Received November 9, 2020, accepted November 22, 2020, date of publication November 26, 2020, date of current version January 6, 2021.

Digital Object Identifier 10.1109/ACCESS.2020.3040719

Scheduling Spark Tasks With Data Skew and Deadline Constraints

HAIHUA GU[1], XIAOPING LI[2,3], (Senior Member, IEEE), AND ZHIPENG LU[2,3]
[1]School of Artificial Intelligence, Nanjing Vocational College of Information Technology, Nanjing 210023, China
[2]School of Computer Science and Engineering, Southeast University, Nanjing 211189, China
[3]Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing 211189, China

Corresponding author: Xiaoping Li (xpli@seu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1402500, in part by the National Natural Science Foundation of China under Grant 61872077 and Grant 61832004, in part by the National Hi-Tech Project under Grant 315055101, in part by the project of advanced research of the leading professional teachers in Higher Vocational Colleges in Jiangsu Province under Grant 2019GRFX078, and in part by the Collaborative Innovation Center of Wireless Communications Technology.

ABSTRACT Data skew has an essential impact on the performance of big data processing. Spark task scheduling with data skew and deadline constraints is considered to minimize the total rental cost in this paper. A modified scheduling architecture is developed in terms of the unique characteristics of the considered problem. A mathematical model is constructed, and a Spark task scheduling algorithm is proposed considering both the data skew and deadline constraints. The algorithm consists of three components: stage sequencing, task scheduling, and scheduling adjustment. Strategies for each of the components are presented. The parameters and components of the proposed algorithm are calibrated over many random instances. The calibrated algorithm is compared to two existing algorithms for similar problems over classical scientific workflow applications. Experimental results show that the proposed algorithm outperforms the compared algorithms statistically.

INDEX TERMS Data skew, spark, scheduling optimization, cloud computing.

I. INTRODUCTION

Deploying Spark applications on public clouds has become increasingly popular for customers to efficiently process big data or large quantities of data, mainly from social media, e-commerce platforms, and search engines. Generally, these data are incredibly unevenly distributed, i.e., data provided by different customers have different sizes, and their processing times are significantly distinct. If the processing times of some data are much longer than those of the other data, data skew has resulted, which might seriously degrade the performance of a Spark system in cloud computing.

The most significant characteristic of these data is the extremely uneven distribution of data, such as popular social media topics, popular products on e-commerce platforms, and search engines.

This paper considers the Spark task scheduling problem to minimize the total rental cost in a public cloud. Cloud nodes are heterogeneous and rented in an on-demand manner. The Spark application jobs are precedence constrained, which are depicted by a DAG (directed acyclic graph). Every job contains many DAG-constrained stages, and there are some independent tasks in each stage. Tasks are data skewed. The Spark application is deadline constrained. There are two main challenges for the considered problem: (i) The optimized objective is greatly influenced by the sequence of all the tasks. Generally, the optimal task order is hard to obtain for DAG applications. However, there are two levels of DAGs. The stage DAG precedence constraints are embedded into those of jobs, making it difficult to obtain an optimal topological task order. (ii) The skewed data lead to incredibly varying task processing times.

For the above challenges, the main contributions of this paper are as follows: (i) The considered problem is mathematically modeled using integer programming with heterogeneous public cloud nodes, data skew, and the deadline constraint. (ii) SF (skew factor) is used to measure the data skew of tasks. CoV (coefficient of variation) is adopted at the stage level to measure the variation in tasks. (iii) A task scheduling algorithm is proposed for tasks in a Spark application with data skew and deadline in a public cloud to minimize the total rental cost. There are three

The associate editor coordinating the review of this manuscript and approving it for publication was Sun-Yuan Hsieh.

algorithmic components in the proposed algorithm: stage sequencing, task scheduling, and scheduling adjustment. An algorithm used a reservoir algorithm to sample the distribution of key data values and labeled the nodes with heavy loads before running. Workloads of massive nodes were offloaded to other nodes to maintain the data balance in the reduction stage.

The rest of the paper is organized as follows. Related works are reviewed in Section II. Section III describes and formalizes the problem under study. A heuristic algorithm is proposed for the considered problem in Section IV. Section V evaluates the performance of the proposal under different workload scenarios followed by conclusions in Section VI.

II. RELATED WORK

Data skew is important for the performance of processing big data. Mohammad *et al.* [1] surveyed data skew from 2010 to 2017. They divided the data skew problem into four types: map stage, reduce stage, map & reduce stage, and shuffle stage, according to data skew. They divided the reduction phase into the sampling-based method and the nonsampling-based method. Qi *et al.* [2] proposed the LIBRA algorithm, which uses a novel sampling method to alleviate lightweight data skews. A highly accurate approximation of the intermediate data distribution was achieved by sampling only a small part of the intermediate data during map processing. Liu *et al.* [3] proposed a partitioning algorithm for data skew in Spark streaming's reduce stage. The data are regarded as candidate samples. The characteristics of intermediate data are predicted by selected samples based on sampling.

Task scheduling problems for big data have been studied in recent years. The most concerning constraints are deadlines and budgets, and the commonly considered objectives are makespan, the total rental cost, and energy minimization. To minimize the VM rental cost, Cai *et al.* [4] proposed a heuristic algorithm based on multiple priority rules. Zeng *et al.* [5] proposed a greedy-based MapReduce application scheduling algorithm to minimize the total rental cost of cloud resources while meeting the SLA (service level agreements) level with deadline constraints. To meet the deadline requirements in a hybrid cloud, Michael *et al.* [6] proposed a strategy for dynamically configuring public cloud resources. Tasks were concurrently executed to accelerate MapReduce. Zacheilas *et al.* [7] considered the balance between performance and budget. A Pareto-based scheduler was constructed to minimize completion time without exceeding user budgets. The scheduler systematically searches and detects the allocation of slots. Recently, attention has been on Spark task scheduling from different perspectives. Cheng *et al.* [8] designed a cross-platform middleware for resource scheduling on Hadoop and Spark platforms that can improve resource utilization and performance on multitenant Spark-on-YARN clusters. Liu *et al.* [9] proposed a framework that facilitates submitting, monitoring, and executing Spark workflows in heterogeneous environments. For multijob and multicluster scheduling, they used a greedy strategy to obtain an initial solution and a simulated annealing algorithm to search the globally optimal solution.

As mentioned above, data skew has been studied in MapReduce task scheduling. Although task scheduling has been widely studied, little attention has been on Spark task scheduling with data skew. To the best of our knowledge, this problem has not yet been studied. Existing scheduling architectures are not suitable for the problem considered in this paper.

III. PROBLEM DESCRIPTION

By considering the data skew and deadline constraints, a modified system architecture is developed from the traditional architecture, as shown in Figure 1. Three roles are included in the system architecture: user, broker, and cloud provider. There are four modules in the proposed architecture: stage dividing, task scheduler, resource pool, and scheduling adjustment. Stage dividing divides a Spark application into jobs and stages according to the embedded precedence constraints. Their priorities queue all stages. Task scheduler sorts tasks of stages and assigns the tasks to VMs (virtual machines) of the resource pool sequentially. The resource pool is responsible for renting and releasing VMs from the cloud provider with the on-demand pricing model. Scheduling adjustment manages to adjust the task schedule and the resource pool to minimize the Spark application's total rental cost without exceeding the deadline.

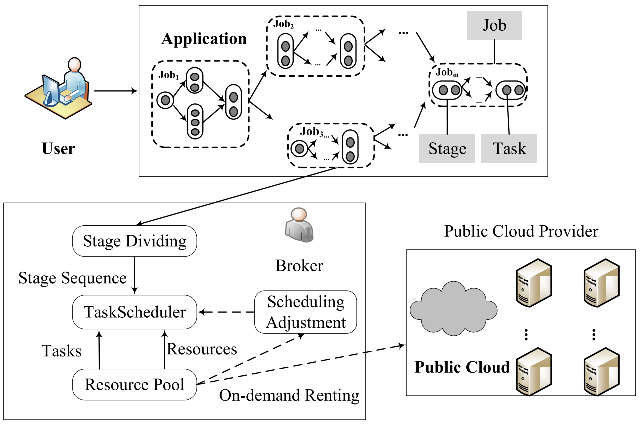

FIGURE 1. A system architecture for Spark.

A Spark workflow application consists of a set of precedence constrained jobs depicted by a job-DAG. Each job is composed of a set of precedence constrained stages, which are represented by a stage DAG. A stage-DAG is embedded into a vertex of the job-DAG. Each stage consists of several independent tasks that can be processed in parallel. The schematic architecture of a Spark application is depicted in Figure 2. We make the following assumptions for the problem considered in this paper: (i) The bandwidths are the same in the public cloud regardless of the network environment. (ii) One node processes only one task, and one task is processed only by one node at a time. Tasks are

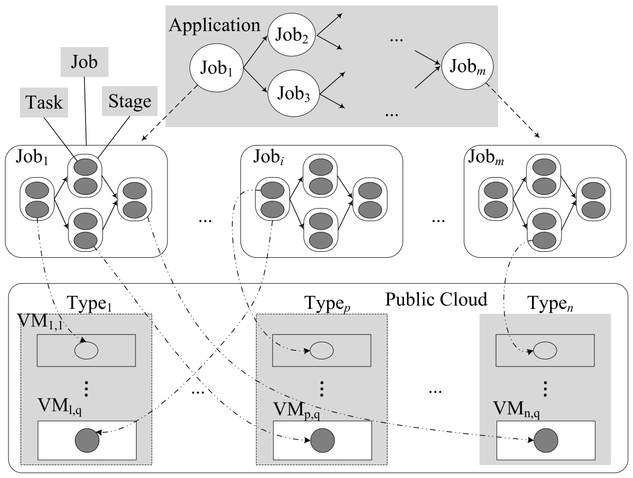

FIGURE 2. Schematic architecture of the Spark system in a public cloud.

nonpreemptive during execution. (iii) VM setup times are neglected. (iv) Heterogeneity of VMs implies their difference in processing speeds. Notations to be used in the following are listed in Table 1.

TABLE 1. Notations to be used.

Notation	Definition
m	number of jobs in the application
m_i	number of stages in the i^{th} job
S_k	the k^{th} stage
w_k	number of tasks in S_k
$T_{k,l}$	the k^{th} task in S_k, $l \in \{1, \ldots, w_k\}$
B_k	start time of S_k
F_k	finish time of S_k
$b_{k,l}$	start time of $T_{k,l}$
$f_{k,l}$	finish time of $T_{k,l}$
Z_k^P	set of immediate predecessors of S_k
Z_k^S	set of immediate successors of S_k
$Data_{k,l}$	data volume of $T_{k,l}$ needed to process
$\mathrm{VM_{pool}}$	VM pool
a	total number of VMs in the VM pool
n	total number of VM types
a_i	number of the i^{th} type of VM, $i \in \{1, \ldots, \nu\}$
v_p	process speed of the i^{th} type of VM, $i \in \{1, \ldots, \nu\}$
$VM_{p,q}$	the q^{th} VM of the p^{th} type
P_p	unit price of the p^{th} type of VM
$ECT_{k,l}$	execution time of $T_{k,l}$
$\tau_{k,l}$	transmission time of $T_{k,l}$
$L_{p,q}$	task list of $VM_{p,q}$
$AT_{p,q}$	available time of $VM_{p,q}$
BTU	VM billing time unit
Cov_k	coefficient of variation S_k
$SF_{k,l}$	skewed factor of $T_{k,l}$
$\overline{Data_k}$	mean data size of S_k
$Std_{i,j}$	standard deviation of data size of $S_{i,j}$
bw	the bandwidth in a public cloud

Since stage-DAGs are embedded in a Spark application's job-DAG, we depict the whole Spark application as "big" stage-DAG $G = (S, E)$. Let the number of jobs be m and the i^{th} job contains μ_i stages, i.e., there are $|S| = \sum_{i=1}^{m} \mu_i$ vertices in G. Stage $S_k = \{T_{k,1}, \ldots, T_{k,w_k}\}$ contains w_k tasks that can be executed in parallel. The start and finish times of V_k are B_k and F_k, respectively. The start and finish times of $T_{k,l}$ are $b_{k,l}$ and $f_{k,l}$, respectively. The Spark application deadline is D, i.e., all tasks of the application must finish before D. Z_k^P and Z_k^S are the sets of immediate predecessors and immediate successors of V_k.

In the considered situation, we assume that there are n types of VM. The p^{th} type contains a_p VMs with the process speed v_p. The resource pool is denoted as $\mathrm{VM_{pool}} = \{VM_{p,q} | p \in \{1, \ldots, n\}, q \in \{1, \ldots, a_p\}\}$. The total number of VMs $a = \sum_{p=1}^{n} a_p$. The decision variable $x_{k,l;p,q} \in \{0, 1\}$ takes 1 if task $T_{k,l}$ is scheduled to VM $VM_{p,q}$ and 0 otherwise. The execution time and transmission time of $T_{k,l}$ are $ECT_{k,l}$ and $\tau_{k,l}$, respectively. The data size of $T_{k,l}$ is $Data_{k,l}$. P_p is the unit price of the p^{th} type VM. The bandwidth of the public cloud is bw.

$$\min \ Cost = \sum_{p=1}^{n} \sum_{q=1}^{a_p} P_p \left\lceil \frac{\sum_{k=1}^{w} \sum_{l=1}^{w_k} x_{k,l;p,q}(ECT_{k,l} + \tau_{k,l})}{BTU} \right\rceil \tag{1}$$

$$\text{s.t. } \tau_{k,l} = \sum_{p=1}^{n} \sum_{q=1}^{a_p} x_{k,l;p,q} \times \frac{Data_{k,l}}{bw} \tag{2}$$

$$ECT_{k,l} = \sum_{p=1}^{n} \sum_{q=1}^{a_p} x_{k,l;p,q} \times \frac{Data_{k,l}}{v_p} \tag{3}$$

$$\max_{k=1,\ldots,w} F_k \leqslant D \tag{4}$$

$$b_{k,1} = B_k \tag{5}$$

$$F_k = \max_{l=1,\ldots,w_k} f_{k,l} \tag{6}$$

$$f_{k,l} = b_{k,l} + \tau_{k,l} + ECT_{k,l} \tag{7}$$

$$F_k \leqslant \min_{S_{k'} \in Z_k^S} B_{k'} \tag{8}$$

$$B_k \geqslant \max_{S_{k'} \in Z_k^P} F_{k'} \tag{9}$$

$$B_1 = 0 \tag{10}$$

$$\sum_{k=1}^{w} \sum_{l=1}^{w_k} x_{k,l;p,q} = 1 \tag{11}$$

$$x_{k,l;p,q} \in \{0, 1\} \tag{12}$$

where $k = 1, \ldots, w$; $l = 1, \ldots, w_k$; $p = 1, \ldots, n$; $q = 1, \ldots, a_p$.

Eqn. (1) calculates the optimization objective, which is mainly determined by the processing and transmission times of tasks. Eqn. (2) calculates the data transmission time of a task. Eqn. (3) computes the processing time of task $T_{k,l}$. Formula (4) indicates that the completion time of all stages cannot exceed the deadline of the Spark application. Eqn. (5) indicates that the start time of a task is set as that of the stage to which it belongs. Eqn. (6) indicates that the completion time of S_k is determined by the latest completion time of tasks in S_k. Eqn. (7) means the completion time of task $T_{k,l}$ is the sum of its start time, processing time, and transmission time. Eqn. (8) and Eqn. (9) represents the precedence constrained relationship between stages. It is assumed that the application arrives at time zero, which is indicated in Eqn. (10). Eqn. (11) and Eqn. (12) ensures that each task can be assigned only to one virtual machine at a time.

IV. PROPOSED ALGORITHMS

Although there are some existing algorithms for scheduling MapReduce tasks and Spark tasks, no existing algorithm can be directly used for the considered problem with both data skew and deadline constraints. In this paper, we propose the STSDD (Spark task scheduling with data skew and deadline) algorithm for the considered problem.

A. SKEWNESS MEASUREMENT

Since data skew is a qualitative term, it needs to be specified or measured for task scheduling. Similar to Zhuo *et al.* [10], CoV (coefficient of variation) is used to measure skewness quantitatively. The stage skewness of S_k is measured by Cov_k, which is the ratio of the standard data volume deviation to the average data volumes of the tasks in S_k as in Eqn. (13). In Eqn. (14), Std_k is the standard deviation of the data required to be processed by all tasks in S_k. Eqn. (15) represents the average data volume required by the task set. A larger Cov_k implies a higher skewness of S_k, and vice versa.

$$Cov_k = \frac{Std_k}{\overline{Data_k}} \tag{13}$$

$$Std_k = \sqrt{\frac{\sum_{l=1}^{w_k}(Data_{k,l} - \overline{Data_k})^2}{w_k}} \tag{14}$$

$$\overline{Data_k} = \frac{\sum_{l=1}^{w_k} Data_{k,l}}{w_k} \tag{15}$$

Similarly, the skewness of task $T_{k,l}$ is measured by the SF (skew factor) $FS_{k,l}$ defined in Eqn. (16). $FS_{k,l}$ is limited within the upper bound UB and the lower bound LB. If $SF_{k,l}$ is greater than UB, $T_{k,l}$ is regarded as a skewed task. If it is less than the lower bound LB, $T_{k,l}$ is regarded as a small task.

$$FS_{k,l} = \frac{Data_{k,l} - \overline{Data_k}}{\overline{Data_k}} \tag{16}$$

B. PROPOSED ALGORITHM FRAMEWORK

For the problem under study, we propose an STSDD (Spark task scheduling with data skew and deadline) framework. The skewness of stages and tasks are calculated. Parameters are initialized. STSDD mainly consists of the following algorithm components: (i) Stages are sequenced by an SPQ (stage priority queue). (ii) Tasks of each stage are scheduled by task scheduling. (iii) The obtained solutions are iteratively improved. The proposed STSDD algorithm is formally described in Algorithm 1.

C. STAGE SEQUENCING

During scheduling, stages are processed one by one with some order. Because of the precedence constraints, stages of the application cannot start at the same time. We use the stage priority queue SPQ to keep the currently ready stages, i.e., stages in SPQ dynamically change during scheduling. Stages in SPQ are managed by a heap structure (a complete binary tree). If a stage S_k is scheduled, all of its immediate successors are checked to be ready or not. Any ready immediate successors are inserted to SPQ. The stage sequence is

Algorithm 1 STSDD (Spark Task Scheduling With Data Skew and deadline)

```
Input: Spark application $G$; deadline $D$;
Output: The rental cost $Cost$
begin
    for $k = 1$ to $m$ do
        for $l = 1$ to $m_l$ do
            $B_{k,l} \leftarrow 0$;
    $VM_{pool} \leftarrow \emptyset$;
    Generate a stage topological order;
    Create the stage priority queue $SPQ$ based on a stage
    sorting strategy;
    while $SPQ \neq \emptyset$ do
        Pop $S_k$ from $SPQ$;
        foreach $T_{k,l} \in S_k$ do
            Call TSA; /* Task Scheduling
                Algorithm                    */
        Update $b_{k,l}$ and $f_{k,l}$ of $S_k$;
        Add new ready stages to $SPQ$;
    Call SAA; /* Scheduling Adjustment
        Algorithm                            */
    return $Cost$.
```

crucial to the scheduling performance, i.e., how to construct the SPQ heap is important. Based on the stage skewness, four-stage sequencing strategies are developed in this paper: HSF (heaviest skew first), LSRF (largest skew rate first), LDF (largest data first), and RAND (RANDOM).

(1) HSF (heaviest skew first): All ready stages in SPQ are sorted by a nonincreasing order of CoVs (coefficient of variation) defined in Eqn. (13), i.e., stages in SPQ are organized using the max-heap.

(2) LSRF (largest skew rate and skew first): All ready stages in SPQ are sorted by a nonincreasing order of skew rates defined in Eqn.(17) where st_k is the number of skewed tasks in S_k with skew factors greater than UB. w_k is the number of tasks in S_k. If the skew rates are equal, the tie is broken by the descending order of CoVs. Stages in SPQ are organized using the max-heap.

$$SR_k = \frac{st_k}{w_k} \tag{17}$$

(3) LDF (largest data first): All ready stages in SPQ are sorted by a nonincreasing order of the total quantity of data defined in Eqn. (18), i.e., stages in SPQ are organized using the max-heap.

$$D_k = \sum_{l=1}^{w_k} D_{k,l} \tag{18}$$

(4) RAND: Stages are sorted randomly.

To illustrate the four strategies' distinctions, an example with five stages is shown in Figure 3. The numbers in the circles are the data sizes to be processed by the tasks. The

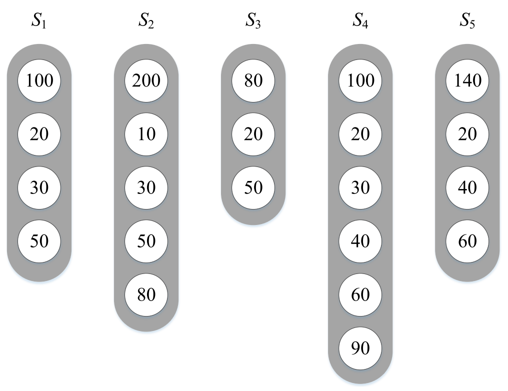

FIGURE 3. Stage sequencing.

TABLE 2. Stage parameter calculation results.

Stage	Coefficient of variation	Skew rate	Data size (MB)
S_1	0.62	0.25	200
S_2	0.91	0.2	370
S_3	0.49	0.33	150
S_4	0.53	0.33	340
S_5	0.70	0.25	260

corresponding measurements (CoVs, skew rates, and data sizes) are shown in Table 2. According to the above strategies, the stage sequences in *SPQ* are HSF: $S_2 \rightarrow S_5 \rightarrow S_1 \rightarrow S_4 \rightarrow S_3$; LSRF: $S_4 \rightarrow S_3 \rightarrow S_5 \rightarrow S_1 \rightarrow S_2$); LDF: ($S_2 \rightarrow S_4 \rightarrow S_5 \rightarrow S_1 \rightarrow S_3$. Different stage orders are obtained through different strategies.

D. TASK SCHEDULING

Because of heterogeneity, VMs have different execution speeds, meaning that the processing times of tasks cannot be determined in advance. Since all tasks' data sizes are assumed to be fixed, it is necessary to estimate the average processing speed of all VMs before estimating the temporal parameter. In this paper, we define the average processing speed of all VM types as:

$$\widehat{v} = \frac{\sum_{p=1}^{n} v_p}{n}. \tag{19}$$

The processing time of $T_{k,l}$ includes two parts: the data processing time and the data transmission time, as shown in Eqn. (20). $\frac{Data_{k,l}}{\widehat{v}}$ is the data processing time, and $\frac{Data_{k,l}}{bw}$ denotes the data transmission time.

$$\widehat{Time}_{k,l} = \frac{Data_{k,l}}{\widehat{v}} + \frac{Data_{k,l}}{bw}. \tag{20}$$

Since all tasks in S_k can be executed in parallel, the processing time of S_k is determined by the latest finishing task:

$$\widehat{Time}_k = \max_{l=1,\dots,w_k} \widehat{Time}_{k,l}. \tag{21}$$

The latest start and finish times of S_k can be estimated by Eqn.(22) and Eqn.(23).

$$\widehat{LST}_k = \widehat{LFT}_k - \widehat{Time}_k. \tag{22}$$

$$\widehat{LFT}_k = \begin{cases} D, & \text{if} Z_k^S == \emptyset \\ \min\limits_{S_{k'} \in Z_k^S} \widehat{LST}_{k'}, & \text{otherwise.} \end{cases} \tag{23}$$

(i) By comparing the calculated skew factor to the upper and lower bounds, UB and LB, all ready tasks are classified into three categories according to task skewness: skewed tasks, normal tasks, and small tasks. $UB > 0$ and $LB \leq 0$ are set by users. (ii) In terms of task skewness and the data volume, suitable VM types are selected by a strategy. (iii) Available VMs of each selected type are searched from VM_{pool} for the set of available virtual machine resources. (iv) Specific VMs are selected for the ready tasks by a VM selection method, i.e., the ready tasks are scheduled to VMs. The task scheduling process is shown in Algorithm 2.

Algorithm 2 TSA (Task Scheduling Algorithm)

```
Input: $T_{k,l}$
begin
    if ($FS_{k,l} \geqslant UB$ ) then
        $T_{k,l}$ is a skewed task ;
    else
        if ($FS_{k,l} \leqslant LB$ ) then
            $T_{k,l}$ is a small task ;
        else
            $T_{k,l}$ is a normal task ;
    Call VM type selection;
    Generate the available VM set $\pi_a$ by AVMSA;
    /* Available virtual machines
       searching algorithm                    */
    Select $VM_{p,q}$ from $\pi_a$ using the VM selection;
    Schedule $T_{k,l}$ to $VM_{p,q}$ ;
    Update $B_{k,l}$ and $F_{k,l}$ of $T_{k,l}$ ;
    Update $BT_{p,q}$ and $FT_{p,q}$ of $VM_{p,q}$ ;
    return.
```

1) TASK CLASSIFICATION

Tasks are classified into skewed tasks, small tasks, and normal tasks:

(i) A task is a skewed task if the skew factor calculated by Eqn. (16) is greater than *UB*.
(ii) A task is a small task if the skew factor is less than *LB*.
(iii) A task is a normal task if the skew factor locates between *LB* and *UB*.

2) VM TYPE SELECTION

We assume that the same types of VMs have the same speeds, which means that the processing time is identical for VMs of the same type. In terms of data skewness and the classified tasks, four strategies are introduced for VM type selection:

(1) STF (skewed task first): The VM type with the fastest speeds are selected for skewed tasks.
(2) LTF (largest task first): The VM type with the fastest speeds are selected for tasks with the largest data size.

TABLE 3. Task related parameter calculation.

Task	size of data (MB)	Skew factor	Classification
$T_{1,1}$	100	0.67	skewed task
$T_{1,2}$	50	-0.17	normal task
$T_{1,3}$	30	-0.5	small task
$T_{2,1}$	30	-0.88	small task
$T_{2,2}$	70	-0.72	small task
$T_{2,3}$	500	1	skewed task
$T_{2,4}$	400	0.6	skewed task
$T_{3,1}$	240	0.44	normal task
$T_{3,2}$	230	-0.38	normal task
$T_{3,3}$	30	-0.82	small task

(3) SLTF (skewed and largest task first): The VM types with the fastest speeds are selected for skewed tasks and tasks with the largest data size.
(4) RAND (RANDOM): A type of VM is randomly selected for a task.

To demonstrate the effects of the above strategies, an example with the skewed upper bound $UB = 0.5$ and the skewed lower bound $LB = -0.5$ is shown in Table 3. If the STF strategy is used, $T_{1,1}$, $T_{2,3}$ and $T_{2,4}$ select the type 1 VM. $T_{1,2}$, $T_{3,1}$ and $T_{3,2}$ select the type 2 VM. $T_{1,3}$, $T_{2,1}$, $T_{2,2}$ and $T_{3,3}$ select the type 3 VM. If the LTF strategy is used: $T_{1,1}$, $T_{2,3}$ and $T_{3,1}$ select the type 1 VM. $T_{1,2}$, $T_{2,4}$, $T_{3,1}$ and $T_{3,2}$ select the type 2 VM. $T_{1,3}$, $T_{2,1}$, $T_{2,2}$ and $T_{3,3}$ select the type 3 VM. If the SLTF strategy is used: $T_{1,1}$ and $T_{2,3}$ select the type 1 VM. $T_{1,2}$, $T_{2,1}$, $T_{2,2}$, $T_{2,4}$, $T_{3,1}$ and $T_{3,2}$ select the type 2 VM. $T_{1,3}$, $T_{2,2}$ and $T_{3,3}$ select the type 3 VM.

3) AVAILABLE VIRTUAL MACHINES SEARCHING

An available VM set π_a for task $T_{k,l}$ is constructed by searching all available VMs for $T_{k,l}$ from the current resource pool in terms of the start time of $T_{k,l}$ and the required VM type $Type_{k,l}$. If no available VM exists, i.e., $\pi_a = \emptyset$, a new VM with the time slot required by $T_{k,l}$ is rented from the service provider and added to π_a. The process is shown in Algorithm 3.

4) VIRTUAL MACHINE SELECTION

A specific VM is selected from the available resource set π_a for the current task. In this paper, three strategies are proposed for VM selection:

- EATF (earliest available time first): The VM with the earliest available time in π_a is selected.
- LRTIF (Longest remaining time interval first): Since VMs are leased by time units (e.g., hours) in an on-demand leasing manner, the VM with the longest remaining available time interval slice is selected from π_a.
- RAND(RANDOM): A VM is randomly selected from π_a.

Algorithm 3 AVMSA (Available Virtual Machines Searching Algorithm)

```
Input: $T_{k,l}$, $Type_{k,l}$ and VM$_{pool}$
Output: $\pi_a$
begin
    $\pi_a \leftarrow \emptyset$ ;
    if $VM_{pool} = \emptyset$ then
        Lease a $VM_{Type_{k,l},1}$ from the public cloud ;
        Add $VM_{Type_{k,l},1}$ to VM$_{pool}$ ;
    foreach $VM_{p,q} \in VM_{pool}$ do
        if $(p = Type_{k,l})$ AND $(AT_{p,q} \leqslant B_{k,l})$ then
            Add $VM_{p,q}$ to $\pi_a$ ;
    if $\pi_a = \emptyset$ then
        Lease a $VM_{Type_{k,l},1}$ from the public cloud ;
        Add $VM_{Type_{k,l},1}$ to VM$_{pool}$ ;
        Add $VM_{Type_{k,l},1}$ to $\pi_a$ ;
    return $\pi_a$.
```

E. SCHEDULING ADJUSTMENT

An initial scheduling solution is generated by task scheduling. However, there are many idle time slots on VMs that can be fully utilized by scheduling adjustments to reduce the total rental cost. In this paper, a scheduling adjustment algorithm is proposed that mainly consists of two parts: FTSM (fragmented time slot merging) and ITSF (idle time slot filling). The scheduling adjustment algorithm is formally depicted in Algorithm 4. It is not hard to determine that the SSA algorithm's time complexity is $O(w \times w_k \times a)$.

To illustrate the above process, examples are shown as follows. Figure 4 shows the FTSM process. The initial solution is shown in Figure 4(a), where tasks $T_{k,1}$, $T_{k,2}$, $T_{k,3}$, $T_{k,4}$, $T_{k,5}$ are on $VM_{3,2}$, $VM_{3,1}$, $VM_{1,1}$, $VM_{2,1}$, and $VM_{2,2}$, respectively. The current rental cost is \$ 2.975. By the FTSM process, a new solution is obtained, as shown in Figure 4(b): Tasks $T_{k,1}$ and $T_{k,2}$ are placed on $VM_{3,1}$, $T_{k,3}$ is placed on $VM_{1,1}$, tasks $T_{k,4}$ and $T_{k,5}$ are placed on $VM_{2,2}$. The rental cost is only \$ 2.72. Figure 5shows the ITSF process. The initial solution is shown in Figure 5(a), where tasks $T_{k,1}$, $T_{k,2}$, $T_{k,3}$, $T_{k,4}$, $T_{k,5}$ and $T_{k,6}$ are on $VM_{3,2}$, $VM_{3,1}$, $VM_{2,1}$, $VM_{2,1}$, $VM_{1,1}$ and $VM_{1,1}$, respectively. The current rental cost is \$ 2.295. Each duration includes two parts: transmission time and processing time. The gray rectangles indicate the processing times, and the dashed rectangles denote the transmission times of the considered tasks. By searching the idle slots between the same VM's two tasks, virtual machines $VM_{1,1}$ and $VM_{2,1}$ have idle slots. ITSF attempts to make full use of these idle slots by adjusting. $T_{k,1}$ is moved to $VM_{1,1}$ and $T_{k,2}$ is moved to $VM_{2,1}$. The total rental cost becomes \$ 2.04.

SIVL obtains a set of VMs with an idle slot list $List$, as shown in Algorithm 5. s_i and f_i refer to the start and finish times of the i^{th} task in $List$.

Algorithm 4 SAA (Scheduling Adjustment Algorithm)

```
Input: $G$, $VM_{pool}$
Output: $Cost$
begin
    for $k = 1$ to $w$ do    /* FTSM (fragmented time slot merging) */
        for $l = 1$ to $w_k$ do
            if $f_{k,l} \neq F_k$ then
                Search available time slots;
                Try to put tasks on the same type of VMs together without exceeding $F_k$;
    $List \leftarrow$ SIVL();    /* ITSF (idle time slot filling) */
    foreach $VM_{p,q} \in VM_{pool}$ do
        foreach $VM_{p',q'} \in List$ do
            Construct the task list $L_{p,q}$ from $VM_{p,q}$ ;
            foreach $T_{k,l} \in L_{p,q}$ do
                Try to fill the idle time in $VM_{p,q}$ ;
                if $T_{k,l}$ can fill the idle time then
                    Remove $T_{k,l}$ from $L_{p,q}$ ;
    Calculate $Cost$ by Eqn. (1);
    return $Cost$.
```

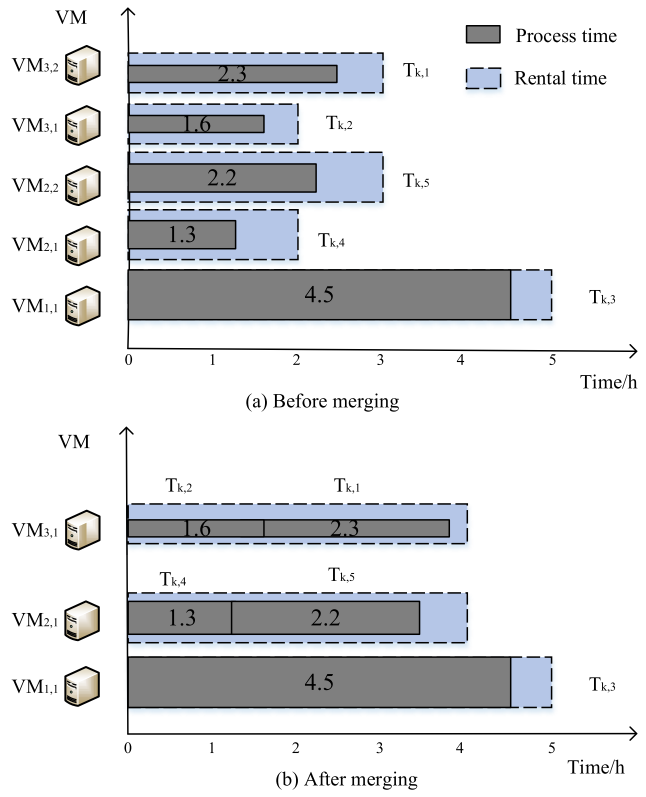

FIGURE 4. An example of FTSM.

V. PERFORMANCE EVALUATION

The proposed algorithm contains several variants for each component. We first calibrate these components and select the best combination for the considered problem. The calibrated algorithm is compared to existing highly related algorithms for similar problems. All tested algorithms are coded in Java and run on an Intel Core i5-3479 3.7 GHz with 8 GB of RAM.

To evaluate the proposed algorithm, WorkflowSim simulates resource provisioning and Spark task scheduling in cloud environments. Based on the platform, VM configurations are similar to Amazon EC2.[1] In the experiment, three different types of VMs were considered. The configuration information and price of each VM type are set according to the Amazon instances, as shown in Table 4. The bandwidth inside the data center is assumed to be 100 Mbps.

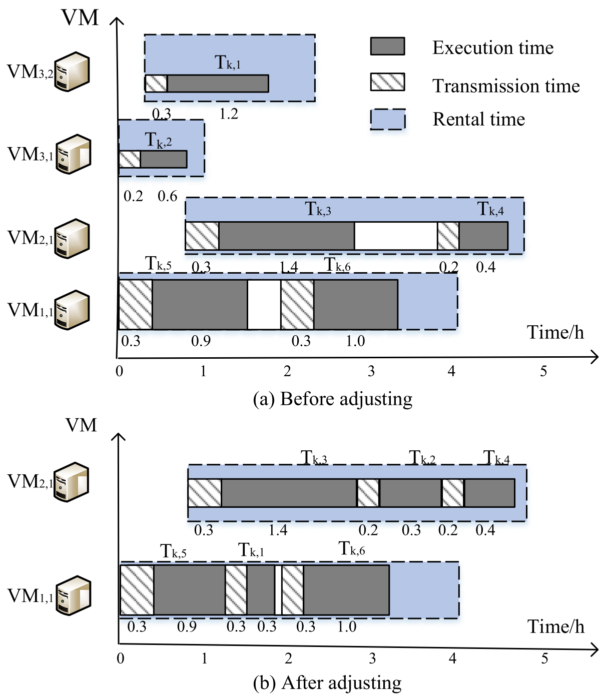

FIGURE 5. An example of ITSF.

Algorithm 5 SIVL (Search Idle VM List)

```
Output: $List$
begin
    $List \leftarrow \emptyset$ ;
    foreach $VM_{p,q} \in VM_{pool}$ do
        Get task list $L_{p,q}$ from $VM_{p,q}$ ;
        for $i = 1$ to $len(L_{p,q})$ do
            if $s_i > f_{i-1}$ then
                Remove $VM_{p,q}$ from $VM_{pool}$;
                Append $VM_{p,q}$ to $List$ ;
                break ;
    return $List$.
```

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

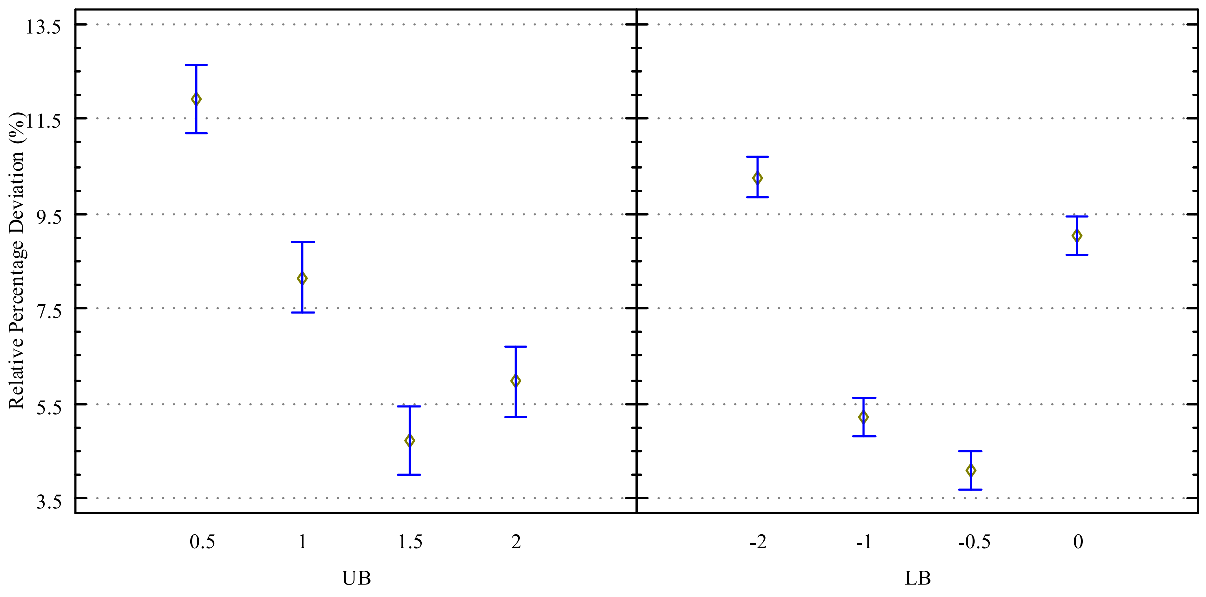

FIGURE 6. Mean plots of parameters for STSDD with 95% Tukey HSD intervals.

TABLE 4. Virtual machine specifications.

Type	vCPU	Memory (GiB)	Pricing
c5.large	2vCPU	4GB	\$ 0.085 per Hour
c5.xlarge	4vCPU	8GB	\$ 0.17 per Hour
c5.2xlarge	8vCPU	16GB	\$ 0.34 per Hour

The multifactor analysis of variance (ANOVA) technique is used to analyze the results. All algorithms are evaluated by relative percentage deviation (RPD). Let C^*_{max} be the best solution found so far. The relative percentage deviation (RPD) is used to measure the performance of the proposed method, which is defined by:

$$RPD = \frac{C_{max} - C^*_{max}}{C^*_{max}} \times 100\%. \tag{24}$$

C denotes the rental cost of an instance, and C^* is the minimum rental cost among all the compared algorithms on the instance.

A. PARAMETER AND COMPONENT CALIBRATION

The number of jobs in a Spark workflow instance takes the value from $\{30, 40, 50, 60\}$. The number of stages in a job is uniformly distributed in [20, 30], and the number of tasks in a stage is uniformly distributed in [100, 150]. The total input data of a stage is divided into multiple blocks. Each block is randomly assigned to a task. The size of each block is 128 MB. The data volume of each task is the number of data blocks assigned to it. The number of data blocks uniformly distributed in [2, 20] for data skew considerations. To verify the different effects on the algorithm under different deadlines. Deadlines are set in the longest scheduling time (LST) and the shortest scheduling time (SST) defined in Eqn. (25). In this paper, β takes the value from $\{0.2, 0.4, 0.6, 0.8, 1\}$ and five deadline levels are defined as D1, D2, D3, D4 and D5, respectively.

$$D = (1 + \beta) \times SST \tag{25}$$

There are $4 \times 5 = 20$ instance combinations, and 10 instances are generated for each combination. Therefore, $20 \times 10 = 200$ instances are generated in total.

In the STSDD framework, $UB \in \{0.5, 1, 1.5, 2\}$ and $LB \in \{-2, -1, 0.5, 0\}$ are adopted. Four stage priorities (HSF, LSRF, LDF, and RAND) are adopted. There are four component candidates (STF, LTF, SLTF, and RAND) for VM type selection and three-component candidates (EATF, LRTIF, and RAND) for VM instance selection. Therefore, there are $4 \times 4 \times 4 \times 4 \times 3 = 768$ tests for each workflow instance, and the total number of tests is $200 \times 768 = 153, 600$.

Experimental results are analyzed by the multifactor analysis of variance (ANOVA) statistical technique. 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. The p-values are less than 0.05, which means that all the factors studied significantly affect the RPD response variable at the 95% confidence level.

The mean plots of the components and the parameters with 95.0% Tukey honest significant difference (HSD) intervals are shown in Figures 6 and 7. From Figure 6, it can be observed that RPD is the minimum at $UB = 1.5$. A smaller UB means that a smaller number of tasks have been classified as small tasks. With the decreasing number of tasks with lower price VMs, the rental cost might increase. It can be observed that RPD is minimum with $LB = -0.5$, i.e., the minimum total rental cost can be obtained. Figure 7

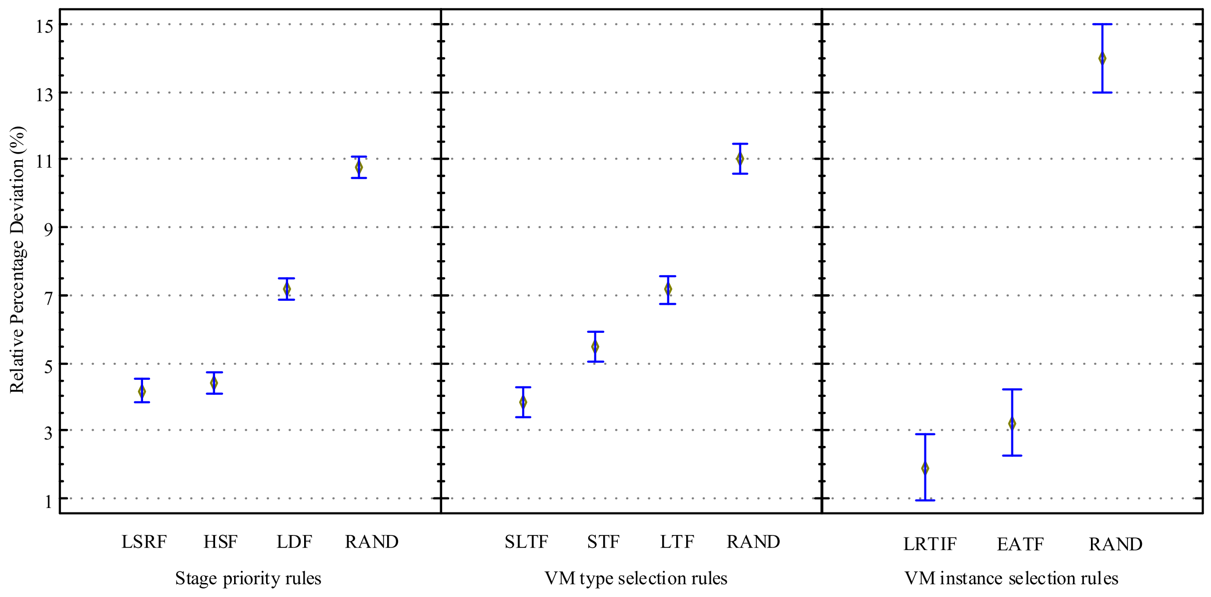

FIGURE 7. Mean plots of components for STSDD with 95% Tukey HSD intervals.

illustrates that RPDs of LSRF are statistically significantly lower than those of HSF, LDF, and RAND among the four components of stage sequencing strategies. The reason lies in those stages with larger skew rates that are always more effective for cost optimization. Among the four VM type selection methods, it can be observed that RPDs of SLTF are statistically significantly lower than those of STF, LTF, and RAND. Because SLTF considers both task skewness and data size, VM types can be selected more accurately. STF considers only task skewness, while LTF considers only the data size to be processed.

Among the three rules for VM instance selection, it can be observed that RPDs of LRTIF are statistically significantly lower than those of EATF and RAND. LRTIF selects VM by making full use of the time slot of each leased VM. EATF greedily selects the earliest available without considering the leased time slots, which wastes the rented time slots. Therefore, STSDD takes $UB = 1.5$, $LB = -0.5$, LSRF, SLTF, and LRTIF in the following comparisons.

B. ALGORITHM COMPARISON

To further evaluate the proposed algorithm's performance, five scientific workflow applications, CyberShake, Montage, Genome, SIPHT, and LIGO, are used. The application instances and task configurations are identical to those of parameter& component calibration. In total, $5 \times 4 \times 5 \times 3 \times 10 = 3,000$ tests are conducted.

The proposed STSDD is compared to two classical algorithms, IC-PCP [11] and CEDA [12]. IC-PCP is a workflow scheduling algorithm that optimizes rental costs under deadline constraints. The algorithm is adapted for Spark task scheduling by finding parts of the critical path and scheduling all critical path tasks to the least expensive VMs. CEDA is a workflow scheduling algorithm for optimizing the total execution time and the rental cost under the deadline constraint. The task with the highest priority is assigned to the least expensive VM meeting the deadline.

To fairly conduct comparisons, the two instance parameters (deadline level and job number) are considered in the five scientific workflow instances: CyberShake, Montage, Genome, SIPHT, and LIGO. ANOVA was also used to analyze the performance. Interactions between each parameter and the compared algorithms on different scientific workflow instances with 95% Tukey HSD intervals are shown in Figures 8-12.

The interaction plots of the compared algorithms on CyberShake workflows with different deadline levels and job numbers with 95% Tukey HSD confidence level intervals are shown in Figure 8. Figure 8 illustrates that STSDD outperforms the other two algorithms for minimizing the rental cost in the Spark application for all deadline levels. With the increase in the deadline level, the three algorithms' RPD values have a downward trend because a higher deadline level implies more tasks choose lower unit price VMs and the rental cost decreases. With the increase in job number, the RPD values of the three algorithms increase because more processing tasks require more VMs and the rental cost increases.

Figure 9 illustrates that STSDD outperforms the other two algorithms for Montage Spark applications. RPDs of IC-PCP and CEDA increase with the slight increase in the job number of workflows. The three algorithms have similar changes in the deadline level.

Figure 10 illustrates that STSDD outperforms the other two algorithms for minimizing the rental cost in genome Spark applications. When the deadline level is lower, STSDD is more effective in cost optimization than the two algorithms because STSDD preferentially schedules tasks with

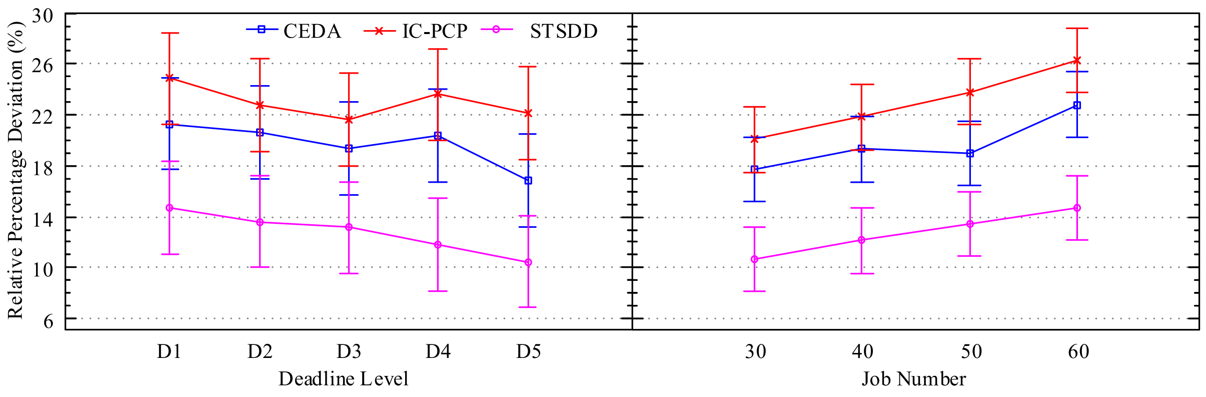

FIGURE 8. Interaction plots of the compared algorithms and instance parameters on CyberShake applications with 95.0% Tukey HSD confidence level intervals.

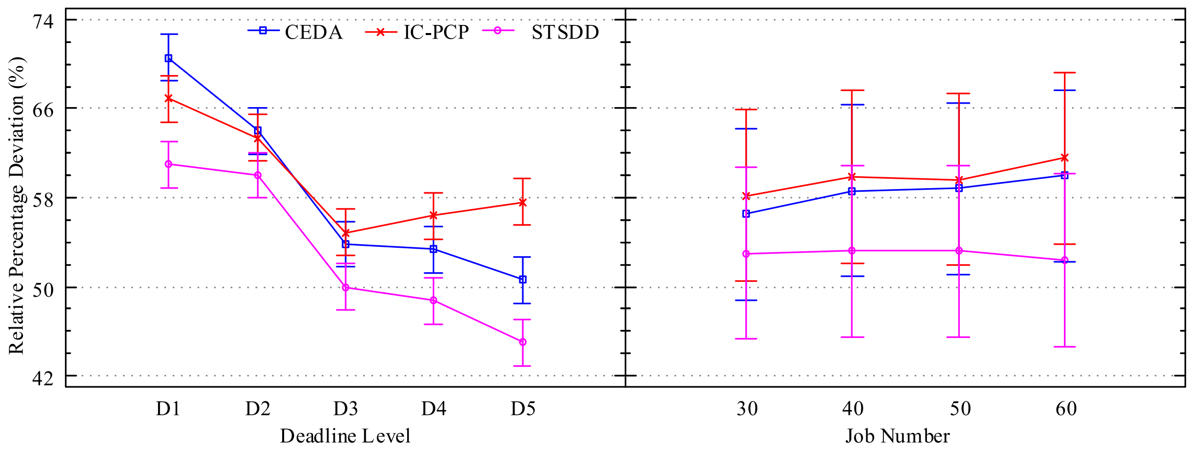

FIGURE 9. Interaction plots of the compared algorithms and instance parameters on Montage applications with 95.0% Tukey HSD confidence level intervals.

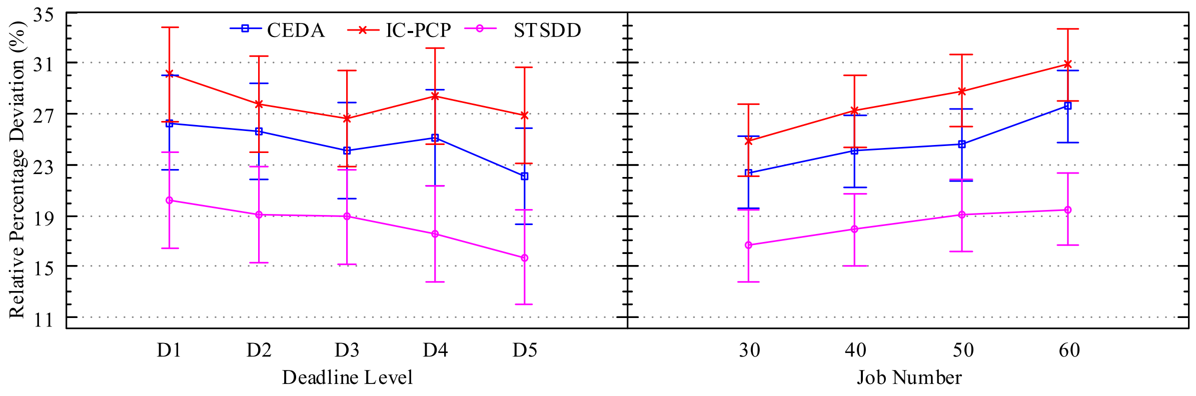

FIGURE 10. Interaction plots of the compared algorithms and instance parameters on Genome applications with 95.0% Tukey HSD confidence level intervals.

data skew and adjusts schedules to further reduce the rental cost.

Figure 11 illustrates that STSDD outperforms the other two algorithms for minimizing the rental cost in SIPHT Spark applications. RPDs of IC-PCP and CEDA increase with the increase in job number of workflows. When the deadline level is lower, STSDD is more effective for cost optimization than the other two compared algorithms because STSDD gives tasks with data skew higher priorities than the other two. In addition, STSDD reduces the rental cost by scheduling adjustments. RPDs of STSDD change slightly with the increase in the job number of workflows.

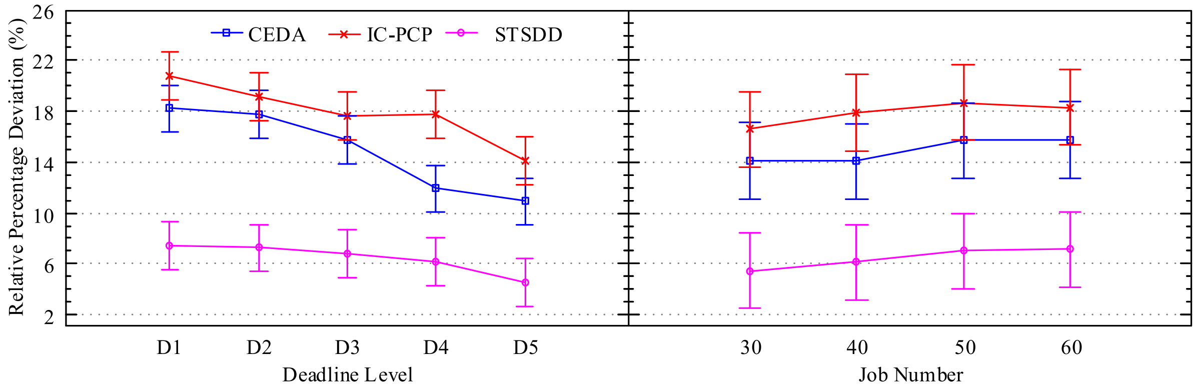

FIGURE 11. Interaction plots of the compared algorithms and instance parameters on SIPHT applications with 95.0% Tukey HSD confidence level intervals.

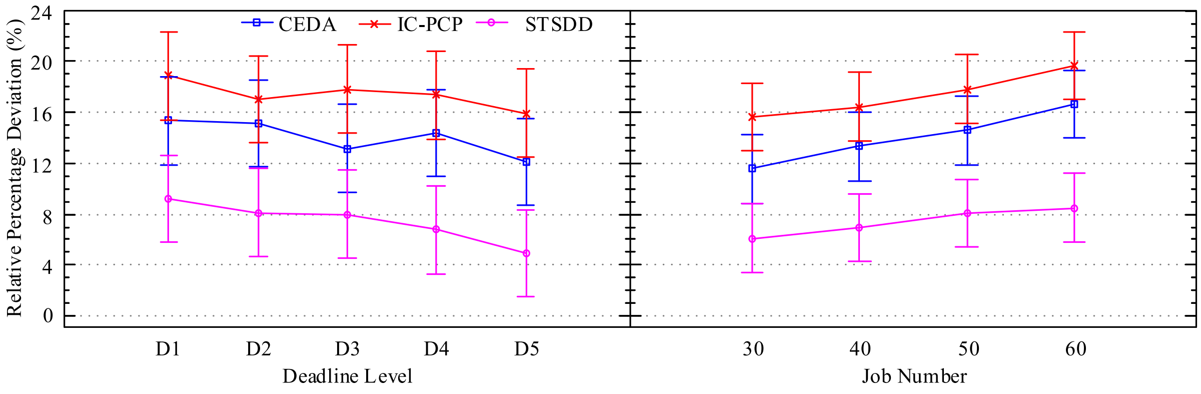

FIGURE 12. Interaction plots of the compared algorithms and instance parameters on LIGO applications with 95.0% Tukey HSD confidence level intervals.

Figure 12 illustrates that STSDD outperforms the other two algorithms for minimizing the rental cost in LIGO Spark applications. The RPDs of the three algorithms decrease with an increasing deadline level and an increase with an increasing job number. The reason is similar to the above cases. Slightly different from the SIPHT examples, STSDD optimizes the rental cost more effectively than the other two algorithms when the deadline level is higher. Since the two compared algorithms calculate critical paths using estimation methods, it may appear that the estimated critical path is no longer a critical path after actual execution.

VI. CONCLUSION

In this paper, the Spark task scheduling problem is considered to minimize Spark jobs' rental cost with skewed data while meeting the deadline in a public cloud with heterogeneous nodes. A modified architecture is constructed, and a heuristic algorithm is proposed in terms of the unique characteristics of the problem under study. Parameters and components of the proposal are calibrated using the ANOVA technique over many random instances. Two modified classical algorithms for similar problems are employed to evaluate the performance of the proposed STSDD algorithm over standard scientific workflow instances. Experimental results indicate that the proposed algorithm outperforms the compared algorithms with different deadline levels and job numbers because STSDD considers data skew in stage sequencing and task scheduling and improves solutions using the scheduling adjustment.

In the future, scheduling problems with more practical characteristics (e.g., affinity) are promising topics.

REFERENCES

[1] M. A. Irandoost, A. M. Rahmani, and S. Setayeshi, "Mapreduce data skewness handling: A systematic literature review," *Int. J. Parallel Program.*, vol. 47, nos. 5–6, pp. 907–950, 2019.

[2] Q. Chen, J. Yao, and Z. Xiao, "LIBRA: Lightweight data skew mitigation in MapReduce," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 9, pp. 2520–2533, Sep. 2015.

[3] G. Liu, X. Zhu, J. Wang, D. Guo, W. Bao, and H. Guo, "SP-partitioner: A novel partition method to handle intermediate data skew in spark streaming," *Future Gener. Comput. Syst.*, vol. 86, pp. 1054–1063, Sep. 2018.

[4] Z. Cai, X. Li, and J. N. D. Gupta, "Heuristics for provisioning services to workflows in XaaS clouds," *IEEE Trans. Services Comput.*, vol. 9, no. 2, pp. 250–263, Mar. 2016.

[5] X. Zeng, S. K. Garg, Z. Wen, P. Strazdins, A. Y. Zomaya, and R. Ranjan, "Cost efficient scheduling of MapReduce applications on public clouds," *J. Comput. Sci.*, vol. 26, pp. 375–388, May 2018.

[6] M. Mattess, R. N. Calheiros, and R. Buyya, "Scaling MapReduce applications across hybrid clouds to meet soft deadlines," in *Proc. IEEE 27th Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Mar. 2013, pp. 629–636.

[7] N. Zacheilas and V. Kalogeraki, "A Pareto-based scheduler for exploring cost-performance trade-offs for MapReduce workloads," *EURASIP J. Embedded Syst.*, vol. 2017, no. 1, p. 29, Dec. 2017.
[8] D. Cheng, X. Zhou, P. Lama, J. Wu, and C. Jiang, "Cross-platform resource scheduling for spark and MapReduce on YARN," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1341–1353, Aug. 2017.
[9] Z. Liu, H. Zhang, and L. Wang, "Hierarchical spark: A multi-cluster big data computing framework," in *Proc. IEEE 10th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2017, pp. 90–97.
[10] Z. Tang, W. Ma, K. Li, and K. Li, "A data skew oriented reduce placement algorithm based on sampling," *IEEE Trans. Cloud Comput.*, vol. 8, no. 4, pp. 1149–1161, Oct. 2020.
[11] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 158–169, Jan. 2013.
[12] T. Gouasmi, W. Louati, and A. H. Kacem, "Cost-efficient distributed MapReduce job scheduling across cloud federation," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun. 2017, pp. 289–296.

HAIHUA GU received the B.Sc. degree in physics science from Jiangsu Normal University, in 2000, and the M.Sc. degree in applied computer science from Southeast University, in 2009. She is currently an Associate Professor with the School of Artificial Intelligence, Nanjing Vocational College of Information Technology, Nanjing, China. Her research interests include scheduling in cloud computing and service computing.

XIAOPING LI (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in applied computer science from the Harbin University of Science and Technology, Harbin, China, in 1993 and 1999, respectively, and the Ph.D. degree in applied computer science from the Harbin Institute of Technology, Harbin, China, in 2002. He is currently a Distinguished Professor with the School of Computer Science and Engineering, Southeast University, Nanjing, China. He is the author or coauthor of 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*, and *European Journal of Operational Research*. His research interests include scheduling in cloud computing, scheduling in cloud manufacturing, service computing, big data, and machine learning.

ZHIPENG LU received the M.Sc. degree in computer technology from Southeast University, Nanjing, China, in 2018. His current research interests include workflow scheduling in cloud computing and big data.