Noname manuscript No.
(will be inserted by the editor)

Allocating MapReduce workflows with deadlines to heterogeneous servers in a cloud data center

Jia Wang · Xiaoping Li* · Rubén Ruiz · Hanchuan Xu · Dianhui Chu

Received: date / Accepted: date

Abstract Total profit is one of the most important factors to be considered from the perspective of resource providers. In this paper, an original MapReduce workflow scheduling with deadline and data locality is proposed to maximize total profit of resource providers. A new workflow conversion based on Dynamic Programming and ChainMap/ChainReduce is designed to decrease transmission times among MapReduce jobs of workflows. A new deadline division considering execution time, float time and job level is proposed to obtain better deadlines of MapReduce jobs in workflows. With the adapted replica strategy in MapReduce workflow, a new task scheduling is proposed to improve data locality which assigns tasks to servers with the earliest completion time in order to ensure resource providers obtain more profit. Experimental results show that the proposed heuristic results in larger total profit than other adopted algorithms.

Jia Wang and Xiaoping Li
School of Computer Science and Engineering, Southeast University, Nanjing, 211189, China and Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing, 211189, China
E-mail: {wangjia1024, xpli}@seu.edu.cn

Rubén Ruiz
Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46022, València, Spain
E-mail: rruiz@eio.upv.es

Hanchuan Xu
School of Computer Science & Technology, Harbin Institute of Technology, Harbin, Heilongjiang, 150001, China,
E-mail: xhc@hit.edu.cn

Dianhui Chu
School of Computer Science and Software Engineering, the Harbin Institute of Technology, Weihai, Shandong, 264209, China
E-mail: chudh@hit.edu.cn

* The corresponding author is Xiaoping Li. Email: xpli@seu.edu.cn.

Keywords MapReduce workflow scheduling · heterogeneous cloud center · deadline · data locality · profit

1 Introduction

Even though Spark [1] has been widely applied to workflows, many applications (e.g., weather forecasting [2] and frequent itemsets mining [3]) are MapReduce workflows in which activities are MapReduce jobs [4]. The practical example shown in Figure 1 shows an application instance of face recognition MapReduce workflow. In the workflow, there are six dependent MapReduce jobs and each MapReduce job contains many map and/or reduce tasks. Usually each MapReduce workflow application is required to finish before a given deadline. Results of each MapReduce job are stored in HDFS (Hadoop Distributed File System). Since data are located on geo-distributed servers, dealing with the massive transmission times among jobs is imperative. During the process of MapReduce workflows, the requirements of computation resource requests users to pay for services. Also resource providers need to be punished if workflow instances cannot finish before deadlines. Generally, resource providers always focus on total profit for commercial operations [4]. Therefore, it is desirable to develop effective and efficient scheduling methods for MapReduce workflows with deadlines in order to maximize total profit.

In this paper, we consider the problem of scheduling MapReduce workflows to geo-distributed servers in a cloud data center to maximize total profit. The servers are heterogeneous, with different processing speeds and different number of map and reduce slots. Slot is the minimal resource unit in Hadoop. Data blocks are located on these geo-distributed servers. Some of them might be transmitted to target servers if necessary. A set of MapReduce workflow instances are processed, each of which is deadline con-

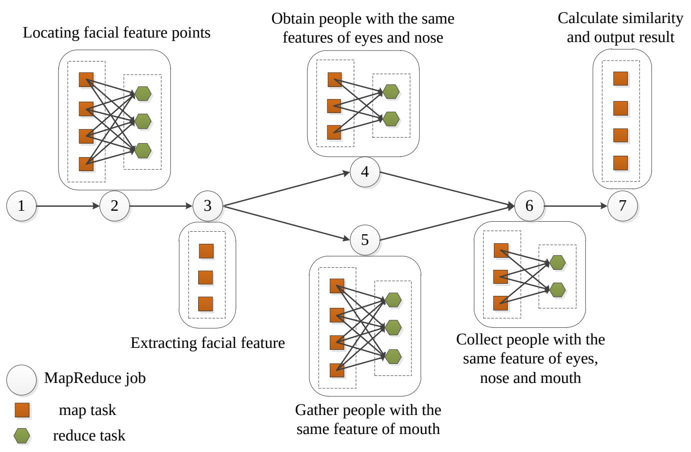
Locating facial feature points
Obtain people with the same features of eyes and nose
Calculate similarity and output result
1
2
3
4
5
6
7
Extracting facial feature
Collect people with the same feature of eyes, nose and mouth
Gather people with the same feature of mouth
MapReduce job
map task
reduce task

Fig. 1: An example of MapReduce workflow instance.

strained. A workflow instance consists of many dependent activities. Many dependent activities consist of a workflow instance. Each activity is a MapReduce job which contains a batch of map and/or reduce tasks. Some jobs may include only map tasks. The map, shuffle and reduce phases are performed sequentially, i.e., the shuffle and reduce phases cannot start until all the map tasks of the job have finished. The same type of tasks (map or reduce tasks) are independent, i.e., they can be processed in parallel.

Because of the heterogeneity and geo-distribution of resources and complex precedence relationships of MapReduce workflows, there are two challenges for the problem under study. (i) Many DAG (Directed Acyclic Graph) constrained workflow jobs competing for limited computation resources makes the allocation of tasks and slots difficult. Moreover, each workflow has to complete before the deadline. Though deadline division is commonly used to get a controllable scheme for workflow scheduling, different division strategies result in varying performance. (ii) Total profit of workflows is closely related to the transmission time among jobs T_j, transmission time among tasks T_t, and the execution time T_e. T_j decreases if adjacent jobs are allocated to the same server, e.g., T_j could be decreased if we allocate jobs 2 and 3 in Figure 1 to the same server. However, different combinations of a workflow instance have different T_j. T_t includes transmission time of intermediate data between jobs (inter-jobs) and those processed in the shuffle phase (intra-jobs), is related to T_j. T_e depends on the speed of the allocated server. These three times are interrelated which results in a different problem of scheduling precedence-constrained jobs and tasks.

The total profit of workflows is closely related to I/O times, data transmission times and processing times of tasks/jobs. Since workflows are constrained by deadlines, it is crucial to design an effective workflow scheduling. In this paper, we adopt Dynamic Programming to dynamically convert workflow instances according to deadlines and execution times. Except workflow conversion, other three components are included: (i) Deadlines are divided into subdeadlines of jobs according to time and space parameters. (ii) Tasks list is constructed with the consideration of workflow sequences, job orders and task sequences. (iii) A heuristic is proposed to allocate tasks to slots. The main contributions of this paper are summarized as follows:

- A workflow conversion is presented to dynamically convert workflows by Dynamic Programming in order to balance transmission times among jobs and processing times of jobs.
- A deadline division considering execution times, float times and locations of MapReduce jobs is proposed to better subdeadlines. Meanwhile a task list is constructed.
- A heuristic task scheduling with replica strategy is investigated.

The rest of the paper is organized as follows: Section 2 reviews related works. A detailed description of the considered problem is presented in Section 3. Section 4 describes a scheduling framework for the problem under study. Experimental results are given in Section 5 followed by conclusions and further work in Section 6.

2 Related work

MapReduce is a popular framework for big data of which job scheduling is crucial for practical applications. Fairness, throughput, response time SLA/deadline, energy efficiency, data locality and resource utilization are commonly considered [5]. Since users require their jobs completed on time, MapReduce job scheduling problems with deadlines have been widely studied in the literature. In terms of the relationships between jobs, there are three kinds of scheduling problems with independent jobs, iterative jobs and dependent jobs, respectively. Since some frameworks without deadlines [21–23] are designed for iterative MapReduce job, the detailed works on independent and dependent MapReduce job scheduling with deadlines are described in this section.

Much attention has been paid to independent MapReduce job scheduling. Deadlines are one of the most important constraints in the existing literature. The required number of slots for each job [6, 7] or the assignment of tasks and resources [8–12] were always calculated based on deadlines. A two-level scheduling framework was designed in [6] which consists of a real-time scheduler and a non real-time scheduler. The real-time scheduler determines the number of slots based on both jobs' deadlines and their running states. Palanisamy et al. [10] proposed two methods for offline and online VM-aware scheduling problems with deadlines, respectively. The former was converted to a multiple bin-packing problem and a heuristic strategy was presented for the latter. Besides deadlines, other factors have been taken into account in this kind of problems, such as data locality [13–17] and energy efficiency [18–20]. Two scheduling strategies were presented [17] for allocating tasks to nodes with or without data locality consideration in order to minimize the number of jobs violating deadlines. A smart energy-efficient MapReduce scheduling algorithm with deadline consideration was introduced by [20] to minimize the energy consumption of a data center which is partially powered by the renewable energy.

Existing studies on dependent MapReduce job scheduling mainly focused on deadlines [4, 24–28, 45], data locality [29] and resource utilization [30]. A task-unit based deadline division and an enhanced multiple-rules based time slot selection were investigated [27] for renting the appropriate type and number of VMs for batch-task based workflow applications. Tang et al. [29] proposed a MapReduce workflow scheduling algorithm with data locality consideration. Both job prioritization and task assignment phases were included. Xu et al. [30] improved resource utilization of servers for cloud-based MapReduce workflows. An optimal assignment of tasks and a heuristic strategy were presented for matching VMs and tasks. Ardagna et al. [31] migrated workflow to reasonable service according to the physical and logical architecture at runtime. BaresiA et al. [32] bound workflow and service dynamically in terms of time parameters. However, the problem considered in this paper focuses on the composition and scheduling of MapReduce workflow.

In addition, transmission times among tasks and jobs have a great influence on the performance of practical MapReduce based systems, especially among geo-distributed servers in cloud data centers. Lim et al. [33] showed that transmission times among jobs might be greatly reduced by converting workflows with data dependencies in terms of original and intermediate data. Some other existing studies optimized transmission times among jobs. Data transmission at the shuffle phase has seldom been considered in existing dependent MapReduce job scheduling problems with deadlines, which always result in a poor assignment of tasks to slots [35]. For independent MapReduce jobs, data operators (e.g., partition, aggregation) [35,36] and transfer routers [34, 37, 38] were usually highlighted on data transmission to reduce shuffle times. Various strategies of shuffle phase results in a different performance for the reduce phase. Few dependent MapReduce job scheduling problems with deadlines have taken into account the influence of the shuffle phase on the reduce phase. For independent MapReduce jobs, reduce tasks were assigned in accordance with the data size and the network distance [39] or the estimated shuffle time and the network distance [40]. Deadlines were seldom considered among assignments of reduce tasks.

Compared to the existing MapReduce workflow scheduling with deadlines, the problem considered in this paper is different in two aspects: (i) Apart from optimizing assignments of tasks to slots in map and reduce phases, an improvement of data transmission is also considered in this paper. (ii) Since resource providers care more about total profit, the objective of this paper is to maximize total profit. To the best of our knowledge, the problem of scheduling dependent MapReduce jobs constrained by deadlines and data locality to maximize total profit has not been studied yet in the literature, as the previous review shows.

3 Problem description

MapReduce workflows share servers in the cloud data center. All the considered batch of workflow instances arrive at time 0. For each workflow, the penalty per unit time and profit are given. Map tasks are assigned only to map slots, so do reduce tasks. Reduce tasks of each job can start only when all its map tasks are finished. Each task is processed by only one slot at the same time while each slot is allocated to only one task at a time. Tasks are non-preemptive during processing. Data needed by workflows is distributed over heterogeneous geo-distributed physical servers. Server or task failures during execution are not considered. We focus on the MapReduce workflow job scheduling in a cloud data center to maximize the total profit. Symbols used in the following are shown in Table 1.

In this paper, n MapReduce workflow instances $W = \{W_1, W_2, \ldots, W_n\}$ are processed by a cloud data center with m geo-distributed heterogeneous servers $S = \{S_1, S_2, \ldots, S_m\}$. We assume that there is only one server at each place (all servers in the same place can be regarded as one super server) in order to simply the problem under studied. Each workflow instance can be represented by a DAG (Directed Acyclic Graph) $W_i = \{V_i, E_i\}$ with a deadline D_i. $V_i = \{J_{i,1}, J_{i,2}, \ldots, J_{i,B_i}\}$ is the set of MapReduce jobs in W_i, i.e., W_i contains B_i $(1 \leq i \leq n)$ jobs. $E_i = \{(j, j')|J_{i,j}, J_{i,j'} \in V_i\}$ are precedence constraints of the jobs. $(j, j') \in E_i$ indicates that $J_{i,j'}$ cannot start until $J_{i,j}$ completes. There are $N^M_{i,j}$ map tasks and $N^R_{i,j}$ reduce tasks in job $J_{i,j}$. We use $a \in \{M, R\}$ to denote the map or reduce phase, i.e., M means the map phase and R the reduce phase. $T^a_{i,j,k}$ represents the k^{th} type a (map or reduce) task of job $J_{i,j}$. A profit I_i can be obtained if W_i completes within D_i. A penalty is incurred if the completion of W_i exceeds D_i which is determined by the penalty per unit time P_i and the tardiness $\mathbb{A}_i$. $\mathcal{N}^M_u$ map slots $\{L^M_{u,1}, \ldots, L^M_{u,\mathcal{N}^M_u}\}$ and $\mathcal{N}^R_u$ reduce slots $\{L^R_{u,1}, \ldots, L^R_{u,\mathcal{N}^R_u}\}$ are configured on server S_u according to different servers configurations (such as CPUs, memory). Generally $\mathcal{N}^M_u = \mathcal{N}^R_u$ on each server S_u [41].

The objective of the considered problem is to maximize the total profit I, which is determined by the profit cost I_i, the penalty per unit time P_i, the deadline D_i and the completion time C_i of each workflow instance W_i $(i =$

Table 1: Symbols to be used.

Notation	Definition
n	Number of workflows
W	The set of workflows
W_i	The i^{th} workflow instance
V_i	The set of activities in W_i
E_i	The set of edges in W_i
D_i	Deadline of W_i
$J_{i,j}$	The j^{th} activity in W_i
B_i	Number of activities in W_i
$N^a_{i,j}$	Number of tasks of $J_{i,j}$ at phase a
m	Number of servers
S	The set of servers
S_u	The u^{th} server of S
$\mathcal{N}^a_u$	The number of slots of S_u with type a
$T^a_{i,j,k}$	The k^{th} task of $J_{i,j}$ at phase a
$L^a_{u,v}$	The v_{th} slot of server S_u with type a
I	The total profit of resource provider
I_i	The profit of W_i
P_i	The penalty per unit time of W_i
$\mathbb{A}_i$	The tardiness of W_i
C_i	The completion time of W_i
$F_{i,j}$	The finish time of $J_{i,j}$
$A_{i,j}$	The start time of $J_{i,j}$
$f^a_{i,j,k}$	The completion time of task $T^a_{i,j,k}$
$t^a_{i,j,k}$	The start time of task $T^a_{i,j,k}$
$p^a_{i,j,k}$	The execution time of task $T^a_{i,j,k}$
$\tau^a_{i,j,k}$	The transmission time of task $T^a_{i,j,k}$
$\mathcal{D}^a_{i,j,k}$	The data volume of task $T^a_{i,j,k}$
μ^a_u	The processing speed of server S_u with the task of type a
β	The bandwidth among servers
$\tau^{\mathcal{I},a}_{i,j,k}$	The transmission time of input data of task $T^a_{i,j,k}$
$\tau^{O,a}_{i,j,k}$	The transmission time of output data of task $T^a_{i,j,k}$
$\gamma^a_{i,j,k}$	The ratio of the data volume before processing to that after processing of task $T^a_{i,j,k}$
$\mathcal{I}^M_{i,j,k}$	The set of servers containing data of task $T^M_{i,j,k}$
$O^a_{i,j,k}$	The set of servers storing results of task $T^a_{i,j,k}$

$1,\ldots,n$). More specifically, $I = \sum_{i=1}^{n}(I_i - P_i \times \mathbb{A}_i)$ and $\mathbb{A}_i = \max\{0, C_i - D_i\}$. C_i depends on the finish time F_{i,B_i} of the MapReduce job J_{i,B_i}, i.e., $C_i = F_{i,B_i}$. If job $J_{i,j}$ is one of the immediate predecessors of job $J_{i,j'}$, the start time $A_{i,j'}$ of $J_{i,j'}$ is no less than the finish time of $J_{i,j}$, i.e., $A_{i,j'} \geq \max_{(j,j')\in E_i} F_{i,j}$. $A_{i,j}$ is the earliest start time $t^M_{i,j,k}$ of all the map tasks $T^M_{i,j,k}$. Therefore $t^M_{i,j,k} \geq A_{i,j}$. $F_{i,j}$ depends on the completion time $f^R_{i,j,k'}$ of reduce tasks $T^R_{i,j,k'}$. $F_{i,j} = \max_{1\leq k'\leq N^R_{i,j}} f^R_{i,j,k'}$. Since reduce task $T^R_{i,j,k'}$ can start only after all map tasks of $J_{i,j}$ finish, $t^R_{i,j,k'} \geq \max_{1\leq k\leq N^M_{i,j}} f^M_{i,j,k}$. The completion time $f^a_{i,j,k}$ of task $T^a_{i,j,k}$ is determined by the start time $t^a_{i,j,k}$, the execution time $p^a_{i,j,k}$ and the transmission time $\tau^a_{i,j,k}$, i.e.

$$f^a_{i,j,k} = t^a_{i,j,k} + p^a_{i,j,k} + \tau^a_{i,j,k} \tag{1}$$

Except $t^a_{i,j,k}$, the other two parameters are determined by the following way:

- $p^a_{i,j,k}$ depends on the data volume $\mathcal{D}^a_{i,j,k}$ of task $T^a_{i,j,k}$ and the processing speed μ^a_u of server S_u. Decision variables $x^a_{i,j,k;u,v} \in \{0,1\}$ are defined for assignments of $T^a_{i,j,k}$ to $L^a_{u,v}$. $x^a_{i,j,k;u,v} = 1$ only if task $T^a_{i,j,k}$ is assigned to slot $L^a_{u,v}$ and 0 otherwise. Therefore, $p^a_{i,j,k} = \sum_{u=1}^{m}\sum_{v=1}^{\mathcal{N}^a_u} \mathcal{D}^a_{i,j,k}/\mu^a_u \times x^a_{i,j,k;u,v}$. The data processed by a reduce task $T^R_{i,j,k'}$ comes from more than one map task, i.e., data transmission is necessary among servers. We use $\gamma^a_{i,j,k}$ to denote the ratio of the data volume before processing to that after processing of task $T^a_{i,j,k}$, i.e., $\mathcal{D}^R_{i,j,k'} = \sum_{k=1}^{N^M_{i,j}} \mathcal{D}^M_{i,j,k}\gamma^M_{i,j,k}$.
- $\tau^a_{i,j,k}$ is closely related to the bandwidth β between servers. In order to improve data locality in map and reduce phases, we consider replicas as three in Hadoop. In other words, there are 3 input and 3 output replications for each map/reduce task. Therefore $\tau^a_{i,j,k}$ consists of the transmission time of input data $\tau^{\mathcal{I},a}_{i,j,k}$ and that of output data $\tau^{O,a}_{i,j,k}$, i.e., $\tau^a_{i,j,k} = \tau^{\mathcal{I},a}_{i,j,k} + \tau^{O,a}_{i,j,k}$. Assume $\mathcal{I}^M_{i,j,k}$ is the set of three servers containing input data of $T^M_{i,j,k}$ and $O^a_{i,j,k}$ are the servers storing output data of $T^a_{i,j,k}$. $\tau^{\mathcal{I},M}_{i,j,k}$ is the minimum transmission time from the servers in $\mathcal{I}^M_{i,j,k}$ to the server where $T^M_{i,j,k}$ is allocated, i.e.,

$$\tau^{\mathcal{I},M}_{i,j,k} = \begin{cases} 0, & \text{If } T^M_{i,j,k} \text{ is allocated to any server in } \mathcal{I}^M_{i,j,k} \\ \frac{\mathcal{D}^{Mc}_{i,j,k}}{\beta}, & \text{Otherwise} \end{cases}$$

Since generally there are many map tasks transmitting mapped data to a reduce task and every copy of mapped data has two other copies, $\tau^{\mathcal{I},R}_{i,j,k'}$ is the maximum transmission time from the server containing the results of all map tasks to the server where the reduce task $T^R_{i,j,k'}$ is located, i.e., $\tau^{\mathcal{I},R}_{i,j,k'} = \max_{1\leq k\leq N^M_{i,j}} \tau^{\mathcal{I},R'}_{i,j,k'}$ where $\tau^{\mathcal{I},R'}_{i,j,k'}$ is the minimum transmission time of one map task output. The calculation of $\tau^{\mathcal{I},R'}_{i,j,k'}$ is similar to that of $\tau^{\mathcal{I},M}_{i,j,k'}$. $\tau^{O,a}_{i,j,k}$ is the maximum transmission time from the server processing $T^a_{i,j,k}$ to servers $S_w \in O^a_{i,j,k}$, $\tau^{O,a}_{i,j,k} = \frac{\mathcal{D}^a_{i,j,k}\gamma^a_{i,j,k}}{\beta}$ according to the different locations of the three replicas.

To illustrate the relationships among the parameters, Figure 2 shows an example of assigning some tasks of Mapreduce job $J_{i,4}$ in workflow instance W_i to some slots. The grey colored rectangles are busy slots and the white ones are idle slots. The map task $T^M_{i,4,2}$ is to be scheduled with $\mathcal{I}^M_{i,4,2} = \{S_2, S_u, S_m\}$. Since only map slot $L^M_{1,2}$ is idle, task $T^M_{i,4,2}$ is assigned to $L^M_{1,2}$ and $x^M_{i,4,2;1,2} = 1$. The transmission time $\tau^M_{i,4,2} = \tau^{\mathcal{I},M}_{i,4,2} + \tau^{O,M}_{i,4,2}$. $\tau^{\mathcal{I},M}_{i,4,2}$ is the minimum transmission time from $\{S_2, S_u, S_m\}$ to S_1 when data $\mathcal{D}^M_{i,4,2}$ is transmitted from servers in $\mathcal{I}^M_{i,4,2}$ to S_1. $\tau^{O,M}_{i,4,2}$ is the minimum transmission time of the output data of task $T^M_{i,4,2}$ from S_1 to $O^M_{i,4,2} = \{S_w, S_{u'}, S_m\}$. The reduce task $T^R_{i,4,1}$ is allocated to slot $L^R_{u',1}$ according to current status of servers. Suppose data $\mathcal{D}^R_{i,4,1}$ of $T^R_{i,4,1}$ comes from the results of $T^M_{i,4,2}$ and $T^M_{i,4,3}$. Since $O^M_{i,4,2} = \{S_w, S_{u'}, S_m\}$ and $O^M_{i,4,3} = \{S_2, S_u, S_m\}$, $\tau^{\mathcal{I},R}_{i,4,1}$ transfer the intermediate data with size $\mathcal{D}^M_{i,4,2} \times \gamma^M_{i,4,2}$ from servers in $O^M_{i,4,2}$ to $S_{u'}$ and the intermediate data with size $\mathcal{D}^M_{i,4,3} \times \gamma^M_{i,4,3}$ from servers in $O^M_{i,4,3}$ to $S_{u'}$. The transmission time of $\mathcal{D}^M_{i,4,2} \times \gamma^M_{i,4,2}$ is the minimum time from $\{S_w, S_{u'}, S_m\}$ to $S_{u'}$ and that of $\mathcal{D}^M_{i,4,3} \times \gamma^M_{i,4,3}$ is the minimum time from $\{S_2, S_u, S_m\}$ to $S_{u'}$. $\tau^{\mathcal{I},R}_{i,4,1}$ is the maximum time of transmission time of $\mathcal{D}^M_{i,4,2} \times \gamma^M_{i,4,2}$ and $\mathcal{D}^M_{i,4,3} \times \gamma^M_{i,4,3}$. At last, the result $\mathcal{D}^R_{i,4,1} \times \gamma^R_{i,4,1}$ of the reduce task $T^R_{i,4,1}$ is stored in $O^R_{i,4,1} = \{S_2, S_u, S_w\}$ with time $\tau^{O,R}_{i,4,1}$ which is the minimum time from $S_{u'}$ to $\{S_2, S_u, S_w\}$.

Based on above descriptions and assumptions, the problem under study can be mathematically modelled as follows:

$$\max I = \sum_{i=1}^{n}(I_i - P_i \times \mathbb{A}_i) \tag{2}$$

s.t.

$$\mathbb{A}_i \geq 0 \qquad \forall i \in \{1, 2, \ldots, n\} \tag{3}$$

$$\mathbb{A}_i \geq C_i - D_i \qquad \forall i \in \{1, 2, \ldots, n\} \tag{4}$$

$$C_i \leq \max_{1 \leq k' \leq N^R_{i,B_i}} \sum_{u'=1}^{m} \sum_{v=1}^{\mathcal{N}^R_{u'}} f^R_{i,B_i,k'} \times x_{i,B_i,k';u',v}$$
$$\forall i \in \{1, 2, \ldots, n\} \tag{5}$$

$$t^R_{i,j,k'} \geq \sum_{u=1}^{m} \sum_{v=1}^{\mathcal{N}^M_u} f^M_{i,j,k} \times x^M_{i,j,k;u,v} \qquad \forall i \in \{1, 2, \ldots, n\},$$
$$\forall j \in \{1, 2, \ldots, B_i\}, \forall k \in \{1, 2, \ldots, N^M_{i,j}\},$$
$$\forall k' \in \{1, 2, \ldots, N^R_{i,j}\} \tag{6}$$

$$t^M_{i,j',k} \geq \max_{(j,j') \in E_i} F_{i,j} \qquad \forall i \in \{1, 2, \ldots, n\},$$
$$\forall j \in \{1, 2, \ldots, B_i\}, \forall k \in \{1, 2, \ldots, N^M_{i,j}\},$$
$$\forall j' \in \{1, 2, \ldots, B_i\} \tag{7}$$

$$F_{i,j} \leq \max_{1 \leq k' \leq N^R_{i,j}} \sum_{u'=1}^{m} \sum_{v=1}^{\mathcal{N}^R_{u'}} f^R_{i,j,k'} \times x_{i,j,k';u',v}$$
$$\forall i \in \{1, 2, \ldots, n\}, \forall j \in \{1, 2, \ldots, B_i\} \tag{8}$$

$$t^M_{i,1,k} \geq 0 \qquad \forall i \in \{1, 2, \ldots, n\}, \forall k \in \{1, 2, \ldots, N^M_{i,j}\} \tag{9}$$

$$\sum_{u=1}^{m} \sum_{v=1}^{\mathcal{N}^a_u} x^a_{i,j,k;u,v} = 1 \qquad \forall i \in \{1, 2, \ldots, n\},$$
$$\forall j \in \{1, 2, \ldots, B_i\}, \forall k \in \{1, 2, \ldots, N^M_{i,j}\}, a \in \{M, R\} \tag{10}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{B_i} \sum_{k=1}^{N^a_{i,j}} x^a_{i,j,k;u,v} = 1 \qquad \forall u \in \{1, 2, \ldots, m\},$$
$$\forall v \in \{1, 2, \ldots, \mathcal{N}^a_u\}, a \in \{M, R\} \tag{11}$$

$$x^a_{i,j,k;u,v} \in \{0, 1\} \qquad \forall i \in \{1, 2, \ldots, n\},$$
$$\forall j \in \{1, 2, \ldots, B_i\}, \forall k \in \{1, 2, \ldots, N^M_{i,j}\},$$
$$\forall u \in \{1, 2, \ldots, m\}, \forall v \in \{1, 2, \ldots, \mathcal{N}^a_u\}, a \in \{M, R\} \tag{12}$$

Equation (3) and (4) define the tardiness time of W_i. Equation (5) guarantees the completion time of workflow instance W_i. Due to the precedence among map and reduce phases, formula (6) ensures that the start time of any reduce tasks is no less than completion time of any map tasks. Many MapReduce jobs in a workflow instance have precedences, formula (7) and (8) controls that $J_{i,j'}$ cannot start processing until all its immediate predecessors are finished. Formula (9) assures that start time of any map tasks is non-negative. Equation (10) makes sure that each map or reduce task can be assigned only to one slot and equation (11) ensures that each map or reduce slot can be assigned to only one task. Finally, formula (12) defines the decision variable $x^a_{i,j,k;u,v}$.

4 Proposed algorithms

In the considered scenario where jobs arrive at the same time and servers are free, the independent MapReduce job scheduling problem considering data locality has been shown to be NP-hard [42]. In this paper, MapReduce workflow scheduling with deadlines is at least as hard as the above problem because MapReduce jobs are dependent in a workflow instance and both deadlines and data locality are considered. Heuristics are effective for NP-hard combinational optimization problems. In this paper, we propose a heuristic for this MapReduce workflow scheduling framework.

In the proposed framework, all map/reduce slots in the cloud data center are managed by a list $\mathcal{L}^a_s$. Each workflow W_i is converted using the proposed CWDP (the Conversion of Workflow based on Dynamic Programming). In terms of the converted workflow W^c_i, D_i of W_i is divided into $\mathbb{D}_{i,j}$ of MapReduce job $J^c_{i,j}(1 \leq j \leq B^c_i)$ by the designed DFL (the Deadline Division based on Float time and job Level), in which B^c_i is the number of jobs in W^c_i. $N^{Mc}_{i,j}$ map tasks and $N^{Rc}_{i,j}$ reduce tasks in each MapReduce job $J^c_{i,j}$ are ordered by the LTF (Longest Time First) rule according to [43]. All the corresponding notations in the converted workflow use now ‘c’ as a superscript. According to the workflow sequence $\mathcal{L}_w$, job sequence $\mathcal{L}_j$ and the job's task orders, map/reduce tasks are selected and kept in the task list $\mathcal{L}_t$ using the proposed CT (the Construction of Task list). In other words, $\mathcal{L}_t$ contains both map and reduce tasks. A new TS (Task Scheduling) procedure is proposed to assign $\mathcal{L}_t$ to $\mathcal{L}^a_s$ for maximizing total profit. As aforementioned, the proposed framework contains four main components: constructing workflow instance W^c_i, dividing workflow deadline D_i, constructing task list $\mathcal{L}_t$ and scheduling tasks. The

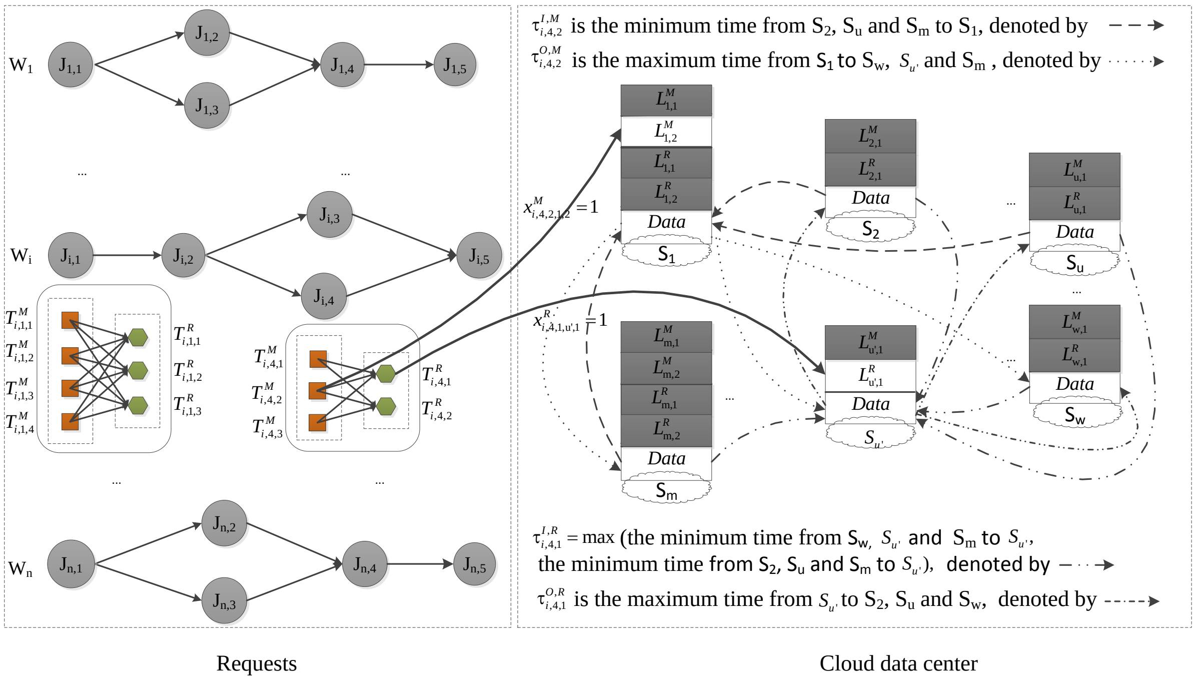
$\tau^{I,M}_{i,4,2}$ is the minimum time from S_2, S_u and S_m to S_1, denoted by
$\tau^{O,M}_{i,4,2}$ is the maximum time from S_1 to S_w, $S_{u'}$ and S_m, denoted by
$\tau^{I,R}_{i,4,1}$ = max (the minimum time from S_w, $S_{u'}$ and S_m to $S_{u'}$, the minimum time from S_2, S_u and S_m to $S_{u'}$), denoted by
$\tau^{O,R}_{i,4,1}$ is the maximum time from $S_{u'}$ to S_2, S_u and S_w, denoted by
$x^M_{i,4,2,1,2}=1$
$x^R_{i,4,1,u',1}=1$
Requests
Cloud data center

Fig. 2: Relationships among the parameters.

WS (Workflow Scheduling) framework is shown in Algorithm 1.

Algorithm 1: Workflow Scheduling (WS) framework

```
1 Initialize list of map/reduce slots $\mathcal{L}^a_s$;
2 for $i = 1$ to $n$ do
3     Convert workflow instance $W_i$ to $W^c_i$ by CWDP
      (Algorithm 2);
4     Divide $D_i$ to $\mathbb{D}_{i,j}$ of job $J^c_{i,j}$ in $W^c_i$ by DFL
      (Algorithm 3);
5     for $j = 1$ to $B^c_i$ do
6         Sort map and reduce tasks of job $J^c_{i,j}$ using LTF;
7 Constructing map and reduce task list $\mathcal{L}_t$ by CT (Algorithm 4);
8 Call TS to allocate $\mathcal{L}_t$ to $\mathcal{L}^a_s$;
9 return.
```

4.1 Converting workflows

Data transmission times among jobs are closely related to completion times of jobs and workflows. Most papers studying independent MapReduce job scheduling consider transmission times of input data from HDFS to target servers. They avoid transmission times of output data because single MapReduce job can obtain final results. However, in MapReduce workflows, the processed data of job $J_{i,j'}$ is determined by the output of all its immediate predecessors, i.e., output data of $J_{i,j} \in \{(j,j') \in E_i\}$ need transfer to HDFS. In other words, transmission time of output data of jobs must be considered in MapReduce workflow scheduling. Workflow conversion can greatly decrease data transmission times among jobs by joining multiple jobs into one. Workflow is converted according to detailed functions of tasks in [33]. It has to be noted that resource providers do not usually know this information beforehand. Therefore, a new workflow conversion needs to be designed from the resource providers perspective.

The completion time C_i has great influence on the total benefit in terms of equation (2) and formula (4). The combination of jobs can decrease transmission times and C_i, e.g., ChainMap/ChainReduce are used to combine MapReduce jobs for better performance. Since the combination of ChainMap/ChainReduce without any help of detailed functions of tasks, a workflow conversion with ChainMap/ChainReduce is proposed to maximize total profit I.

4.1.1 ChainMap/ChainReduce

ChainMap/ChainReduce is generated for MapReduce jobs with linear constraint to combine some jobs in order to decrease completion times. With the constraints of ChainMap/ChainReduce, jobs are combined as much as possible so as to decrease transmission times among jobs. Since there are few Reduce-only jobs (MapReduce jobs have no map phase) [44], both Map-only jobs and MapReduce jobs are considered in this paper. According to ChainMap/ChainReduce,

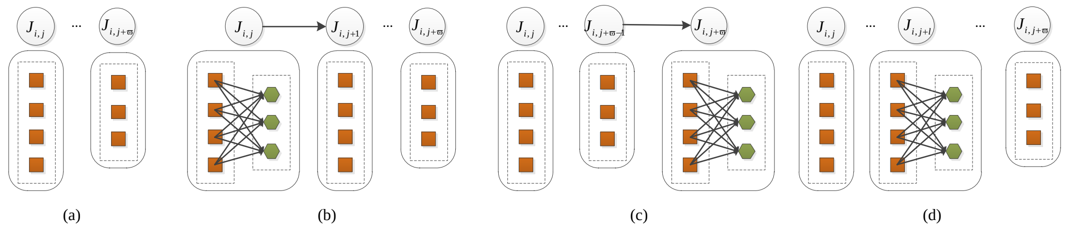

Fig. 3: Four basic combinations of MapReduce jobs.

four basic combinations of MapReduce jobs are shown in Figure 3. $\varpi + 1$ jobs are converted into a new job $J^{c}_{i,j}$. In general $N^{a}_{i,j}$ and $\gamma^{a}_{i,j,k}(1 \leq k \leq N^{a}_{i,j})$ of job $J_{i,j}$ change to $N^{ac}_{i,j}$ and $\gamma^{ac}_{i,j,k}(1 \leq k \leq N^{ac}_{i,j})$ of $J^{c}_{i,j}$ after the combination. Because of the static size of the input and output data of $J_{i,j}$, the calculations of $N^{ac}_{i,j}$ and $\gamma^{ac}_{i,j,k}$ for the four combinations are shown as follows: Figure 3 (a) shows the combination of Map-only jobs. $N^{Mc}_{i,j} = \frac{\sum_{o=j}^{j+\varpi} N^{M}_{i,o}}{\varpi+1}$, $N^{Rc}_{i,j} = 0$, $\gamma^{Mc}_{i,j,k} = \gamma^{Mc}_{i,j+\varpi,k}$ and $\gamma^{Rc}_{i,j,k} = -$. Figure 3 (b) shows the combination of Map-only jobs and a MapReduce job allocating to the left of all Map-only jobs. $N^{Mc}_{i,j} = N^{M}_{i,j}$, $N^{Rc}_{i,j} = \frac{N^{R}_{i,j} + \sum_{o=j+1}^{j+\varpi} N^{M}_{i,o}}{\varpi+1}$, $\gamma^{Mc}_{i,j,k} = \gamma^{M}_{i,j,k}$ and $\gamma^{Rc}_{i,j,k} = \gamma^{M}_{i,j+\varpi,k}$. The combination of a MapReduce job to the right of all Map-only jobs is shown in Figure 3 (c). $N^{Mc}_{i,j} = \frac{\sum_{o=j}^{j+\varpi} N^{M}_{i,o}}{\varpi+1}$, $N^{Rc}_{i,j} = N^{R}_{i,j}$, $\gamma^{Mc}_{i,j,k} = \gamma^{M}_{i,j+\varpi,k}$ and $\gamma^{Rc}_{i,j,k} = \gamma^{R}_{i,j+\varpi,k}$. The last combination of a MapReduce job at the middle of Map-only jobs is shown in Figure 3 (d) in which $1 \leq l \leq \varpi$. $N^{Mc}_{i,j} = \frac{\sum_{o=j}^{j+l} N^{M}_{i,o}}{l+1}$, $N^{Rc}_{i,j} = \frac{N^{R}_{i,j+l} + \sum_{o=j+l+1}^{j+\varpi} N^{M}_{i,o}}{\varpi-l+1}$, $\gamma^{Mc}_{i,j,k} = \gamma^{M}_{i,j+l,k}$ and $\gamma^{Rc}_{i,j,k} = \gamma^{M}_{i,j+\varpi,k}$.

4.1.2 Examples of workflow conversions

Different workflow conversions result in different completion times for the workflows. In general, C_i depends on the execution time of all jobs $J_{i,j} \in V_i$. The execution time of $J_{i,j}$ consists of that of map and reduce phases (shuffle phase is seen as part of reduce phase in this paper), which are closely related to $\tau^{\mathcal{I},a}_{i,j,k}$, $p^{a}_{i,j,k}$ and $\tau^{O,a}_{i,j,k}$. Suppose $\bar{\mu^{a}}$ and $\bar{L^{a}}$ are the average processing speed of servers and the average allocated number of slots, respectively. The estimated execution time of job $J_{i,j}$ is denoted as:

$$EE_{i,j} = \lceil \frac{N^{M}_{i,j}}{\bar{L^{M}}} \rceil \times (\frac{\mathcal{D}^{M}_{i,j,k}}{\beta} + \frac{\mathcal{D}^{M}_{i,j,k}}{\bar{\mu^{M}}} + \frac{\mathcal{D}^{M}_{i,j,k}\gamma^{M}_{i,j,k}}{\beta})$$
$$+ \lceil \frac{N^{R}_{i,j}}{\bar{L^{R}}} \rceil \times (\frac{\mathcal{D}^{R}_{i,j,k}}{\beta} + \frac{\mathcal{D}^{R}_{i,j,k}}{\bar{\mu^{R}}} + \frac{\mathcal{D}^{R}_{i,j,k}\gamma^{R}_{i,j,k}}{\beta}). \quad (13)$$

With the obtained execution times of jobs, we can obtain C_i.

Figure 4 shows an example of MapReduce workflow W_i. 10 dependent MapReduce jobs are in W_i. Table 2 lists

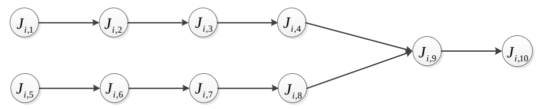

Fig. 4: A workflow containing 10 dependent MapReduce jobs.

Table 2: Configurations of workflow instance W_i in the example.

	$J_{i,1}$	$J_{i,2}$	$J_{i,3}$	$J_{i,4}$	$J_{i,5}$	$J_{i,6}$	$J_{i,7}$	$J_{i,8}$	$J_{i,9}$	$J_{i,10}$
$N^{M}_{i,j}$	63	26	16	4	40	52	21	5	26	10
$N^{R}_{i,j}$	0	0	0	0	64	0	0	0	24	0
$\gamma^{M}_{i,j,k}$	0.4	0.6	0.2	2.4	1.6	0.4	0.2	3.1	0.9	0.5
$\gamma^{R}_{i,j,k}$	-	-	-	-	0.8	-	-	-	0.4	-

the configurations ($N^{a}_{i,j}$ and $\gamma^{a}_{i,j,k}$) of the MapReduce jobs, where $-$ means that there is no $\gamma^{R}_{i,j,k}$. Suppose $\bar{L^{a}}$ =5, $\bar{\mu^{a}}$=50 MB/s and β=100 MB/s, respectively. The execution times of jobs $J_{i,j}(1 \leq j \leq 10)$ are estimated according to equation (13), and the completion time of W_i is 494.336. In terms of the four combinations of Figure 3, the example of Figure 4 can be converted to the workflow of Figure 5 (a) by ChainMap/ChainReduce. The corresponding configurations after the workflow conversion are listed in Table 3 (a). Replacing $N^{a}_{i,j}$ and $\gamma^{a}_{i,j,k}$ by $N^{ac}_{i,j}$ and $\gamma^{ac}_{i,j,k}$ respectively in equation (13), the completion time of W^{c}_{i} in Figure 5 (a) is 291.84. Different combinations of jobs results in different workflow conversions. Another workflow conversion based on ChainMap/ChainReduce is shown in Figure 5 (b) and Table 3 (b) details the resulting configurations. The completion time of W^{c}_{i} in Figure 5 (b) is 264.448 according to equation (13). The completion time of workflow in Figure 5 (b) is less than that of (a), i.e., the ChainMap/ChainReduce without any heuristics need to improve for better workflow conversion. In this paper, an effective workflow conversion based on ChainMap/ChainReduce is proposed in order to maximize the total benefit.

4.1.3 The proposed Conversion of Workflow based on Dynamic Programming (CWDP)

Because ChainMap/ChainReduce is constrained by jobs with linear dependencies, jobs of workflow W_i are divided into

Table 3: Configurations of W_i^c after conversion by Figure 5.

	(a)			(b)				
	$J_{i,1}^c$	$J_{i,2}^c$	$J_{i,3}^c$	$J_{i,1}^c$	$J_{i,2}^c$	$J_{i,3}^c$	$J_{i,4}^c$	$J_{i,5}^c$
$N_{i,j}^{Mc}$	28	40	26	35	4	40	5	26
$N_{i,j}^{Rc}$	0	36	17	0	0	46	0	17
$\gamma_{i,j,k}^{Mc}$	2.4	1.6	0.9	0.2	2.4	1.6	3.1	0.9
$\gamma_{i,j,k}^{Rc}$	-	3.1	0.5	-	-	0.2	-	0.5

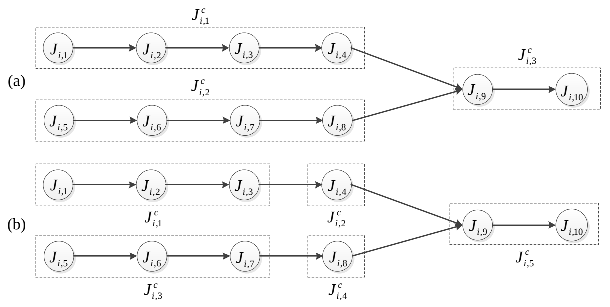

Fig. 5: Different conversions of workflow instances of the workflow instance example of Figure 4.

ξ_i set of jobs $\mathcal{S}_i = \{\mathcal{S}_i^1, \mathcal{S}_i^2, \ldots, \mathcal{S}_i^{\xi_i}\}$. In each set $\mathcal{S}_i^\omega (1 \le \omega \le \xi_i)$, jobs $\{J_{i,l}, J_{i,l+1}, \ldots, J_{i,l+|\mathcal{S}_i^\omega|-1}\}$ have linear dependencies. The workflow conversion in $\mathcal{S}_i^\omega$ can be seen as a segmentation of jobs with constraints of ChainMap and ChianReduce, i.e., jobs in each set need to be segmented in order to maximize profit of the workflow instance. Since the job segmentation of $\mathcal{S}_i^\omega$ meets the optimal substructure and overlapping sub-problems requirements, the problem can be solved by Bottom-Up Dynamic Programming. In other words, p jobs are repeatedly decomposed into a left part of q jobs and a right part of $p-q$ jobs. The optimal solution for the p jobs segmentation is obtained by repeatedly calculating $p-q$ jobs segmentation. An array $r[|\mathcal{S}_i^\omega|+1]$ is used to store execution times of subproblems for $\mathcal{S}_i^\omega$. $r[0] = 0$ means there are no jobs in the right part (i.e., $p - q = 0$) and the execution time is 0. For p jobs segmentation, $r[p]$ is initialized to ∞ and list $R[p]$ stores new combined jobs. Assume each job can be combined with $\varnothing$. $N_{i,l+|\mathcal{S}_i^\omega|-p}^{ac}$, $\gamma_{i,l+|\mathcal{S}_i^\omega|-p,k}^{ac}$ and $EE_{i,l+|\mathcal{S}_i^\omega|-p}^c$ are needed after a new combination. An update of $r[p]$ and $R[p]$ might be necessary. $r[|\mathcal{S}_i^\omega|+1]$ and $R[|\mathcal{S}_i^\omega|]$ are results of the job segmentation for $\mathcal{S}_i^\omega$. Since the number of combined jobs is not continuous, the combined jobs in $\mathcal{S}_i^\omega$ for the converted workflow W_i^c are renumbered. The detailed CWDP algorithm is shown in Algorithm 2. In Algorithm 2, the time complexity of line 1 is $O(B_i)$, that of line 3 is $O(\xi_i)$ while that of line 6, 9 and 24 are the same as $O(|\mathcal{S}_i^\omega|)$. So CWDP has a computational time complexity of $O(B_i + \xi_i|\mathcal{S}_i^\omega|^3)$.

Algorithm 2: Conversion of Workflow based on Dynamic Programming (CWDP)

```
1  Divide $J_{i,j} \in V_i$ into $\xi_i$ set of jobs $\mathcal{S}_i = \{\mathcal{S}_i^1, \mathcal{S}_i^2, \ldots, \mathcal{S}_i^{\xi_i}\}$;
2  $j \leftarrow 1$;
3  for $\omega = 1$ to $\xi_i$ do
4      Initialize array $r[|\mathcal{S}_i^\omega|+1]$;
5      $r[0] \leftarrow 0$;
6      for $p = 1$ to $|\mathcal{S}_i^\omega|$ do
           /* Combine jobs in $\mathcal{S}_i^\omega$ by dynamic programming */
7          $r[p] \leftarrow \infty$;
8          Initialize list $R[p]$;
9          for $q = 1$ to $p$ do
10             $\mathbb{S} \leftarrow \varnothing$;
11             if $q > 1$ then
12                 for $h = 1$ to $q - 1$ do
13                     Add $J_{i,l+|\mathcal{S}_i^\omega|-p+h}$ to $\mathbb{S}$;
14             if $J_{i,l+|\mathcal{S}_i^\omega|-p}$ can be combined with jobs in $\mathbb{S}$ in terms of ChainMap/ChainReduce then
15                 Calculate $N_{i,l+|\mathcal{S}_i^\omega|-p}^{ac}$ and $\gamma_{i,l+|\mathcal{S}_i^\omega|-p,k}^{ac}$ according to combinations in Figure 3;
16                 Calculate $EE_{i,l+|\mathcal{S}_i^\omega|-p}^c$ according to equation (13);
17                 $\mathbb{A}_i' = EE_{i,l+|\mathcal{S}_i^\omega|-p}^c + r[p-q] > D_i ? (EE_{i,l+|\mathcal{S}_i^\omega|-p}^c + r[p-q] - D_i) : 0$;
18                 $\mathbb{A}_i'' = r[p] > D_i ? (r[p] - D_i) : 0$;
19                 if $I_i - P_i \times \mathbb{A}_i' > I_i - P_i \times \mathbb{A}_i''$ then
20                     $r[p] \leftarrow EE_{i,l+|\mathcal{S}_i^\omega|-p}^c + r[p-q]$;
21                     Clear $R[p]$;
22                     Add $J_{i,l+|\mathcal{S}_i^\omega|-p}^c$ to $R[p]$;
23                     if $p - q > 0$ then
24                         for $g = 0$ to $|R[p-q]|$ do
25                             Add the $g^{th}$ job of $R[p-q]$ to $R[p]$;
26             else
27                 Break;
28     for $f = 0$ to $|R[|\mathcal{S}_i^\omega|]|$ do
29         Set $J_{i,j}^c$ as the $f^{th}$ job of $R[|\mathcal{S}_i^\omega|]$; /* Renumber the combined job $J_{i,j}^c$ */
30         $j++$;
31 return.
```

4.2 Dividing deadlines

Let $\mathcal{A}_{i,j}^e$, $\mathcal{F}_{i,j}^e$ and $\mathcal{F}_{i,j}^l$ be the earliest start time, the earliest finish time and the latest finish time of job $J_{i,j}^c$, respectively. Since the subdeadline $\mathbb{D}_{i,j}$ of jobs $J_{i,j}^c \in W_i^c$ is determined by the execution time of $J_{i,j}^c$ and time float [45] among jobs, two divisions considering execution times [46] or time float [45] are used. The former assigns long subdeadlines to tasks/jobs because all jobs' $\mathcal{A}_{i,j}^e$, $\mathcal{F}_{i,j}^e$ and $\mathcal{F}_{i,j}^l$ are fixed with the only one consideration of execution time. While the later always obtain better subdeadlines by the variable $\mathcal{A}_{i,j}^e$,

$\mathcal{F}^e_{i,j}$ and $\mathcal{F}^l_{i,j}$ of jobs, which divides the total time float of the critical path in proportion to the time float and execution time. Because C_i is greatly related to the total profit I, the time float assigned to $J^c_{i,j} \in \{(j, j') \in E^c_i\}$ needs less than that of $J^c_{i,j'}$ in order to make the profit of workflow larger where E^c_i is the set of edges in W^c_i. In other words, subdeadlines of jobs also depend on their locations in the workflow. However, there is no information of job level in deadline division considering time float. Therefore, a new deadline division (DFL) based on job level and time float is proposed.

Let $\mathbb{F}^{\mathbb{P}}$ and $\mathbb{F}^{\mathbb{D}}_{i,j}$ be the total time float among jobs in critical path $\mathbb{P}$ and the distributed time float of $J^c_{i,j}$, respectively. The main problem of DFL is to distribute $\mathbb{F}^{\mathbb{P}}$ to $\mathbb{F}^{\mathbb{D}}_{i,j}$ of all jobs in $\mathbb{P} \in \{J^c_{i,j}, J^c_{i,j+1}, \cdots, J^c_{i,\iota}\}$ according to job level and time float. Assume $\mathbb{F}_{i,j}$ is the float duration of $J^c_{i,j}$ after one deadline division. $\mathbb{F}_{i,j}$ consists of execution time $EE^c_{i,j}$ and $\mathbb{F}^{\mathbb{D}}_{i,j}$, which is similar to [45]. Initially, $\mathbb{F}_{i,j} = EE^c_{i,j}$. With the given $\mathbb{P}$, $\mathbb{F}^{\mathbb{P}}$ is closely related to $\mathcal{A}^e_{i,j}$, $\mathcal{F}^e_{i,j}$ and $\mathcal{F}^l_{i,j}$ of jobs, which can be computed by Algorithm EFT & LFT in [45]. With $\mathcal{F}^l_{i,B^c_i} = D_i$,

$$\mathbb{F}^{\mathbb{P}} = \sum_{\substack{J^c_{i,j} \in \{\mathbb{P}/\Phi/\{J^c_{i,\iota}\}\} \\ (j,j') \in E^c_i}} (\mathcal{A}^e_{i,j'} - \mathcal{F}^e_{i,j}) + \mathcal{F}^l_{i,\iota} - \mathcal{F}^e_{i,\iota}. \quad (14)$$

where Φ is the set of jobs with $\mathcal{A}^e_{i,j} + \mathbb{F}_{i,j} = \mathcal{F}^l_{i,j}$. Assume $\Gamma_{i,j}$ is the level of job $J^c_{i,j}$ in W^c_i. $\mathbb{F}^{\mathbb{D}}_{i,j}$ of $J^c_{i,j}$ is determined by $\mathbb{F}^{\mathbb{P}}$, $\Gamma_{i,j}$ and $\mathbb{F}_{i,j}$, which is different from the deadline division in [45]. $\Gamma_{i,j}$ is calculated as the maximum number of edges in any path from $J^c_{i,1}$ to job $J^c_{i,j}$ in each workflow W^c_i. $\Gamma_{i,j'}$ is calculated recursively according to $\Gamma_{i,j}$ of $J^c_{i,j} \in \{(j,j') \in E^c_i\}$ with $\Gamma_{i,1} = 0$. In other words:

$$\Gamma_{i,j'} = \begin{cases} 0, & j' = 1 \\ \max\limits_{(j,j') \in E^c_i} \Gamma_{i,j} + 1, & \text{Otherwise} \end{cases} \quad (15)$$

Suppose ρ is a weight of $\Gamma_{i,j}$,

$$\mathbb{F}^{\mathbb{D}}_{i,j} = \mathbb{F}^{\mathbb{P}} \times \left(\frac{\rho \Gamma_{i,j}}{\sum\limits_{J^c_{i,j} \in \{\mathbb{P}/\Phi\}} \Gamma_{i,j}} + \frac{(1-\rho)\mathbb{F}_{i,j}}{\sum\limits_{J^c_{i,j} \in \{\mathbb{P}/\Phi\}} \mathbb{F}_{i,j}}\right). \quad (16)$$

Specially when $\mathcal{F}^e_{i,j} + \mathbb{F}^{\mathbb{D}}_{i,j} > \mathcal{F}^l_{i,j}$, $\mathbb{F}^{\mathbb{D}}_{i,j} = \mathcal{F}^l_{i,j} - \mathcal{F}^e_{i,j}$. With a new obtained $\mathbb{F}^{\mathbb{D}}_{i,j}$, $\mathbb{F}_{i,j}$, $\mathcal{A}^e_{i,j'}$ and $\mathcal{F}^e_{i,j'}$ of successors of $J^c_{i,j}$ are updated. After $\mathbb{F}^{\mathbb{P}}$ is distributed to all jobs in $\{\mathbb{P}/\Phi\}$, $\mathbb{F}^{\mathbb{P}}$ and $\mathcal{F}^l_{i,j}$ of $J^c_{i,j} \in \mathbb{P}$ needs to be updated in order to determine whether a new critical path has been generated or not. Finally $\mathbb{D}_{i,j}$ of $J^c_{i,j} \in W^c_i$ is set as $\mathcal{F}^e_{i,j}$, i.e., $\mathbb{D}_{i,j} = \mathcal{F}^e_{i,j}$. The detailed DFL is shown in Algorithm 3. The time complexity of Algorithm 3 is mainly from line 5-16. The complexity of line 5 is $O(B^c_i)$, that of line 7 is $O(|\mathbb{P}|)$, that of line 8 is $O(|\mathbb{P}|)$, that of line 14 is $O(B^c_i)$, that of line 15 is $O(B^c_i)$ while that of line 16 is $O(B^{c2}_i)$. So DFL has a computational time complexity of $O(B^{c2}_i|\mathbb{P}|^2 + B^{c2}_i|\mathbb{P}| + B^{c3}_i)$.

Algorithm 3: Deadline Division based on Float time and job Level (DFL)

```
1  Obtain $EE^c_{i,j}$ and $\Gamma_{i,j}$ for jobs in $W^c_i$ by equation (13) and (15);
2  Calculate $\mathcal{A}^e_{i,j}$, $\mathcal{F}^e_{i,j}$ and $\mathcal{F}^l_{i,j}$ of the jobs using EFT&LFT in [45];
3  Initialize $\mathbb{F}_{i,j} \leftarrow EE^c_{i,j}$ for all jobs $J^c_{i,j} \in W^c_i$;
4  Generate critical path $\mathbb{P}$ in terms of similar method ASSIGNPARENT in [46];
5  while $\mathbb{P} \neq \varnothing$ do
6      Compute $\mathbb{F}^{\mathbb{P}}$ according to equation (14);
7      while $\mathbb{F}^{\mathbb{P}} > 0$ do
           /* Reasonably distribute $\mathbb{F}^{\mathbb{P}}$ to jobs in $\mathbb{P}$ */
8          foreach $J^c_{i,j} \in \mathbb{P}$ do
9              if $J^c_{i,j} \notin \Phi$ then
10                 Calculate $\mathbb{F}^{\mathbb{D}}_{i,j}$ by equation (16);
11                 if $\mathcal{F}^e_{i,j} + \mathbb{F}^{\mathbb{D}}_{i,j} > \mathcal{F}^l_{i,j}$ then
12                     $\mathbb{F}^{\mathbb{D}}_{i,j} \leftarrow \mathcal{F}^l_{i,j} - \mathcal{F}^e_{i,j}$
13                 $\mathbb{F}_{i,j} \leftarrow \mathbb{F}_{i,j} + \mathbb{F}^{\mathbb{D}}_{i,j}$;
14                 Update $\mathcal{A}^e_{i,j'}$ and $\mathcal{F}^e_{i,j'}$ of $J^c_{i,j'} \in \{(j,j') \in E^c_i\}$;
15         Update $\mathcal{F}^l_{i,j}$ of $J^c_{i,j} \in W^c_i$ and $\mathbb{F}^{\mathbb{P}}$;
16     Generate new critical path;
17 Set $\mathbb{D}_{i,j}$ as $\mathcal{F}^e_{i,j}$ of $J^c_{i,j} \in W^c_i$;
18 return.
```

4.3 Constructing the task list

Different task lists result in different completion times for the workflows. In MapReduce workflow scheduling, sequences of workflow, MapReduce jobs and map/reduce tasks have a great influence on the position of each task in $\mathcal{L}_t$. Usually, three strategies are used to schedule multiple workflow instances: independently & sequentially, independently & interleaving and merging [47]. Since various workflow instances have different P_i, i.e., workflows have precedences constraints, the last strategy - merging is not suitable for workflow scheduling in this paper. With the objective of maximizing total profit I, $W^c_i \in W$ are sequenced in non-increasing order of P_i in order for workflows with a higher P_i to finish earlier. Generally, jobs in each workflow are sequenced by a list-based strategy (e.g., topological sequence, priority sequence) [47]. Job sequences based on topologies and deadlines are adopted in this paper. The main difference between these two methods is the order of the jobs that might be processed in parallel in each workflow. For job sequences based on deadlines, jobs are sorted by EDF (Earliest Deadline First) rule in order to make jobs complete as early as possible. While jobs processing in parallel are sequenced randomly by another method. Because of the good performance of LTF for minimizing makespan [43], tasks in each job are ordered by LTF before constructing $\mathcal{L}_t$. As aforementioned, we adopt four methods: WSJT (scheduling workflows sequentially and jobs based on topologies), WSJD (scheduling workflows sequentially and jobs based on deadlines), WIJT (scheduling workflows interleaving and jobs based on topologies) and WIJD (scheduling workflows interleaving and jobs based on deadlines) to construct the task list $\mathcal{L}_t$. With the sequences of workflow, MapReduce jobs and map/reduce tasks, map and reduce tasks are sequentially appened to $\mathcal{L}_t$ in each method. The detailed description of WSJT and WIJT are shown in Algorithms 4 and 5, respectively. Sequencing jobs in non-decreasing order of $\mathbb{D}_{i,j}$ is used to replace Line 3 of Algorithms 4 and 5 in order to obtain the WSJD and WIJD alternatives. Because the time complexity of line 1 is $O(n\log n)$, that of line 2 is $O(n(B_i^c + |E_i^c|))$ while that of line 5 is $O(\sum\limits_{W_i^c\in W} B_i^c(\max\limits_{J_{i,j}^c\in\mathcal{L}_j} N_{i,j}^{Mc} + \max\limits_{J_{i,j}^c\in\mathcal{L}_j} N_{i,j}^{Rc}))$, the time complexity of WSJT is $O(n\log n + n(B_i^c + |E_i^c|) + \sum\limits_{W_i^c\in W} B_i^c(\max\limits_{J_{i,j}^c\in\mathcal{L}_j} N_{i,j}^{Mc} + \max\limits_{J_{i,j}^c\in\mathcal{L}_j} N_{i,j}^{Rc}))$. Similarly, the time complexity of WIJT is $O(n\log n + n(B_i^c + |E_i^c|) + \max\limits_{W_i^c\in W} B_i^c n + \sum\limits_{W_i^c\in W} B_i^c(\max\limits_{J_{i,j}^c\in\mathcal{L}_j} N_{i,j}^{Mc} + \max\limits_{J_{i,j}^c\in\mathcal{L}_j} N_{i,j}^{Rc}))$.

Algorithm 4: Constructing Task list (CT) – WSJT

```
1  Generate $\mathcal{L}_w$ by sequencing workflows in decreasing order of $P_i$;
2  foreach workflow $W_i^c$ in $\mathcal{L}_w$ do
3      Sort jobs of $W_i^c$ according to topological sequence;
4      Append jobs to $\mathcal{L}_j$;
5  foreach MapReduce job $J_{i,j}^c$ in $\mathcal{L}_j$ do
6      Append map tasks to $\mathcal{L}_t$;
7      Append reduce tasks to $\mathcal{L}_t$;
8  return.
```

Algorithm 5: Constructing Task list (CT) – WIJT

```
1  Generate $\mathcal{L}_w$ by sequencing workflows in decreasing order of $P_i$;
2  foreach workflow $W_i^c$ in $\mathcal{L}_w$ do
3      Sort jobs of $W_i^c$ according to topological sequence;
4  for $j = 1$ to $\max_{W_i^c\in\mathcal{L}_w} B_i^c$ do
5      foreach workflow $W_i^c$ in $\mathcal{L}_w$ do
6          if $J_{i,j}^c \in W_i^c$ then
7              Append $J_{i,j}^c$ to $\mathcal{L}_j$;
8  foreach MapReduce job $J_{i,j}^c$ in $\mathcal{L}_j$ do
9      Append map tasks to $\mathcal{L}_t$;
10     Append reduce tasks to $\mathcal{L}_t$;
11 return.
```

4.4 Task scheduling

With the constructed $\mathcal{L}_t$, tasks in $\mathcal{L}_t$ are allocated to map slots in $\mathcal{L}_s^M$ or reduce slots in $\mathcal{L}_s^R$. Transmission times of map/reduce tasks are always considered in order to decrease the completion times of jobs and workflows. Since data chunks of each map task has several replicas, existing map task scheduling strategies always locate map tasks to servers containing their data to minimize transmission times [13–17]. Since the input data of a reduce task is the output data of more than one map task, reduce tasks are allocated to servers with the largest part of data in order to minimize transmission and completion times [39, 40]. Both map and reduce scheduling strategies are focused on independent MapReduce jobs. They are not directly applicable to MapReduce workflow scheduling because the transmission time of output of jobs (transmission time among jobs) must be considered. Therefore, a new task scheduling (TS) considering transmission times of input and output is designed to decrease completion times.

Since the replica strategy can greatly decrease transmission times in MapReduce scheduling, it is adopted to MapReduce workflow scheduling. In map task scheduling of MapReduce, the replica strategy make it more likely that map tasks will allocate to servers where holding their data. The replica strategy is applied in map task scheduling of MapReduce workflow. With the great influence of the shuffle phase (transmission time of input data of reduce tasks) on the reduce phase [34, 35], the replica strategy would adopt to reduce task scheduling of MapReduce workflow. In order to give more chances for a reduce task to decrease transmission time, the output of each map task needs to have a replica, i.e., output of $T_{i,j,k}^{Mc}$ must transfer from the processing server S_u to servers $S_w \in O_{i,j,k}^{Mc}$. Since transmission times among jobs must be considered in MapReduce workflow scheduling, the output of each reduce task also needs to be duplicated to ensure that successors' map tasks can also benefit from replica strategy. As aforementioned, replica strategy would use in MapReduce workflow scheduling.

After the adapted replica strategy in MapReduce workflow scheduling, TS considering replica strategy is proposed. The transmission time consists of input and output data transmission times in TS, which is the main differences from the existing MapReduce scheduling strategies. In other words, $\tau_{i,j,k}^{ac} = \tau_{i,j,k}^{\mathcal{I},ac} + \tau_{i,j,k}^{O,ac}$ for each task $T_{i,j,k}^{ac}$. Obviously:

$$\tau_{i,j,k}^{\mathcal{I},Mc} = \begin{cases} 0, & S_u \in \mathcal{I}_{i,j,k}^{Mc} \\ \frac{\mathcal{D}_{i,j,k}^{Mc}}{\beta}, & \text{Otherwise} \end{cases}$$

and $\tau_{i,j,k}^{O,ac} = \frac{\mathcal{D}_{i,j,k}^{ac}\gamma_{i,j,k}^{ac}}{\beta}$. Assume $\mathcal{I}_{i,j,k}^{Rc}$ is the set of servers holding data of $\mathcal{D}_{i,j,k}^{Rc}$. Initially, $\mathcal{I}_{i,j,k}^{Rc} = \varnothing$. Suppose S_u is the server to process the reduce task $T_{i,j,k}^{Rc}$. Let $\hbar_{w'}$ be the transmission time from $S_{w'}$ to S_u. With the adapted replica strategy at reduce task scheduling, the output data of each map task $T^{Mc}_{i,j,k'}$ need to transfer from $O^{Mc}_{i,j,k'}$ servers to S_u. In order to decrease $\tau^{\mathcal{I},Rc}_{i,j,k}$ of $T^{Rc}_{i,j,k}$, the server S_w in $O^{Mc}_{i,j,k'}$ with the minimal transmission time is added to $\mathcal{I}^{Rc}_{i,j,k}$, i.e., $S_w = \operatorname{argmin}_{S_{w'} \in O^{Mc}_{i,j,k'}} \hbar_{w'}$. According to the constructed $\mathcal{I}^{Rc}_{i,j,k}$, $\tau^{\mathcal{I},Rc}_{i,j,k} = \max_{S_w \in \mathcal{I}^{Rc}_{i,j,k}} \hbar_w$. In terms of the calculated $\tau^{\mathcal{I},Mc}_{i,j,k}$, $\tau^{\mathcal{I},Rc}_{i,j,k}$ and $\tau^{O,ac}_{i,j,k}$, the transmission time of tasks is obtained. The details of CTT (Calculate Transmission Time) are given in Algorithm 6 with a computational time complexity $O(N^{Mc}_{i,j'}|O^{Mc}_{i,j',k'}|)$. With the given slot $L^a_{u,v}$, the completion time $f^{ac}_{i,j,k}$ of task is calculated by $t^{ac}_{i,j,k}$, $p^{ac}_{i,j,k}$ and $\tau^{ac}_{i,j,k}$ according to equation (1). Because the minimization of C_i depends on $f^{ac}_{i,j,k}$, the slot $L^a_{u,v}$ with the minimal $f^{ac}_{i,j,k}$ is chosen for task $T^{ac}_{i,j,k}$ in task scheduling. Assume $\lambda^a_{u,v}$ is the next available time of slot $L^a_{u,v}$. For map tasks, $t^{Mc}_{i,j',k}$ and $f^{Mc}_{i,j',k}$ are initialized to the maximal completion time of jobs $J^c_{i,j} \in \{(j,j') \in E^c_i\}$ and ∞, respectively. More specifically, $t^{Mc}_{i,j',k} = 0$ if $J^c_{i,j'}$ is the first job of workflow W^c_i. After the selection of map slot $L^M_{u,v}$ with the minimal $f^{Mc}_{i,j',k}$, $\lambda^M_{u,v}$ is reset as $f^{Mc}_{i,j',k}$. For reduce tasks, $t^{Rc}_{i,j',k}$ is set to the maximal completion time of map tasks of $J^c_{i,j'}$, i.e, $t^{Rc}_{i,j',k} = \max_{1 \le k' \le N^{Mc}_{i,j'}} f^{Mc}_{i,j',k'}$. Similar to the scheduling of map tasks, a reduce slot $L^R_{u,v}$ with the minimal $f^{Rc}_{i,j',k}$ is chosen to schedule $T^{Rc}_{i,j',k}$ and $\lambda^R_{u,v} = f^{Rc}_{i,j',k}$. Let σ be the number of immediate predecessors of $J^c_{i,j}$. The TS is given in Algorithm 7 and the time complexity of Algorithm 7 is depended on that of line 3-21. Because the time complexity of line 3 is $O(\sum_{i=1}^{n}\sum_{j=1}^{B^c_i}(N^{Mc}_{i,j} + N^{Rc}_{i,j}))$, that of line 9 is $O(B^c_i N^{Rc}_{i,j})$, that of line 11 is $O(N^{Mc}_{i,j'})$, that of line 14 is $O(\sum_{u=1}^{m}\mathcal{N}^M_u + \sum_{u=1}^{m}\mathcal{N}^R_u)$ while that of line 19 is $O(\sum_{u=1}^{m}\mathcal{N}^M_u + \sum_{u=1}^{m}\mathcal{N}^R_u)$, we can obtain the computational time complexity is $O\big((\sum_{i=1}^{n}\sum_{j=1}^{B^c_i}(N^{Mc}_{i,j} + N^{Rc}_{i,j})) \times (\sum_{u=1}^{m}\mathcal{N}^M_u + \sum_{u=1}^{m}\mathcal{N}^R_u) \times (N^{Mc}_{i,j'}|O^{Mc}_{i,j',k'}|)\big)$.

Algorithm 6: Calculate Transmission Time (CTT)

```
   Input: $T^{ac}_{i,j',k}$, $L^a_{u,v}$
   Output: $\tau^{ac}_{i,j',k}$
1  $F = 1$;
2  if $a = M$ then
3      if $S_u \in \mathcal{I}^{Mc}_{i,j',k}$ then
4          F=0;
5      $\tau^{Mc}_{i,j',k} \leftarrow \frac{\mathcal{D}^{Mc}_{i,j',k}}{\beta} \times F + \frac{\mathcal{D}^{Mc}_{i,j',k}\gamma^{Mc}_{i,j',k}}{\beta}$; /* Calculate
           transmission times of map tasks */
6  else
7      $\mathcal{I}^{Rc}_{i,j',k} \leftarrow \varnothing$;
8      for $k' = 1$ to $N^{Mc}_{i,j'}$ do
9          foreach server $S_{w'}$ in $O^{Mc}_{i,j',k'}$ do
10             if $S_{w'} = S_u$ then
11                 $\hbar_{w'} \leftarrow 0$;
12             else
13                 $\hbar_{w'} \leftarrow \frac{\mathcal{D}^{Mc}_{i,j',k'}\gamma^{Mc}_{i,j',k'}}{\beta}$;
14         $S_w \leftarrow \operatorname{argmin}_{S_{w'} \in O^{Mc}_{i,j',k'}} \hbar_{w'}$;
15         Add $S_w$ to $\mathcal{I}^{Rc}_{i,j',k}$; /* Obtain the set of
               servers $\mathcal{I}^{Rc}_{i,j',k}$ */
16     if $S_u \in \mathcal{I}^{Rc}_{i,j',k}$ then
17         $\mathcal{I}^{Rc}_{i,j',k} \leftarrow \mathcal{I}^{Rc}_{i,j',k} - \{S_u\}$;
18     $\tau^{Rc}_{i,j',k} \leftarrow \max_{S_w \in \mathcal{I}^{Rc}_{i,j',k}} \hbar_w + \frac{\mathcal{D}^{Rc}_{i,j',k}\gamma^{Rc}_{i,j',k}}{\beta}$;
       /* Calculate transmission times of
       reduce tasks */
19 return $\tau^{ac}_{i,j',k}$.
```

5 Performance Evaluation

We adopt the prototype proposed in [48] which was commonly used to MapReduce workflows for performance evaluation. The number of MapReduce workflows is set as {50,100,150}. For each workflow, the number of DAG dependent MapReduce jobs is generated with a uniform distribution U(50,500). In other words, a given number of workflows mixes any size (small, medium and large) of workflow. For each MapReduce job, the number of map and reduce tasks are normally distributed in N(154,558) and N(19,145) respectively according to [49]. In terms of distributions of processing times of map and reduce tasks [49], the processing time of each MapReduce job is $\lceil \frac{154}{L^M} \rceil \times 50 + \lceil \frac{19}{L^R} \rceil \times 100$. Hereby, we can obtain the longest processing time $\mathcal{C}^L$ of workflow W_i from the critical path. Assume $D^t_i = \mathcal{C}^L \times 1.0$ and $D^l_i = \mathcal{C}^L \times 2.0$ are the tight and loose deadlines of workflow W_i. D_i of W_i is randomly distributed in U(D^t_i,D^l_i) with the unit being seconds. I_i is generated with a uniform distribution U(1000,10000) according to D_i and practical hourly wage. Meanwhile $P_i = 2 \times \frac{I_i}{D_i}$. In this paper, each data chunk is set as 256MB and three replica are configured. $\gamma^a_{i,j,k}$ is a random value from 0 to 10 according to [50]. All data chunks are randomly distributed among 1000 geo-distributed servers of a cloud data center. According to the size of data chunk and the distributions of processing times of map and reduce tasks [49], the speed of servers for map and reduce tasks are generated with uniform distribution U(2,6) and U(1,3) respectively. Different number of slots are configured to servers with different CPU and memory. The configuration of slots ensures that the number of map slots equals to that of reduce slots. Bandwidth among servers is randomly distributed following the uniform distribution U(10,50) with the unit being MB in this paper.

We run all strategies (which are coded in Java with Eclipse Helios Release JDK1.6) on computers with Intel Core

Algorithm 7: Task Scheduling (TS)

```
1  foreach slot $L^a_{u,v}$ in $\mathcal{L}^a_s$ do
2      $\lambda^a_{u,v} \leftarrow 0$; /* The available time of all slots are set as 0 */
3  foreach task $T^{ac}_{i,j',k}$ in $\mathcal{L}_t$ do
4      $\mathbb{L} \leftarrow \varnothing$;
5      if $a = M$ then
6          if $j' = 1$ then
7              $t^{Mc}_{i,j',k} \leftarrow 0$;
8          else
9              $t^{Mc}_{i,j',k} \leftarrow \max_{(j,j')\in E^c_i, 1\le k'\le N^{Rc}_{i,j}} f^{Rc}_{i,j,k'}$; /* Calculate the start time of map tasks */
10     else
11         $t^{Rc}_{i,j',k} \leftarrow \max_{1\le k'\le N^{Mc}_{i,j'}} f^{Mc}_{i,j',k'}$; /* Calculate the start time of reduce tasks */
12         $\mathcal{D}^{Rc}_{i,j',k} = \sum_{k'=1}^{N^{Mc}_{i,j'}} \mathcal{D}^{Mc}_{i,j',k'}\gamma^{Mc}_{i,j',k'}$; /* Obtain the transmission data size of reduce tasks */
13     $f^{ac}_{i,j',k} \leftarrow \infty$;
14     foreach slot $L^a_{u,v}$ in $\mathcal{L}^a_s$ do
           /* Assign the task to the most reasonable slot */
15         Calculate $\tau^{ac}_{i,j',k}$ using CTT ($T^{ac}_{i,j',k}$, $L^a_{u,v}$);
16         if $f^{ac}_{i,j',k} > \max\{\lambda^a_{u,v}, t^{ac}_{i,j',k}\} + \frac{\mathcal{D}^{ac}_{i,j',k}}{\mu^a_u} + \tau^{ac}_{i,j',k}$ then
17             $f^{ac}_{i,j',k} \leftarrow \max\{\lambda^a_{u,v}, t^{ac}_{i,j',k}\} + \frac{\mathcal{D}^{ac}_{i,j',k}}{\mu^a_u} + \tau^{ac}_{i,j',k}$;
18             $\mathbb{L} \leftarrow L^a_{u,v}$;
19     foreach slot $L^a_{u,v}$ in $\mathcal{L}^a_s$ do
20         if $L^a_{u,v} = \mathbb{L}$ then
21             $\lambda^a_{u,v} \leftarrow f^{ac}_{i,j',k}$; /* Set the available time of slot as the finish time of task */
22 return.
```

i5-3479 3.7GHz processors with 4GB of RAM and with Intel Core i7-3770 3.4GHz processors with 8GB of RAM. With the objective of maximizing total profit I, RDI (Relative Deviation Index) of I is set as the response variable. Let I^* and $I^\dagger$ be the maximal and minimal value of all results with the same number of workflows. According to [52], RDI of I is described as follows:

$$RDI = \begin{cases} 0 & I^\dagger = I^* \\ \frac{I-I^*}{I^\dagger - I^*} & \text{Otherwise} \end{cases} \qquad (17)$$

A smaller RDI of I means a higher total profit.

5.1 Parameters tuning

In terms of the WS framework shown in Section 4, four components (workflow conversion, deadline division, task list construction and task scheduling) have a large expected impact on the performance. Different workflow conversions result in different performance. Let NCW and CWC denote the strategy without workflow conversion and the workflow conversion based on ChainMap/ChainReduce, respectively. Three workflow conversions (the proposed CWDP, CWC and NCW) are compared. Various deadline divisions lead to different performance too. We compare DE [46], DF [45] and the generated DFL. ρ is the weight of $\Gamma_{i,j}$, which is used in DFL. We test ρ with four representative values 0.2, 0.4, 0.6 and 0.8 in this paper. Let DFL_2, DFL_4, DFL_6 and DFL_8 represent DFL with $\rho = 0.2$, $\rho = 0.4$, $\rho = 0.6$ and $\rho = 0.8$, respectively. Specially, DF is the DFL with $\rho = 0$. There are six different deadline divisions. Different task list constructions have an influence on completion times of workflows. We compare WSJT, WIJT, WSJD and WIJD, which are shown in Section 4.3. Various task scheduling strategies result in different completion times of workflows as well. Since the considered transmission times of task are different from existing MapReduce task scheduling, EA and EF [51] are adapted in this paper. Assume REA is the adapted EA with the consideration of replicas. Four task scheduling strategies (EA, EF, REA and the proposed TS) are compared. Note that the proposed TS is the same as the adapted EF with replicas. As a result from all the above options, there are $3\times6\times4\times4 = 288$ treatments in the experimental design of parameters tunning. With the aforementioned three levels of the number of workflows, the number of treatments is up to 864. Five random instances for each treatment are tested and the total number of results in the calibration experiment is 4320. All instances are generated by RanGen [53].

Experimental results are analyzed by the multi-factor analysis of variance (ANOVA) statistical technique. All three main hypotheses (normality, homoscedasticity and independence of the residuals) are checked and accepted during the analysis. According to the analysis of variance (workflow conversion, deadline division, task list construction and task scheduling) for RDI of total profit I, all p-values are less than 0.05 which indicates that all studied factors have a significant effect on the response variable at a 95.0% confidence level. The means plots of workflow conversions and task list constructions with 95.0% Tukey Honest Significant Difference (HSD) intervals are shown in Figure 6.

From Figure 6 (a), RDI of I is minimum with CWDP. The reason lies in that CWDP combines MapReduce jobs by dynamic programming in order to minimize completion times of workflows, which balances the processing times of the new combined MapReduce jobs and the transmission times among jobs. CWC has bigger RDI than CWDP. It groups all combinable MapReduce jobs into a new MapReduce job to reduce transmission times. The combination of CWC ignores considerations of profit, which results in long processing times of the new combined MapReduce jobs and

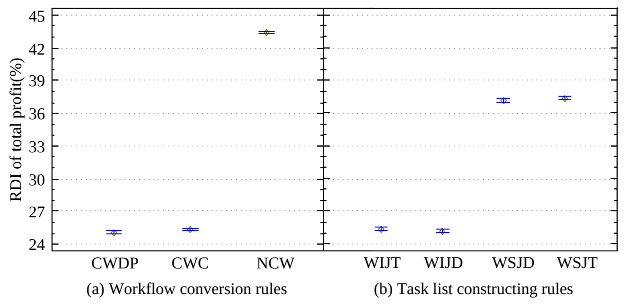

Fig. 6: Means plots of workflow conversions and task list constructions for I with 95.0% Tukey HSD intervals.

nullifies the advantage of combinations. Without doubt, NCW with no combinations leads to long completion times of workflows and less profit for the resource provider. In order to make the comparison of CWDP and CWC clear, a zoomed-in view is shown in Figure 7 (a). From Figure 6 (b), we

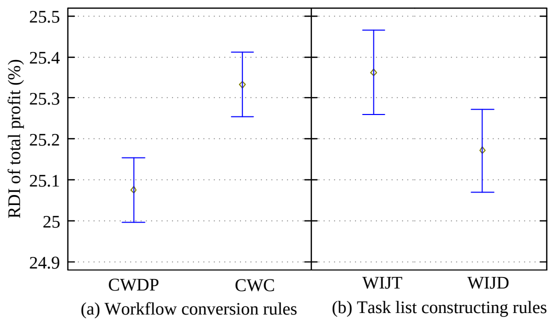

Fig. 7: Means plots of workflow conversions (CWDP and CWC) and task list constructions (WIJT and WIJD) for I with 95.0% Tukey HSD intervals.

can see that two sequential workflow scheduling WSJT and WSJD have the highest RDI, because the sequential processing of workflows makes completion times of workflows too long and violate deadlines and result in a small total profit. Meanwhile the interleaving workflow scheduling performs better than those sequential ones, especially WIJD has less mean value of RDI than WIJT. The reason lies in that WIJD schedules jobs processing in parallel of workflow according to deadlines in order to decrease completion times of workflows. While WIJT just randomly processes these jobs and results in longer completion times and in a worse total profit. The means plots of deadline divisions and task scheduling strategies with 95.0% Tukey HSD intervals are shown in Figure 8. Figure 8 (a) demonstrates that DE has the worst performance. DE obtains the largest RDI because it only considers the execution times, resulting in unreasonable subdeadlines and in the worst total profit. The performance of other five rules have no significant differences, while DFL$_2$ has the minimal mean value of RDI. DFL$_2$

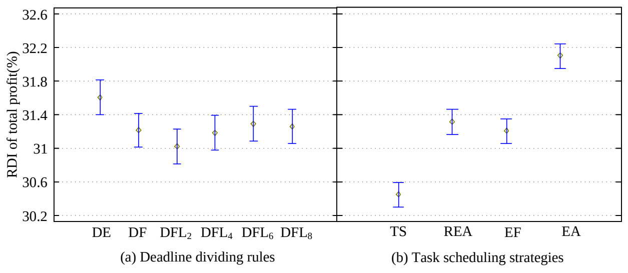

Fig. 8: Means plots of deadline divisions and task scheduling strategies for I with 95.0% Tukey HSD intervals.

divides deadlines of workflows in terms of execution times, float times and locations of MapReduce jobs, which considers both time and space of jobs in critical path to obtain exact subdeadlines. Figure 8 (b) shows that TS statistically outperforms the other three rules. TS considers replica strategy to greatly increase the possibility of tasks being processed by servers holding data in order to decrease transmission times. At the same time, TS selects servers with the earliest finish time of tasks to decrease completion times of workflows. The total profit with REA is less than TS. Even though the replica strategy is adopted by REA, it selects servers with the earliest start time of tasks which cannot ensure small completion times of tasks. Because EF and EA ignore the replica strategy, they also have worse performance. Especially EF has less RDI than EA by assigning tasks to servers with the earliest finish time. As a result from this detailed analysis and calibration, the proposed framework WS with CWDP, DFL$_2$, WIJD and TS will be compared with other existing algorithms in the next section.

5.2 Algorithm comparison

To evaluate performance of the proposed algorithm , four scientific workflow applications (CyberShake, Genome, LIGO and Montage) are adopted. The number of workflow applications takes values from $\{50, 100, 150\}$. The other configurations of workflow instances and tasks are the same as those in the previous section. In existing MapReduce workflow scheduling, MRWS [29] and TCC [4] focus on single MapReduce workflow scheduling and deadlines are not considered. In order to make MRWS and TCC solve the studied problem, they are adapted with the proposed DFL_2 in order to have a fair comparison with the proposed WS. The number of treatments in algorithm comparison is $4 \times 3 \times 3 = 36$. With 5 instances for each treatment, the total number of results is 180.

Figure 9 shows the interactions between the number of workflows and the compared algorithms for RDI with 95% Tukey HSD intervals, where (a), (b), (c) and (d) are the plots for CyberShake, Genome, LIGO and Montage workflow

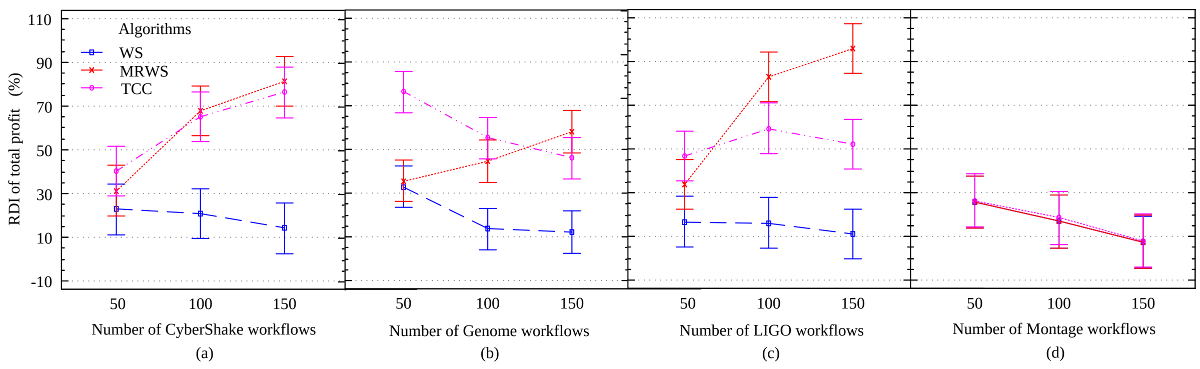

Fig. 9: Interactions between the number of workflows and the compared algorithms for each workflow application for I with 95% Tukey HSD intervals.

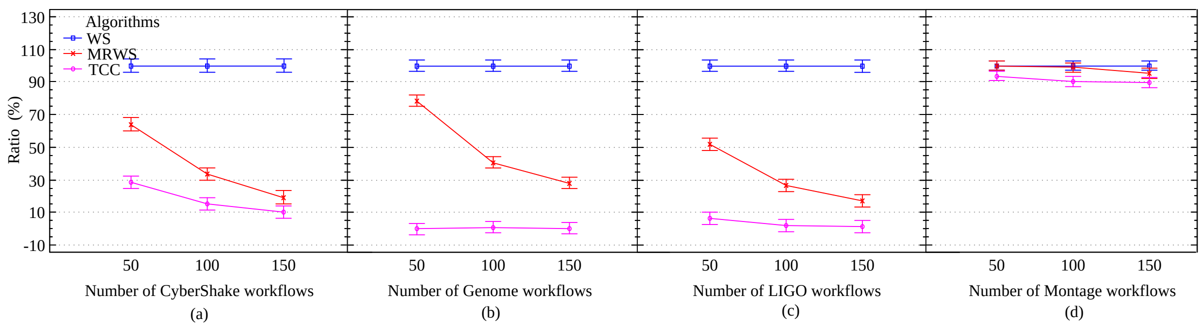

Fig. 10: Interactions between the number of workflows and the compared algorithms for each workflow application for Ratio with 95% Tukey HSD intervals.

instances, respectively. From Figure 9 (a), (b) and (c), WS has the lowest RDI in almost all cases because of the merits of WS using CWDP to convert MapReduce workflows, WIJD to construct the task list and TS to schedule tasks to resources. According to the existing ChainMap/ChainReduce in Hadoop, CWDP converts workflows using Dynamic Programming to balance transmission times among jobs and processing times of new converted jobs. WIJD sequences workflows according to penalty, orders jobs by their divided deadlines and schedules workflows alternatively which guarantees all workflows to be processed in parallel and considers deadlines and penalties of workflows. TS adopts the replica strategy in workflow to improve data locality. It assigns tasks to servers with the earliest finish time to minimize the completion time of workflows. The combination of these strategies increases the total profit of resource providers. MRWS and TCC show no statistically significant differences in most cases. Though they have different task list constructions and task scheduling strategies, the strategy without workflow conversion makes completion times of workflows too long. With the increase of number of workflows, the differences among MRWS and TCC become larger, i.e., the proposed WS has better performance for large number of workflows. There are no significant differences from different compared algorithms in Figure 9 (d). The reason is that Montage workflow instances always have less jobs with linear dependencies. In other words, WS has better performance on CyberShake, Genome and LIGO workflow instances and comparable performance on Montage instances. The comparison results are shown in Table 4 with the best values are bold.

Since deadlines are considered in MapReduce workflow scheduling, we compare the Ratio indicator. Here Ratio is the ratio of the number of workflows completed before the deadlines to the total number of submitted workflows. A higher Ratio implies that more workflows are completed before their deadlines. Figure 10 shows us the interactions between the number of workflows and the compared algorithms for Ratio with 95% Tukey HSD intervals. From Figure 10 (a), (b) and (c), WS obtains the highest Ratio among all compared algorithms. The reason is similar to that for I. MRWS has higher Ratio than TCC. The reason is that MRWS sequences jobs in decreasing order of execution time and transmission time and schedules tasks to servers with the earliest finish time. Both of them are decreased completion times of workflows. While TCC sequences workflows, jobs and tasks randomly, which is not good at the reduction of completion times. In Figure 10 (a), (b) and (c), the

Table 4: Results of the compared algorithms.

Indicators		RDI of total profit												Ratio											
Workflow types		CyberShake			Genome			LIGO			Montage			CyberShake			Genome			LIGO			Montage		
Workflows / Algorithms		50	100	150	50	100	150	50	100	150	50	100	150	50	100	150	50	100	150	50	100	150	50	100	150
WS	1	35.57	37.02	**7.75**	42.59	6.03	15.36	8.33	**6.24**	7.60	27.91	25.34	16.58	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	2	**0.00**	23.13	15.91	37.96	12.82	14.41	28.65	19.99	11.14	31.43	18.11	**0.00**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00
	3	21.83	13.19	11.51	**26.65**	33.49	15.00	**1.14**	25.87	**5.50**	**3.78**	**2.03**	11.40	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	4	29.74	**6.52**	18.11	28.44	15.54	15.32	34.93	10.19	17.18	44.23	9.10	0.01	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	5	26.52	24.32	17.26	29.05	**0.00**	**1.33**	10.75	18.55	14.77	21.35	30.13	8.82	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
MRWS	1	41.11	82.70	77.25	45.64	43.74	59.71	31.16	**69.10**	98.49	27.91	25.34	16.70	0.68	0.27	0.18	0.70	0.35	**0.31**	0.54	0.31	0.16	1.00	1.00	0.97
	2	**8.34**	72.25	88.86	40.89	42.88	62.01	40.33	72.99	100.00	31.43	18.11	0.28	0.62	0.28	0.18	0.86	0.36	0.27	**0.62**	**0.37**	0.17	**1.00**	**1.00**	0.96
	3	26.97	**52.67**	**75.62**	**28.02**	55.52	59.16	**21.10**	100.00	**85.49**	**3.78**	**2.03**	11.41	**0.70**	**0.40**	**0.21**	**0.88**	**0.46**	0.31	0.46	0.23	**0.20**	1.00	1.00	**0.99**
	4	39.36	58.12	80.21	31.82	48.62	61.62	48.59	82.20	98.96	44.23	9.10	**0.08**	0.62	0.34	0.21	0.74	0.41	0.28	0.48	0.23	0.15	1.00	1.00	0.98
	5	41.05	72.76	85.07	32.00	**32.37**	**47.32**	27.67	89.88	96.56	21.35	30.28	9.53	0.58	0.38	0.18	0.74	0.45	0.24	0.50	0.20	0.16	1.00	0.94	0.87
TCC	1	54.11	80.98	76.44	100.00	53.09	49.71	44.72	**44.49**	**41.75**	28.74	25.34	18.11	0.24	0.11	0.11	**0.00**	0.00	**0.01**	0.08	**0.03**	**0.06**	**0.94**	**1.00**	0.89
	2	**16.60**	68.20	77.31	69.56	47.94	43.07	59.74	62.51	54.30	32.21	20.31	0.61	**0.36**	0.12	0.09	0.00	**0.02**	0.00	0.06	0.02	0.01	0.94	0.88	0.91
	3	39.55	**50.61**	75.24	**62.55**	71.23	47.41	31.27	71.69	45.24	**4.41**	**3.89**	12.18	0.22	**0.21**	0.07	0.00	0.00	0.01	0.02	0.03	0.01	0.94	0.84	0.87
	4	45.63	53.65	**74.13**	77.92	56.15	54.61	69.15	54.58	58.31	44.55	11.75	**0.31**	0.36	0.17	**0.12**	0.00	0.00	0.01	0.04	0.01	0.00	0.94	0.90	**0.92**
	5	45.67	72.78	78.29	71.06	**47.30**	**35.36**	**28.58**	63.45	61.40	22.00	31.15	9.15	0.24	0.15	0.12	0.00	0.02	0.00	**0.12**	0.02	0.00	0.92	0.89	0.91

trend of Ratio by MRWS and TCC decreases with an increased number of workflows, while that of WS has no significant differences. In other words, the performance of WS is robust on CyberShake, Genome and LIGO workflow instances. Similar to the total profit, the Ratio for algorithms shows no significant differences in Figure 9 (d) for Montage instances. The results in Table 4 is in accordance with the trends shown in Figure 10.

6 Conclusions and future work

In this paper, MapReduce workflow scheduling with deadline and data locality is studied to maximize the total profit of the resource provider. In the generated MapReduce workflow scheduling framework, a new workflow conversion based on Dynamic Programming and a new deadline division considering job level and float time are produced to better meet deadlines. Meanwhile, a new task scheduling with replica strategy is designed to improve data locality. A number of MapReduce workflows are converted by Dynamic Programming according to ChainMap/ChainReduce in order to decrease transmission times among jobs. In terms of execution times, float times and job level, deadlines of the converted workflows are divided into subdeadlines of jobs. In the new task scheduling, the replica strategy is adapted to decrease transmission times of input and output. With the constructed task list, servers with the earliest finish time are chosen to process tasks in order to decrease completion times and to increase total profit. Experimental results show that the total profit with the proposed strategy is higher than that with other compared algorithms.

Generally speaking, different types of big data applications have special computing frameworks. For example, MapReduce is suitable for batch-like applications and Spark for real time applications. In other words, several computing frameworks are run in a cloud data center to increase resource utilization. Different types of big data applications have different resource requirements, servers allocated to different computing frameworks always have uneven resource utilization. There are still some problems have not been solved in this paper, such as resource management, data placement and initial placement. These problems with more workflow scheduling problems in popular models (e.g., in Spark) are worth studying in the future.

Acknowledgements This work is supported by the National Key Research and Development Program of China (No. 2017YFB1400801), the National Natural Science Foundation of China (Nos. 61872077, 61832004) and Collaborative Innovation Center of Wireless Communications Technology. Rubén Ruiz is partly supported by the Spanish Ministry of Science, Innovation, and Universities, under the project "OPTEP-Port Terminal Operations Optimization" (No. RTI2018-094940-B-I00) financed with FEDER funds"..

References

1. Zaharia M, Chowdhury M, Franklin M et al (2010) Spark: cluster computing with working sets. In: Usenix Conference on Hot Topics in Cloud Computing, pp 1765–1773

2. Li L, Ma Z, Liu L et al (2013) Hadoop-based ARIMA algorithm and its application in weather forecast. International Journal of Database Theory & Application 6(5):119–132
3. Xun Y, Zhang J, Qin X (2017) FiDoop: Parallel Mining of Frequent Itemsets Using MapReduce. IEEE T SYST MAN CY-S 46(3):313–325
4. Wang Y, Shi W (2014) Budget-driven scheduling algorithms for batches of MapReduce jobs in heterogeneous clouds. IEEE T CLOUD COMPUT 2(3):306–319
5. Tiwari N, Sarkar S, Bellur U et al (2015) Classification framework of MapReduce scheduling algorithms. ACM COMPUT SURV 47(3):1–49
6. Dong X, Wang Y, Liao H (2011) Scheduling mixed real-time and non-real-time applications in MapReduce environment. In: International Conference on Parallel and Distributed Systems, pp 9–16
7. Tang Z, Zhou J, Li K et al (2013) A MapReduce task scheduling algorithm for deadline constraints. CLUSTER COMPUT 16(4):651–662
8. Zhang W, Rajasekaran S, Wood T et al (2014) MIMP: deadline and interference aware scheduling of Hadoop virtual machines. In: International Symposium on Cluster, Cloud and Grid Computing, pp 394–403
9. Teng F, Magoulès F, Yu L et al (2014) A novel real-time scheduling algorithm and performance analysis of a MapReduce-based cloud. J SUPERCOMPUT 69(2):739–765
10. Palanisamy B, Singh A, Liu L (2015) Cost-effective resource provisioning for MapReduce in a cloud. IEEE T PARALL DISTR 26(5):1265–1279
11. Hashem I, Anuar N, Marjani M et al (2017) Multi-objective scheduling of MapReduce jobs in big data processing. MULTIMED TOOLS APPL 1–16
12. Xu X, Tang M, Tian Y (2017) QoS-guaranteed resource provisioning for cloud-based MapReduce in dynamical environments. FUTURE GENER COMP SY 78(1):18–30
13. Li H, Wei X, Fu Q et al (2014) MapReduce delay scheduling with deadline constraint. CONCURR COMP-PRACT E 26(3):766–778
14. Polo J, Becerra Y, Carrera D et al (2013) Deadline-based MapReduce workload management. IEEE T NETW SERV MAN 10(2):231–244
15. Chen C, Lin J, Kuo S (2018) MapReduce scheduling for deadline-constrained jobs in heterogeneous cloud computing systems. IEEE T CLOUD COMPUT 6(1):127–140
16. Kao Y, Chen Y (2016) Data-locality-aware MapReduce real-time scheduling framework. J SYST SOFTWARE 112:65–77
17. Bok K, Hwang J, Lim J et al (2017) An efficient MapReduce scheduling scheme for processing large multimedia data. MULTIMED TOOLS APPL 76(16):1–24
18. Chen Y, Borthakur D, Borthakur D et al (2012) Energy efficiency for large-scale MapReduce workloads with significant interactive analysis. In:ACM European Conference on Computer Systems, pp 43–56
19. Mashayekhy L, Nejad M, Grosu D et al (2015) Energy-aware scheduling of MapReduce jobs for big data applications. IEEE T PARALL DISTR 26(10):2720–2733
20. Lei H, Zhang T, Liu Y et al (2015) SGEESS: smart green energy-efficient scheduling strategy with dynamic electricity price for data center. J SYST SOFTWARE 108:23–38
21. Bu Y, Howe B, Balazinska M et al (2012) The HaLoop approach to large-scale iterative data analysis. VLDB J 21(2):169–190
22. Zhang Y, Gao Q, Gao L et al (2012) iMapReduce: a distributed computing framework for iterative computation. J GRID COMPUT 10(1):47–68
23. Gunarathne T, Zhang B, Wu T et al (2013) Scalable parallel computing on clouds using Twister4Azure iterative MapReduce. FUTURE GENER COMP SY 29(4):1035–1048
24. Oliveira D, Ocana K, Baiao F et al (2012) A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. J GRID COMPUT 10(3):521–552
25. Li S, Hu S, Abdelzaher T (2015) The packing server for real-time scheduling of MapReduce workflows. In: IEEE Real-Time and Embedded Technology and Applications Symposium, pp 51–62
26. Cai Z, Li X, Ruiz R et al (2017) A delay-based dynamic scheduling algorithm for bag-of-task workflows with stochastic task execution times in clouds. FUTURE GENER COMP SY 71:57–72
27. Cai Z, Li X, Ruiz R (2017) Resource provisioning for task-batch based workflows with deadlines in public clouds. IEEE T CLOUD COMPUT https://doi.org/10.1109/TCC.2017.2663426
28. Cai Z, Li X, Gupta J (2016) Heuristics for provisioning services to workflows in XaaS clouds. IEEE T SERV COMPUT 9(2):250–263
29. Tang Z, Liu M, Ammar A et al (2014) An optimized MapReduce workflow scheduling algorithm for heterogeneous computing. J SUPERCOMPUT, 72(6):1–21
30. Xu C, Yang J, Yin K et al (2017) Optimal construction of virtual networks for cloud-based MapReduce workflows. COMPUT NETW 112:194–207
31. Chiara S, Danilo A, Gianpaolo C et al (2013) Optimizing service selection and allocation in situational computing applications. IEEE T SERV COMPUT, 6(3):414–428
32. Baresi L, Elisabetta D, Carlo G et al (2007) A framework for the deployment of adaptable web service compositions. Service Oriented Computing and Applications, 1(1):75–91
33. Lim H, Herodotou H, Babu S (2012) Stubby: a transformation-based optimizer for MapReduce workflows. VLDB Endowment 5(11):1196-1207
34. Chowdhury M, Zaharia M, Ma J et al (2011) Managing data transfers in computer clusters with orchestra. ACM SIGCOMM COMP COM 41(4):98–109
35. Ke H, Li P, Guo S et al (2016) On traffic-aware partition and aggregation in MapReduce for big data applications. IEEE T PARALL DISTR 27(3):818–828
36. Yu W, Wang Y, Que X et al (2015) Virtual shuffling for efficient data movement in MapReduce. IEEE T COMPUT 64(2):556–568
37. Guo D, Xie J, Zhou X et al (2015) Exploiting efficient and scalable shuffle transfers in future data center network. IEEE T PARALL DISTR 26(4):997–1009
38. Li D, Yu Y, He W et al (2015) Willow: saving data center network energy for network-limited flows. IEEE T PARALL DISTR 26(9):2610–2620
39. Tan J, Meng X, Zhang L (2013) Coupling task progress for MapReduce resource-aware scheduling. In:IEEE INFOCOM, pp 1618–1626
40. Hammoud M, Rehman M, Sakr M (2012) Center-of-gravity reduce task scheduling to lower MapReduce network traffic. In: International Conference on Cloud Computing, pp 49–58
41. Guo Z, Fox G, Zhou M et al (2012) Improving resource utilization in MapReduce. In: International Conference on Cluster Computing, pp 402–410
42. Fischer M, Su X, Yin Y (2010) Assigning tasks for efficiency in Hadoop. In:Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures, pp 30–39
43. Zhu Y, Jiang Y, Wu W et al (2014) Minimizing makespan and total completion time in MapReduce-like systems. In: IEEE INFOCOM, pp 2166–2174
44. Kavulya S, Tan J, Gandhi R et al (2010) An analysis of traces from a production MapReduce cluster. In: IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp 94–103
45. Li X, Cai Z (2017) Elastic resource provisioning for cloud workflow applications. IEEE T AUTOM SCI ENG 14(2):1195–1210
46. Abrishami S, Naghibzadeh M, Epema D (2013) Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service clouds. FUTURE GENER COMP SY 29(1):158–169
47. Fernando B, Edmundo R (2010) Towards the scheduling of multiple workflows on computational grids. J GRID COMPUT 8(3):419–441

48. Tiwari N, Sarkar S, Bellur U et al (2015) Classification framework of MapReduce scheduling algorithms. ACM COMPUT SURV 47(3):1–38
49. Verma A, Cherkasova L, Campbell R (2013) Orchestrating an ensemble of MapReduce jobs for minimizing their makespan. IEEE T DEPEND SECURE 10(5):314–327
50. Heintz B, Chandra A, Sitaraman R et al (2017) End-to-end optimization for geo-distributed MapReduce. IEEE T CLOUD COMPUT 4(3):293–306
51. Li X, Jiang T, Ruiz R (2016) Heuristics for periodical batch job scheduling in a MapReduce computing framework. INFORM SCIENCES 326:119–133
52. Chen L, Li X (2018) Cloud workflow scheduling with hybrid resource provisioning. J Supercomput 74(12):6529–6553
53. Vanhoucheabcd M, Maenhout B, Tavares L (2008) An evaluation of the adequacy of project network generators with systematically sampled networks. EUR J OPER RES 187(2):511–524