, the minimum $t_\Delta(n_i, v_l)$ on the instance for which adding n_i does not change the cost. After that, the tentative operations in this task iteration are all withdrawn.

After the traverse of *candidates* for n_i, TDS selects the best one for executing n_i which minimizes $t_\Delta(n_i, v_l)$, we need to make a decision about the scheduling of the task with respect to the value of T_2 and T_2^* (Line 28). The case $T_2 = T_2^*$ implies that assigning the task to the instance that yields the minimum $t_\Delta(n_i, v_l)$ of the task does not increase the cost of running the instance. In this case, it easily follows T_1 to schedule data communications, and follows T_2 to schedule n_i. However, when $T_2 \neq T_2^*$, follows T_1 and T_2 to schedule task and edge could result in a budget violation since RP already used all the budget to take as much as instance that it affords. Indeed, RP assumes that each of the taken instances will run only for maximum one hour. Hence, running an instance for more than one hour can lead to a budget violation. Removing this instance ensures that the algorithm remains within budget constraints. However, assigning the task to this instance increases the running cost (as it needs to operate for an extra hour). To offset this cost increase, another instance of the same type, located in Y, is also removed.

Figure 2 provides an illustrative example of this scenario. The earliest finish time of the n_i, exceeds a full-hour regardless of the type of instance selected. There is an unused instance with the same type as instance 1, the unused instance 2 is removed, thereby the instance 1 can run for an extra hour. If there is no such an unused instance, TDS follows T_1^* and T_2^* rather than T_1 and T_2 to schedule related data to avoid budget violation. Follow T_1^* and T_2^* to schedule the task and edges may not be as good as following T_1 and T_2, but this avoids the risk of a budget violation. The worst scenario is when there are not enough idle instances available in the public cloud. At this point, we can take

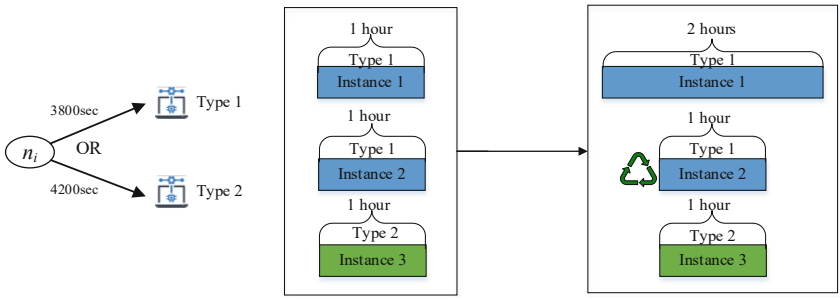

Fig. 2. An illustrative example showing the behavior of the TDS when the total execution time of the tasks hosted by an instance exceeds a full hour.

advantage of the almost free running of instances in the private cloud and deploy this task according to T_1^* and T_2^*.

Finally, iterating through all incoming edges of n_i, if they belong to the same cloud, recovering the prepaid amount associated with those edges (Line 29). The workflow returns the generated schedule after all tasks have been mapped.

5 Performance Evaluations

5.1 Experimental Settings

We select four different kinds of realistic scientific workflows here: Montage, LIGO, Cybershake and SIPHT. These workflows have different structures and can be divided into I/O intensive workflows and CPU intensive workflows. Specifically, Montage is an astronomy application used to generate custom mosaics of the sky, involving a large number of I/O intensive tasks; The CPU intensive tasks involved in the LIGO workflow, which analyzes data from coalescing compact binary systems to detect gravitational waves; Cybershake is used to describe earthquake hazards and is a data-intensive workflow that requires large resource and memory requirements; SIPHT is used to automate the search for sRNA encoding-genes for bacterial replicons which is primarily a CPU-bound workflow. For each application, Fig. 3 depicts the structure of relatively small workflows.

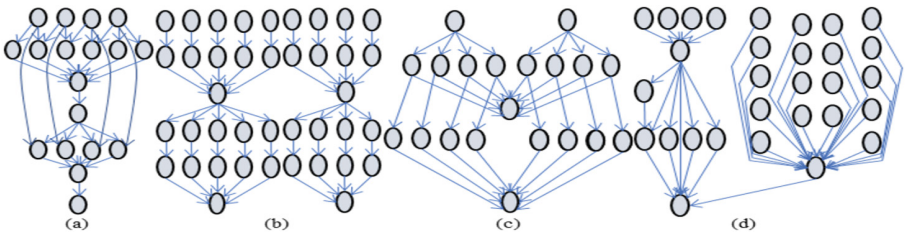

Fig. 3. Structure of scientific workflows. (a) Montage. (b) LIGO. (c) Cybershake. (d) SIPHT.

Table 1. Characteristics of Cloud Instances Used in Our Experiments

Type Id	Type	*CU*	Price ($)	Type Id	Type	*CU*	Price ($)
p_1	m1.small	1.7	0.06	p_5	m3.large	7.5	0.225
p_2	m1.medium	3.75	0.12	p_6	m1.xlarge	15	0.48
p_3	m3.medium	3.75	0.113	p_7	m3.xlarge	15	0.45
p_4	m1.large	7.5	0.24	p_8	m3.2xlarge	30	0.9

We choose two sizes for each workflow type: 50, 100 tasks. Moreover, tasks are divided into privacy-sensitive and privacy-insensitive ones: the former can only be deployed in a private cloud, whereas the latter can be deployed to a public or private cloud. We introduce a privacy factor β to represent the proportion of privacy-sensitive tasks in a workflow (i.e., $\beta = |S|/|N|$), and $\beta \times |N|$ tasks are randomly selected from the workflow and set to be privacy-sensitive. A larger β indicates more private tasks and it is set to 0.1 in default.

For the hybrid cloud platform, the internal bandwidth b_{int} is set to 1Gb/s, while the external one b_{ext} is set to 10Mb/s. As shown in Table 1, the public resources are conducted based on Amazon EC2 instances. The US East General-Purpose instance group with purchase-on-demand option is used. This is the same list of cases used in [12] to run experiment. Computation unit of private resources is 1. The characteristics of the instance types are provided in Table 1. And the number of available private computation resources is 4. Transferring data from the internet into Amazon EC2 is free of charge, so the cost of transferring data from a private cloud to a public cloud over the internet is disregarded. The cost of transferring data from a public cloud to a private cloud is $0.09/GB, with a monthly free quota of 100GB. The default budget is 1$. These parameters remain constant across the following experiments unless otherwise indicated for the purpose of highlighting their effects.

We conduct two sets of sensitivity experiments to evaluate the performance of our algorithms under the cost coefficient θ, the privacy coefficient β.

First, we vary the cost coefficient θ to simulate different complexities of budget constraints. A larger θ represents more instances can be purchased. Second, we vary the privacy coefficient β to simulate different privacy heterogeneities in hybrid cloud. A higher value of β means that more tasks in the workflow can only be performed in the slower private cloud.

We carefully choose and adapt the following representative scheduling techniques as baselines: (1) GRP-HEFT, a novel resource provisioning mechanism and a workflow scheduling algorithm; (2) PSO [11] an evolutionary computation technique called Particle Swarm Optimization (3) IPPTS [23], an improved predict priority task scheduling method to minimize the scheduling length. According to the experiments in [11], 160 iterations and 100 particles can result in the best makespan. For the parameter setting of PSO, the number of particles is set to 100, and the number of iterations is 160.

Table 2. Makespan results for scheduling methods

DAG	BCWS	GRP-HEFT	PSO	IPPTS
Montage 50	**30.44**	31.12	57.09	44.83
Montage 100	**45.49**	47.46	117.13	73.17
LIGO 50	**866.48**	876.85	1208.41	1082.46
LIGO 100	**912.11**	955.34	1869.41	1225.66
Cybershake 50	**140.00**	142.05	207.17	153.56
Cybershake 100	**194.26**	196.71	332.81	206.13
SIPHT 50	**1074.98**	1086.28	1442.11	2036.15
SIPHT 100	**1167.98**	1191.29	1865.90	1645.44

5.2 Results of Overall Experiments

Table 2 lists the average makespan results for all methods, where the best value for each workflow is highlighted in bold. We can observe that for each workflow case, BCWS achieves the best makespan. For example, the makespan of BCWS for LIGO 100 is 912.11 whereas those of the others are all greater than 950. The performance gap between them becomes larger when the task number in a workflow rises. A crucial aspect is that BCWS takes into account the potential data communication contention while considering the task scheduling to instances.

Figure 4 shows the results obtained by comparing the algorithms under different cost coefficient θ. Here, we vary θ from 0.5$ to 4$ with a step of 0.5$ for performance evaluation. We have the following observation that the completion time is decreasing as θ is increasing. This is because a larger value means that more in-stances can be purchased. When there are more instances to choose from, makespan is gradually decreasing. For example, the decrease in the time span is most noticeable when the amount increases from 0.5$ to 1.0$. The reason is that 0.5$ can only purchase a small number of better instances. When the budget exceeds 1$, there are numerous available instances, and at this point, the downward trend in makespan slows down. Compared to other algorithms, BCWS consistently maintains the best performance.

Figure 5 shows the results obtained by comparing the algorithms under different privacy coefficient β. Here, we vary β from 0.05 to 0.4 with a step of 0.05 for performance evaluation. We can observe that the makespan of each method generally increases as the β value rises. This is because more tasks in the workflow can only be executed in slower private resources as the β value is higher. Although situations vary for different workflow types, BCWS performs the best among all in each case.

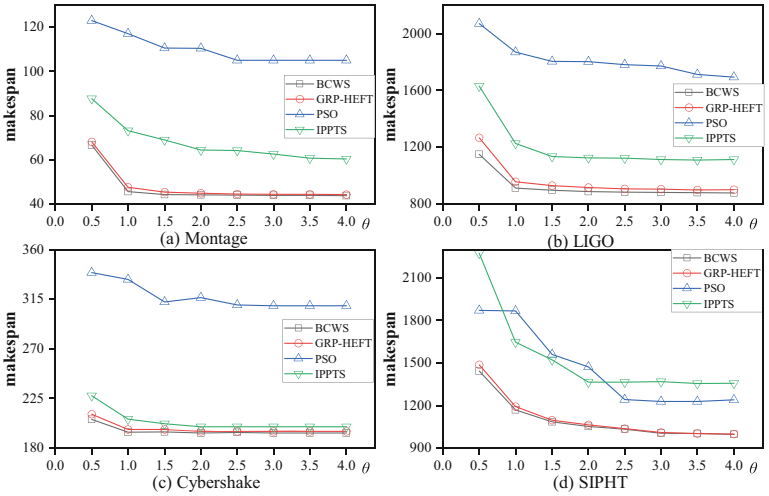

Fig. 4. Performance comparison with regards to cost factor θ

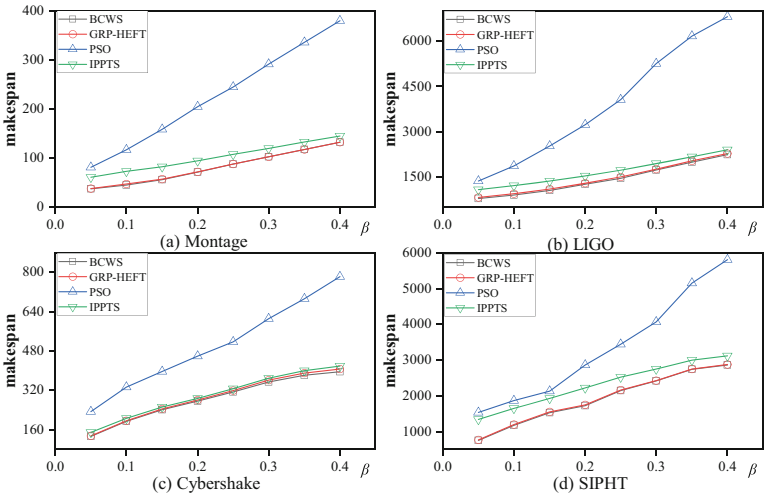

Fig. 5. Performance comparison with regards to privacy factor β

6 Conclusion

This paper effectively addresses the intricate challenges of effective scheduling and communication contention in hybrid cloud environments. The practical scheduling model and the Budget-Constraint Contention-Aware Workflow Scheduling (BCWS) algorithm offer robust solutions. Notably, the challenge of billing instances based on hourly rates is skillfully tackled. BCWS intelligently navigates this complexity by seamlessly integrating instance provisioning strategies and queuing mechanisms, optimizing both resource allocation and communication management. This strategic approach not only optimizes

workflow efficiency but also underscores the importance of cost-effectiveness in hybrid cloud deployments. The successful resolution of these challenges augments the overall system efficiency, enhances performance, and fosters stability within the hybrid cloud, ultimately ensuring smoother workflow execution across varied resource pools.

Acknowledgment. This work is supported in part by the National Natural Science Foundation of China under Grant 61702060 and 61672117.

References

1. Chen, W., Xie, G., Li, R., Bai, Y., Fan, C., Li, K.: Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems. Futur. Gener. Comput. Syst. **74**, 1–11 (2017)
2. Miller, G.: A hybrid attribute based access control model applied to data in a hybrid cloud environment. In: Proceedings of the 15th ACM International Conference on Systems and Storage, p. 150 (2022)
3. Azumah, K.K., Maciel, P.R.M., Sørensen, L.T., Kosta, S.: Modeling and simulating a process mining-influenced load-balancer for the hybrid cloud. IEEE Trans. Cloud Comput. **11**, 1999–2010 (2023)
4. Pasdar, A., Lee, Y.C., Almi'ani, K.: Hybrid scheduling for scientific workflows on hybrid clouds. Comput. Netw. **181**, 107438 (2020)
5. Zhu, J., Li, X., Ruiz, R., Xu, X.: Scheduling stochastic multi-stage jobs to elastic hybrid cloud resources. IEEE Trans. Parallel Distrib. Syst. **29**, 1401–1415 (2018)
6. Sakellariou, R., Zhao, H., Tsiakkouri, E., Dikaiakos, M.D.: Scheduling workflows with budget constraints. In: Gorlatch, S., Danelutto, M. (eds.) Integrated Research in GRID Computing, pp. 189–202. Springer, Boston (2007). https://doi.org/10.1007/978-0-387-47658-2_14
7. Yu, J., Buyya, R.: A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. In: 2006 Workshop on Workflows in Support of Large-Scale Science, pp. 1–10. IEEE (2006)
8. Zheng, W., Sakellariou, R.: Budget-deadline constrained workflow planning for admission control in market-oriented environments. In: Vanmechelen, K., Altmann, J., Rana, O.F. (eds.) Economics of Grids, Clouds, Systems, and Services, pp. 105–119. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28675-9_8
9. Farahabady, M.R.H., Lee, Y.C., Zomaya, A.Y.: Pareto-optimal cloud bursting. IEEE Trans. Parallel Distrib. Syst. **25**, 2670–2682 (2014)
10. Lin, X., Wu, C.Q.: On scientific workflow scheduling in clouds under budget constraint. In: 2013 42nd International Conference on Parallel Processing, pp. 90–99. IEEE (2013)
11. Wang, X., Cao, B., Hou, C., Xiong, L., Fan, J.: Scheduling budget constrained cloud workflows with particle swarm optimization. In: 2015 IEEE Conference on Collaboration and Internet Computing (CIC), pp. 219–226 (2015)
12. Faragardi, H.R., Sedghpour, M.R.S., Fazliahmadi, S., Fahringer, T., Rasouli, N.: GRP-HEFT: a budget-constrained resource provisioning scheme for workflow scheduling in IaaS clouds. IEEE Trans. Parallel Distrib. Syst. **31**, 1239–1254 (2020)
13. Sinnen, O., To, A., Kaur, M.: Contention-aware scheduling with task duplication. J. Parallel Distrib. Comput. **71**, 77–86 (2011)
14. Benoit, A., Hakem, M., Robert, Y.: Contention awareness and fault-tolerant scheduling for precedence constrained tasks in heterogeneous systems. Parallel Comput. **35**, 83–108 (2009)

15. Özkaya, M.Y., Benoit, A., Uçar, B., Herrmann, J., Catalyürek, Ü.V.: A scalable clustering-based task scheduler for homogeneous processors using DAG partitioning. In: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 155–165 (2019)
16. Genez, T.A., Bittencourt, L.F., da Fonseca, N.L., Madeira, E.R.: Estimation of the available bandwidth in inter-cloud links for task scheduling in hybrid clouds. IEEE Trans. Cloud Comput. **7**, 62–74 (2019)
17. Son, J., Buyya, R.: Priority-aware VM allocation and network bandwidth provisioning in software-defined networking (SDN)-enabled clouds. IEEE Trans. Sustain. Comput. **4**, 17–28 (2018)
18. Wu, Q., Zhou, M., Wen, J.: Endpoint communication contention-aware cloud workflow scheduling. IEEE Trans. Autom. Sci. Eng. **19**, 1137–1150 (2022)
19. Mithila, S.P., Baumgartner, G.: Latency-based Vector Scheduling of Many-task Applications for a Hybrid Cloud. In: 2022 IEEE 15th International Conference on Cloud Computing (CLOUD), pp. 257–262 (2022)
20. Durillo, J.J., Prodan, R.: Multi-objective workflow scheduling in Amazon EC2. Clust. Comput. **17**, 169–189 (2014)
21. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Futur. Gener. Comput. Syst. **29**, 682–692 (2013)
22. Chen, Z.G., et al.: Multiobjective cloud workflow scheduling: a multiple populations ant colony system approach. IEEE Trans. Cybern. **49**, 2912–2926 (2019)
23. Djigal, H., Feng, J., Lu, J., Ge, J.: IPPTS: an efficient algorithm for scientific workflow scheduling in heterogeneous computing systems. IEEE Trans. Parallel Distrib. Syst. **32**, 1057–1071 (2021)
24. Lei, J., Wu, Q., Xu, J.: Privacy and security-aware workflow scheduling in a hybrid cloud. Futur. Gener. Comput. Syst. **131**, 269–278 (2022)