



# Energy- and locality-efficient multi-job scheduling based on MapReduce for heterogeneous datacenter

Lei Chen<sup>1</sup> · Zhao-Hua Liu<sup>2</sup>

Received: 12 July 2019 / Revised: 5 August 2019 / Accepted: 13 August 2019 / Published online: 22 August 2019  
© Springer-Verlag London Ltd., part of Springer Nature 2019

## Abstract

Job scheduling of MapReduce is a research hot spot, especially on the heterogeneous datacenter. Huge energy consumption and operating costs are key challenges. Most of the previous work only considers the scheduling optimization of a single job. In this paper, we take multiple jobs of MapReduce as research objects and focus on the goal of “jointly optimizing the scheduling time, job costs and energy consumption.” For that, an energy- and locality-efficient MapReduce multi-job scheduling algorithm is developed for the heterogeneous datacenter. Firstly, we use rack as the basic unit of resource in job scheduling to reduce data communication between jobs and to facilitate energy savings. Secondly, according to the capacity of heterogeneous rack, we design a multi-job pre-mapping method to optimize the execution order of jobs and jointly optimize the scheduling time, job costs and energy consumption. Based this pre-mapping method, we can assign one job to the virtual machine on the same rack, so as to minimize the amount of online rack. This centralized mapping strategy is very helpful to save energy and reduce data transmission of jobs. Thirdly, the map and reduce tasks of a job will be divided into multiple task groups for parallel execution, thereby further reducing data communication and energy consumption. Finally, a lot of experimental results prove the advantages of our algorithm.

**Keywords** Energy-saving · Data locality · MapReduce · Heterogeneous datacenter

## 1 Introduction

In big data environment, how to process the massive data [1] is becoming a hot spot, which has brought great challenge and opportunity for industry and academia. MapReduce [2], proposed by Google in 2004, is a distributed parallel data programming framework which has some prominent features, such as flexibility, open source, scalability and robustness. Due to the huge demand for MapReduce, the traditional small cluster is powerless and a large number of physical resources are renting from multiple heterogeneous datacenters to build the advanced MapReduce platform in the cloud [3] (Cloud MapReduce, simply called CMR), which provides

the MapReduce service on a “pay-as-you-go” model. In MapReduce service, Push, Map and Reduce are three main phases. Push phase is responsible for splitting large-scale data into fixed-size blocks. Map phase parallel processes data blocks to generate the intermediate data. Reduce phase handles and merges the intermediate data to form the final data results.

In the heterogeneous datacenter, MapReduce scheduling has the following new challenges:

- *Resource heterogeneous* In the past works, the resources of datacenter are considered to be homogeneous. That is, all nodes are configured with the same RAM, CPU, and DISK in datacenter. However, this is not in line with reality. The servers or virtual machines always have different RAM, CPU, and DISK in the heterogeneous datacenter.
- *Rack-level management* Rack contains a group of servers with the same properties (CPU, RAM, and DISK). All servers in the same rack are connected to the same network switch (same bandwidth) and public storage. That is to say, rack is the base management of the heterogeneous datacenter.

✉ Lei Chen  
chenlei@hnust.edu.cn

Zhao-Hua Liu  
163liuzhaohua@163.com

<sup>1</sup> School of Information and Electrical Engineering, Hunan University of Science and Technology, Xiangtan, China

<sup>2</sup> State Key Laboratory of Advanced Design and Manufacturing for Vehicle Body, Hunan University, Changsha, China

- *Energy consumption* Usually, the heterogeneous data-center rented lots of heterogeneous servers for providing multiple services. But, when the amount of online jobs is small, lots of physical hosts will be idle, and a large amount of energy will be wasted.
- *Multi-job influence* The traditional cluster usually executes little jobs with specific users. However, a large number of jobs could be submitted in batches by multiple users online in the heterogeneous datacenter. So, a fast and reasonable job scheduling strategy is urgently needed.

In the past few years, lots of works about optimizing MapReduce job scheduling have been developed. These works are mainly divided into the following three categories: energy-saving optimization, task-mapping optimization, and data locality optimization. For energy-saving optimization [4–8], SLA, hardware resource utilization, energy budget, and data migration are considered to reduce energy consumption. However, the related balance studies simultaneously considering job cost, execution time, and energy consumption are still very few. For task-mapping optimization, authors [9–14] try to reduce the scheduling number of the map and reduce tasks to optimize the scheduling time and job cost. Works in [15–17] use task delay strategy to improve task assignment of the heterogeneous environment. For data locality optimization, some data placement methods are used in works [18–20] to enhance the data locality for reducing the communication from immediate data of Map tasks to Reduce tasks. Authors [21,22] reduce task execution time by improving data management and migration.

Based on the above analysis, the motivation of this paper is to exploit the features of heterogeneous datacenter to optimal job scheduling for jointing optimizing the job cost, energy, and scheduling time. To achieve the above motivation, based on the heterogeneous datacenter environment, we see the rack as the basic unit of resource for job scheduling, and an energy- and locality-efficient MapReduce multi-job scheduling algorithm is proposed, called TPJS. TPJS firstly measures the capacity of one rack for different jobs from the scheduling time, energy, and execution cost. Secondly, a multi-job pre-mapping method is developed to dynamically adjust job assignment order for reducing online resource. Finally, after multi-job pre-mapping, a parallel task execution method is used to further enhance the data locality and reduce the data communication from immediate data of map tasks to corresponding reduce tasks by reduce task mapping.

In summary, based on the capacity of rack, multi-job scheduling process is divided into two phases: multi-job pre-mapping and parallel job execution. In the first phase, multiple jobs are merged into job group to adjust assignment order for improving resource utilization. And each job in a group is centrally pre-mapping to multiple booked racks for

decreasing the energy and data transmission. In the second phases, each reduce task of one job is mapped to multiple map tasks to form a task group and all virtual machines of booked racks parallel execute map tasks. After the map tasks are completed, reduce task would try to be assigned to the same virtual machine which executes map tasks with the same group for further enhancing data locality. So, the main contributions of this paper are as follows:

- (1) A multi-job pre-mapping method is developed to divide the multiple jobs into job groups, so as to improve the execution order of jobs and increase the resource utilization. By using the multi-job pre-mapping, one job in a group is centrally mapped to virtual machines located in the same rack. This mapping straggly can reduce the data transmission of jobs and save energy.
- (2) A parallel task execution method is used to further enhance the data locality where one reduce task is mapped to multiple map tasks to form task group, and all tasks of the same task group try to be executed in same virtual machines.

The remaining content of this paper is organized as follows: Section 2 introduces the related works. The problem of job scheduling in the heterogeneous datacenter and corresponding model are described in Sect. 3. The energy- and locality-efficient MapReduce multi-job scheduling algorithm (TPJS) is presented in Sect. 4. Section 5 demonstrates the experimental evaluation. Finally, Sect. 6 concludes the work of this paper.

## 2 Related works

Currently, MapReduce scheduling problem has attracted the interest of many scholars and a large number of outstanding results have been achieved. According to optimization objectives, these works mainly focus on three categories: energy-saving optimization, task-mapping optimization and data locality optimization.

*Energy-saving optimization* An energy-efficient framework is designed in [4] to improve the energy consumption and satisfy the SLA for job scheduling of MapReduce. Bampis [5] proposes a polynomial-time constant-factor approximation algorithm to minimize the total weighted completion time of a set of MapReduce jobs under a given budget of energy. Two heuristic energy-aware task scheduling strategies are designed in [6] for improving the data locality and resource utilization. Maheshwari [7] proposes an energy-efficient data placement and cluster reconfiguration algorithm to cut down operational costs and reduce their carbon footprint. This algorithm dynamically reconfigures the cluster based on the current workload and turns cluster nodes on or off when the average cluster utilization rises above or

falls below administrator-specified thresholds, respectively. In addition, an energy-efficient MapReduce workload manager is designed in [8] to improve the hardware resource utilization. However, little work on balancing multiple objectives (data placement, task scheduling, and energy) of job scheduling for MapReduce exists at present.

**Task-mapping optimization** Task placement optimization is another key research topic for MapReduce scheduling. The main work on this topic is to optimize the assignment process among map and reduce tasks. Palanisamy et al. [9] design, Purlieusa, a MapReduce resource allocation system, and the basic idea is to allocate the map and reduce tasks to the nearby VMs for enhancing the data locality and reducing network traffic overhead. Based on the bipartite graph-matching model, a new scheduling system, BGMRS, is designed in [10] which is a good solution for the slot performance heterogeneous and job time variation problem. Tang et al. [11] propose an adaptive scheduling optimization algorithm SARS. It firstly evaluates the context of each job (task completion time and output size in map phase) and dynamically adjusts the start time of reduce phase for reducing job execution time. Some works on makespan optimization [12–14] have been developed to reduce the job executing time by different task assignment strategies. Cura [15] develops a cost optimization MapReduce framework to save MapReduce service costs in the cloud environment. Heintz et al. [16] summarize phases in MapReduce scheduling process and propose an across-phase optimal scheduling method where some tasks is overlapped execution in the four phases to reduce the whole job execution time. Author [17] further analyses the connection of the four phases of MapReduce job scheduling process and designs corresponding optimal scheduling method to enhance the execution speed. However, all the above works focus on the scheduling of a single job, which does not consider scheduling process among multiple jobs. Moreover, in our works, multiple targets (data placement, task scheduling, and energy) are considered as the goal of job scheduling for multiple jobs.

**Data locality optimization** Data locality is a hot research topic, and a large number of algorithms have been proposed to optimize job scheduling performance of MapReduce. Based on Hadoop cluster, a data placement strategy for data-sensitive applications has been proposed [20] where all data blocks are assigned to each node in a reasonable and balanced way for enhancing the performance of data processing. For common distributed environment, an ADAPT data placement algorithm has been proposed by Jin [19]. The basic idea of this work is to use a prediction mechanism to place data blocks without backup, and the results show that ADAPT reduces network traffics and improves process performance. Oscar Boykin et al. [18] design a new MapReduce framework (MRA++) for the heterogeneous and distribution datacenter. MRA++ considers the data placement and task

scheduling problem together, and a series of algorithms were proposed to optimal traditional MapReduce scheduling process. Papers [21–23] summarize the data management works under the MapReduce framework and point out new problems and challenges. However, the above works mainly focus on the local-aware data placement optimization for the single job. In our work, local-aware and energy-aware are both considered for job scheduling optimization. Moreover, multiple jobs are seen as a group to further optimize the scheduling process.

In summary, there are still a lot of challenges in heterogeneous datacenter environments, especially the balance among data locality, energy consumption and job cost in multiple jobs scheduling. Therefore, this paper focuses on the multiple MapReduce jobs optimization for balancing the data locality, energy consumption and job cost.

### 3 Problem analysis and model

In this section, the problem analysis and corresponding model of job scheduling in heterogeneous datacenter are presented. The common architecture of heterogeneous datacenter is firstly introduced in Sect. 3.1. The problem statement of job scheduling is shown in Sect. 3.2, and the corresponding model is built in Sect. 3.3.

#### 3.1 Heterogeneous datacenter architecture

Heterogeneous datacenter is a pooling of large number of heterogeneous resources, such as servers, storage and network. Heterogeneous datacenter could provide different services of storage, hardware resource and multiple applications for different areas of society. Figure 1 shows the common architecture of the heterogeneous datacenter, where servers, virtual machines, storage and network are four core components.

**Servers** Heterogeneous datacenter usually has a large number of heterogeneous physical servers. Multiple physical servers are assembled to one rack. Multiple racks are connected by high network. Generally, multiple physical servers of one rack are homogeneous and have same profiles. In addition, all servers of one rack are in a local network and are connected via high-performance switches. And all servers of one rack share a public power switch to turn on and off more easily. In the heterogeneous datacenter, each server configures multiple resources with CPU, MEM and a small amount of local disk storage. In order to write and read the massive data, each server is connected to a common storage by using the SAN rack card.

**Virtual machines** In order to manage resources more conveniently and improve the resource utilization, the virtual technology is used in heterogeneous datacenter. That is to say,

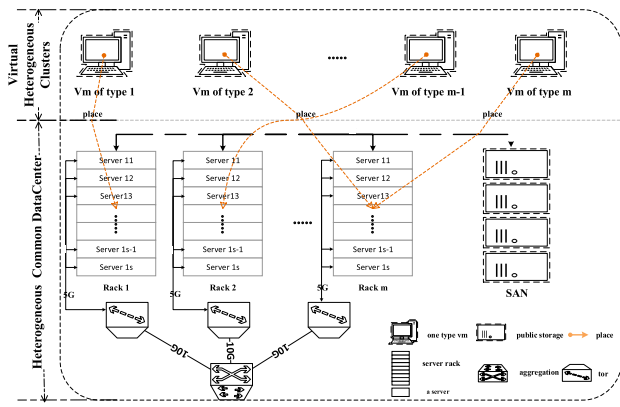


Fig. 1 Common architecture of the heterogeneous datacenter

a large number of heterogeneous virtual machines are flexibly generated and managed according to the user demand. These virtual machines share hardware resources of physical servers, such as network, storage, CPU, and MEM. In addition, heterogeneous virtual machines have different configurations of CPU, MEM, and storage. In the real world, to facilitate management of virtual machines, all virtual machines of one rack usually are homogeneous.

**Storage** Due to the storage need for large dataset, local storage and public storage are used together in the datacenter. Local storage, the local disk, is integrated with the physical hosts to satisfy the running demand. The capacity of local storage is usually small and very difficult for expansion due to the expensive price. Public storage consists of a lot of storage devices where all storage devices are connected via fiber optic network. The capacity of public storage is very huge and of low cost. In additional, any physical host could be connected to public storage via fiber optic network. Therefore, public storage has prominent advantages in scalability, compatibility, capacity, and other characteristics.

**Network** Network is a very important component in the heterogeneous datacenter, and all physical hosts and racks are connected with each other by network devices. Generally, tree (or binary tree) network topology is a common network structure in the datacenter, such as Fat Tree, Portland and VL2. From down- to up-prospective, each rack has a high-performance local switch, and multiple racks are connected by an aggregation layer switches in whole network topology.

### 3.2 Problem statement

The process of MapReduce scheduling problem on heterogeneous datacenter is as follows: CMR platform has a lot of racks, each rack includes many physical hosts, and each host builds multiple virtual machines. Multiple MapReduce jobs  $J = (job_1, \dots, job_n)$  are submitted to CMR plat-

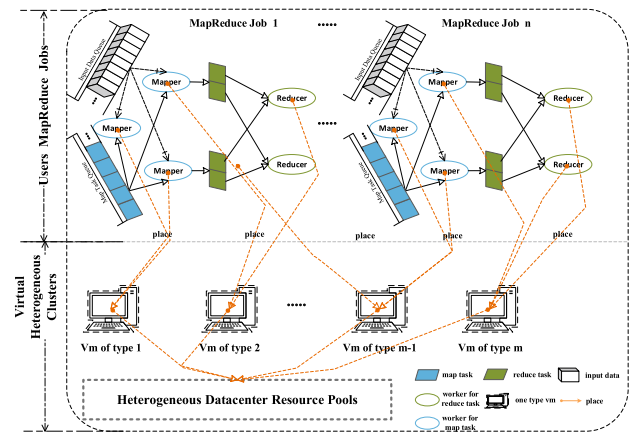


Fig. 2 Scheduling process of MapReduce job

form, and the number of job list  $J$  is  $N$ . Each  $job_i$  contains three task sets: data task, map task and reduce task. The data task is responsible for dividing input data into fixed-size blocks; the map task computes the content of the data block; and the reduce task summarizes the results of map task. Moreover, each job has a different resource request  $Req_i = (req\_cpu, req\_mem, req\_disk)$ , including MEM, CPU, and DISK. Based on this request, CMR platform will select many virtual machines  $VS = (vm_1, \dots, vm_k)$  to execute the job. The virtual machines on CMR platform are also heterogeneous. Each virtual machine  $vm_k$  has different resource  $Cab_k = (cab\_cpu, cab\_mem, cab\_disk, cab\_en)$ , including MEM, CPU, power, DISK, and rack information. The process of MapReduce scheduling is described in Fig. 2.

### 3.3 Model

Based on the problem statement, one MapReduce job is mapped to virtual machines of  $J$  racks to execute. In order to jointly improve the data transmission, execution time, and energy, we need to build a scheduling model to minimize the job cost. Let's introduce a few definitions first.

**Definition 1** A 0–1 variable  $a_{ij}$  is designed to represent whether one MapReduce job is mapped to one rack:

$$a_{ij} = \begin{cases} 1, & job_i \text{ is assigned to rack } j \\ 0, & job_i \text{ is not assigned to rack } j \end{cases} \quad (1)$$

**Definition 2** For one job scheduling, four cost coefficients of resource waste, rent cost, resource balance, and energy consumption are defined:



$$res\_waste_{ij} = \begin{cases} idle\_vm\_size_j \\ -\frac{task\_surplus\_size_i}{task\_vm\_size_{ij}}, & \text{if } res\_waste_{ij} > 0 \\ 0, & \text{if } res\_waste_{ij} \leq 0 \end{cases} \quad (2)$$

$$res\_rent_{ij} = C_j \cdot t_j \quad (3)$$

$$res\_balance_{ij} = \left| \frac{req\_cpu_i}{cab\_cpu_j} - \frac{req\_mem_i}{cab\_mem_j} \right| \quad (4)$$

$$res\_en_{ij} = idle\_vm\_size_j \cdot t_j \cdot power_j \quad (5)$$

Equation 2 is the cost coefficient of resource waste, which indicates the idle number of virtual machines in one online rack, in which  $task\_vm\_size_{ij}$  means the number of tasks of  $job_i$  that a virtual machine of  $rack_j$  can perform,  $idle\_vm\_size_j$  represents the amount of idle virtual machines of  $rack_j$ , and  $task\_surplus\_size_i$  indicates the number of tasks remaining in the current job.

Equation 3 is the cost coefficient of rent cost, where  $t_j$  is the job scheduling time in  $rack_j$  and  $C_j$  is cost fee per millisecond.

Equation 4 is the cost coefficient of resource balance, which represents the balance degree between job requests and virtual machine capacity. The higher the resource balance degree is, the better the utilization of virtual machines in one rack is, and vice versa.

Equation 5 is the cost coefficient of energy consumption, where  $t_j$  is the job scheduling time in  $rack_j$ ,  $power_j$  is the energy usage per millisecond and  $idle\_vm\_size_j$  is the amount of virtual machine in  $rack_j$ .

Therefore, the job optimization scheduling model can be constructed as follows:

$$\begin{aligned} & \min \sum_{i=1}^N \sum_{j=1}^J \left( res\_waste_{ij} + res\_balance_{ij} + res\_rent_{ij} + res\_en_{ij} \right) \cdot a_{ij} \\ & st. \begin{cases} Cab \langle cab\_cpu, cab\_mem, cab\_disk \rangle \\ \geq Req \langle req\_cpu, req\_mem, req\_disk \rangle \\ \\ \sum idle\_vm\_size_j * task\_vm\_size_{ij} \\ \geq task\_surplus\_size_i \end{cases} \end{aligned} \quad (6)$$

### 4 Our algorithm (TPJS)

In this section, an energy- and locality-efficient MapReduce multi-job scheduling algorithm is proposed to finish multiple jobs execution for balancing the performance between job cost, scheduling time, and energy usage in the heterogeneous datacenter, called TPJS.

### 4.1 Rack capacity measure

Rack is the basic cell of resource management in heterogeneous datacenter and very important for job scheduling. Rack is a group of homogenous physical hosts and configures with the same CPU, RAM, and DISK. Firstly, all physical hosts are connected usually to one power switch. Secondly, all servers are connected to the same high-performance network switch. The data communication speed between multiple physical hosts of one rack is very fast. Thirdly, all servers simultaneously share the same storage content. Therefore, when we see the rack as the resource mapping unit, the job execution speed and scheduling time will be improved.

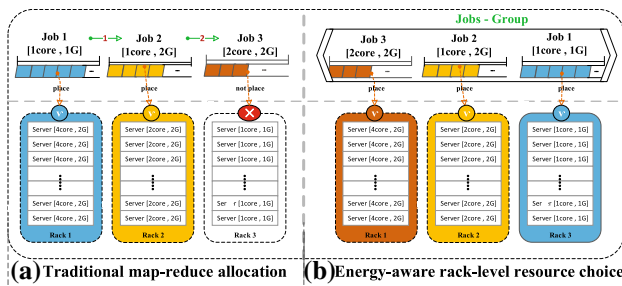
Based on the advantages of rack in the management, we developed a rack capacity measure algorithm in our past work [24]. In this work, we use the measure algorithm to obtain the best ability of one rack for different jobs. In the following, we introduce the basic ideas of rack capacity measure algorithm. The more detailed content can be found in our past work [24].

In order to jointly optimize the job cost, scheduling time, and energy usage, an energy efficiency is introduced to measure whether task allocation of the current rack is the best task allocation for the whole job. If the task allocation of each rack is the best task allocation, then the job cost, scheduling time, and energy usage are the best for the whole job. Because the time and used virtual machine number determine the energy consumption, other more obvious energy term is not considered. Based on the definition of energy efficiency, it is clear that the smaller the energy efficiency is, the better the task allocation will be, and the stronger the capacity of rack for this job will be. Therefore, the energy efficiency is defined as follows:

$$eff_{ij} = \left( \left| \frac{(t_j/task\_vm\_size_j)}{t_i} \right| - \left| \frac{(task_j/vm\_size_j)}{task_i} \right| \right)^2 \quad (7)$$

where  $t_i$  is the final scheduling time of  $job_i$ ,  $t_j$  is the task time of  $rack_j$ ,  $task_i$  is the number of tasks in  $job_i$ ,  $task_j$  is the number of online tasks in  $rack_j$ ,  $vm\_size_j$  is the number of virtual machines in  $rack_j$ , and  $task\_vm\_size_j$  is the average task number of each virtual machine execution.

Based on the energy efficiency, the process of rack capacity measure algorithm is as follows. Firstly, a historical library is built to store the historical best capacity of rack for different jobs. Secondly, based on a random strategy, the rack capacity is judged to be whether it needs to be adjusted at the process of job execution. If the rack capacity needs to be adjusted, then the task number of the rack would be randomly increased or decreased. Otherwise, the rack capacity is not



**Fig. 3** Comparison between traditional job assignment and multiple jobs pre-assignment

changed. Finally, when the job finished, the energy efficiency is computed based on scheduling time and the number of virtual machines used. And update the rack capacity when the energy-efficient value is smaller than the historical value.

## 4.2 Rack-level multi-job pre-mapping

Based on the ability of one rack for different jobs, a rack-level multi-job pre-mapping algorithm is proposed to jointly improve the job scheduling time, job cost, and energy usage. The goal of our multi-job pre-mapping algorithm mainly contains two aspects: Firstly, by dynamically adjusting job execution order, it is able to improve resource utilization and avoid the resource waste; secondly, by centrally assigning jobs to virtual machines of several racks, it is able to reduce data communication and energy consumption. So, before describing our algorithm, let us firstly introduce the resource waste problem from the traditional “first-come-first-served” scheduling model, please see Fig. 3.

In the above figure, a job scheduling scene is presented where three jobs ( $job_1$ ,  $job_2$ ,  $job_3$ ) with different requests need to be mapped to virtual machines of three different racks ( $rack_1$ ,  $rack_2$ , and  $rack_3$ ) for execution. The traditional “first-come-first-served” job scheduling model is shown in Fig. 3a, where the  $job_1$  is firstly executed,  $job_2$  is next, and  $job_3$  is computed at the end. In this scheduling process, in order to enhance execution speed,  $job_1$  selects the virtual machines of  $rack_1$  for execution. Similar,  $job_2$  selects the virtual machines of  $rack_2$  (or  $rack_3$ ) for execution because the capacity of  $rack_2$  is equal to that of  $rack_3$ . When the  $job_3$  starts assignment, because the job request [2core, 2G] of  $job_3$  is greater than the capacity of  $rack_3$  [1core, 1G], the job is not executed until the  $job_1$  is completed. This is the resource waste problem from the traditional “first-come-first-served” job scheduling model. However, if we dynamically adjust the job execution order ( $job_3$  is firstly executed,  $job_1$  is next, and  $job_2$  is final), above resource waste problem is naturally solved. Based on this idea, the job scheduling process of our multi-job pre-mapping algorithm is

### Algorithm 1: *multiple\_jobs\_premapping(J)* //A rack level multi-job pre-mapping algorithm.

**Input:** all job list  $J$ .

**Output:** the assignment results.

```

1 search the idle virtual machines to form the rack groups  $RL$  and
  sort the rack groups with the number of idle virtual machines in
  each rack group.
2 //adjust job execution orders
3 select multiple jobs  $MJ$  from queued job list  $J$  to merge into a
  job group, according to the number of idle resources, and adjust
  the execution order of all jobs in the job group.
4 //job pre-assignment
5 for (each  $job_i$  in  $MJ$ ) do
6   resort the rack groups  $RL$  according to the resource balance
   value between  $job_i$  and each rack by using Equation (3).
7   for (each rack group  $rack_j$  in  $RL$ ) do
8     if ( $rack_j$  is not booked) then
9       get the job request information ( $req\_cpu$ ,  $req\_mem$ )
10      and the property of  $rack_j$  ( $cab\_cpu$ ,  $cab\_mem$ ).
11      compute the best capacity ( $task\_vm\_size_{ij}$ ) of  $rack_j$ 
12      for  $job_i$ .
13      calculate the resource waste  $res\_waste$  according to
14      Equation (2).
15      if ( $0 \geq res\_waste$ ) then
16        the  $rack_j$  is booked for  $job_i$ .
17      else
18        the rack with smallest  $res\_waste$  is booked for
19        the remaining tasks of  $job_i$ .
20
21 return the final pre-assignment results.;
```

presented in Fig. 3b. In our algorithm, multiple jobs are firstly merged into a job group and dynamically adjust job execution order according to the job request. Based on the new job execution order, let rack select the right job for improving the resource utilization. So, in the new job scheduling process,  $job_3$  is executed firstly, then  $job_2$ , finally  $job_1$ . By the above adjustment, three jobs can be instantly assigned for improving the resource utilization and reducing job execution time.

The basic idea of our algorithm is very simple. Multiple jobs are firstly selected to merge into a job group and adjust the job execution order by the job request. According to the new executive order, each job in the group is centrally pre-mapped to virtual machines for minimizing the amount of online racks. After the job is pre-assigned, all booked virtual machines will not execute other jobs until the current job is finished. This centralized allocation strategy has two major advantages. On the one hand, many Map and Reduce tasks are mapped to virtual machines nearby in the same rack, then the immediate data produced by Map tasks are not or less transmitted, thus the data locality is improved, and transmission cost is obviously decreased. On the other hand, based on this centralized map strategy, all virtual machines of a same rack have higher probability to be idle at the same time. That is very important to save energy.

In summary, our rack-level multi-job pre-mapping algorithm contains three steps, and the corresponding pseudocode is presented in Algorithm 1.

**Step1**, all idle virtual machines are scanned to build the rack group, and the rack group is sorted by the number of idle virtual machines.

**Step2**, multiple queued jobs are merged into job group based on the number of idle virtual machines. Then, according to job request, the execution order of all jobs in one job group is adjusted.

**Step3**, according to new job execution order, all jobs of one group are centrally mapped to virtual machines of the same racks for minimizing the rack number. Based on the rack capacity evaluated by our measure algorithm [24], the resource waste and resource balance of the current rack are computed for the current job according to Eqs. (2) and (4). When the resource waste is more than zero, the virtual machines with smallest waste value and smallest resource balance are booked for the current job. When the resource waste is equal to or less than zero and the resource balance is less than other racks, then all virtual machines of this rack are booked for the current job.

### 4.3 Parallel task execution

After the multi-job pre-mapping, each job has booked virtual machines of multiple racks to wait for execution. All virtual machines of booked racks do not execute any task of other jobs until all tasks of the current job are completed. In addition, if any virtual machine of booked racks is idle, the tasks of corresponding job would be assigned to this virtual machine. Therefore, multiple jobs which book the resource of racks are parallel executed in virtual machines of booked racks.

In order to further enhance the data locality, a parallel task execution method is used to reduce the data communication from the immediate data of map tasks to reduce tasks in the job execution process. The basic idea of parallel task execution is that multiple map tasks and one reduce task are merged into task group and the same label is pasted. Each virtual machine would calculate multiple tasks with the same label in the whole job execution process. All virtual machines of one rack would compute adjacent task groups which have neighboring label. In the whole job execution process, all map tasks are firstly executed and the reduce tasks are executed until all map tasks with the same label are completed. Due to the heterogamous feature of virtual racks, computing speed of different racks is different. When a high-performance virtual machine has calculated one task group, if any new task group is not executed, then this virtual machine computes the new task group. If there is no

new task group, this high-capacity virtual machine starts to assist other virtual machines to execute some map tasks. This parallel task execution strategy based on task grouping has two advantages. Firstly, by task grouping, multiple map tasks and corresponding reduce task are executed in same virtual machines to decrease the data communication from immediate data of map tasks to reduce tasks. Secondly, because all virtual machines of one rack would calculate adjacent task group, immediate data communication is further decreased by public storage mapping.

---

#### Algorithm 2: *parallel\_task\_execution(job<sub>i</sub>, RL)*

---

**Input:** (i) job request  $job_i$ ,  
(ii) booked rack list  $RL$  of  $job_i$ .  
**Output:** the final job execution results.

- 1 based on the capacity of rack in  $RL$ , all map tasks and reduce tasks are split into multiple task groups  $TGS$ , and all tasks of one task group  $TG$  paste a same label.
- 2 **while** ( $job_i$  is not completed) **do**
- 3   each rack  $j$  in  $RL$  executes tasks in parallel.
- 4   **for** (each virtual machine  $vm_k$  in rack  $j$ ) **do**
- 5     //map task execution
- 6     **if** (any new  $TG$  in  $TGS$  is not executed) **then**
- 7       calculates all map tasks in one task group  $TG$
- 8       independently .
- 9     **else**
- 10      help other virtual machine to execute map tasks until
- 11      all map tasks of  $job_i$  are completed.
- 12     //reduce task execution
- 13     **if** (all map tasks of  $TG$  are executed on  $vm_k$ ) **then**
- 14       the immediate data of map tasks which are executed
- 15       on other virtual machines are transferred to  $vm_k$  and
- 16       the corresponding reduce task with same label is
- 17       executed on  $vm_k$ .
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- 48
- 49
- 50
- 51
- 52
- 53
- 54
- 55
- 56
- 57
- 58
- 59
- 60
- 61
- 62
- 63
- 64
- 65
- 66
- 67
- 68
- 69
- 70
- 71
- 72
- 73
- 74
- 75
- 76
- 77
- 78
- 79
- 80
- 81
- 82
- 83
- 84
- 85
- 86
- 87
- 88
- 89
- 90
- 91
- 92
- 93
- 94
- 95
- 96
- 97
- 98
- 99
- 100

13 return the final job results.;

---

In summary, the whole process of parallel task execution for one job consists of three steps, and the corresponding pseudocode is shown in Algorithm 2. In addition, because multiple jobs book virtual machines of different racks, the different jobs are parallel executed.

**Step1**, based on the smallest capacity of the rack in the booked racks for this job, all map tasks and reduce tasks are split into multiple task groups, and all tasks of each task group paste the same label.

**Step2**, after task grouping, all map tasks are firstly parallel executed on all virtual machines of booked racks. Each virtual machine independently calculates all map tasks in one task group at the beginning. When a task group is completed and other task group is not executed, then this virtual machine starts to calculate the next task group. If there is not any new task group, and job is not completed, this virtual machine helps other virtual machines to exe-

**Table 1** Illustration of four algorithms

Algorithm	Description
MAR++	Scheduling and data placement on MapReduce for heterogeneous environments [18]
SARS	A Self-adaptive Scheduling Algorithm for Reduce Start Time [11]
EMRSA	Energy-aware Scheduling of MapReduce Jobs for Big Data Applications [4]
TPJS	Energy- and Locality-Efficient Multi-job Scheduling based on MapReduce for Heterogeneous Datacenter

cute map tasks until all map tasks in all task groups are completed.

**Step3**, after all map tasks of one task group are completed, the corresponding reduce task starts to execute. If all map tasks are completed in the same virtual machine, the reduce task is executed on this virtual machine. If all map tasks are executed in multiple virtual machines, then the corresponding reduce task is executed in one virtual machine which calculates most map tasks in this task group.

#### 4.4 Time complexity

Based on the above description, our proposed algorithm contains mainly two phases, the job pre-assignment and parallel task execution. For job pre-assignment phase, multiple jobs are firstly merged into job group and the job execution order is adjusted according to the job request. In addition, the idea capacity of one rack for different jobs is pre-judged in the pre-assignment process. Therefore, the time complexity of job pre-assignment phases is that  $O(|J| \cdot |R|)$  where the term  $|J|$  is the number of multiple jobs and  $|R|$  is the number of idle racks. For parallel task execution phase, all racks parallel execute tasks of multiple jobs. So, the time complexity of parallel task execution is  $O(|V| \cdot |TG| \cdot |TS|)$ , where  $|V|$  is the maximum number of virtual machines in booking racks, the term  $|TG|$  is the task number of each task group and the  $|TS|$  is the average number of task groups which are calculated by each virtual machine. Therefore, the time complexity of our proposed algorithm is  $O(|J| \cdot |R| + |V| \cdot |TG| \cdot |TS|)$ .

**Table 2** Physical hosts

Type	Num	CPU	MEM (g)	DISK (g)	Power (W)
G1:Dell 3010	40	4 CPUs, 3.3 GHz	8 G	300 G	100 W
G2:HP DL320	20	8 CPUs, 2.6 GHz	8 G	300 G	160 W
G3:HP DL160	10	8 CPUs, 2.7 GHz	16 G	300 G	200 W
G4:Dell R720	10	16 CPU, 2.0 GHz	48 G	300 G	250 W

## 5 Experimental evaluation

In this paper, we build a heterogeneous Hadoop cluster to simulate the heterogeneous datacenter. And, we select three typical algorithms (MAR ++, SARS, and EMRSA) to compare with our proposed algorithm (TPJS). Table 1 shows the detailed information of four algorithms.

### 5.1 Experimental environment

The heterogeneous Hadoop cluster consists of four types heterogeneous physical servers, contains 80 physical hosts to form 16 racks. All physical hosts are set up with JDK 1.6, Hadoop1.22 and CentOS6.6. Table 2 shows the parameters of physical servers, including the number of servers, storage, CPU, MEM and power. Table 3 shows the configuration information of the 16 racks, in which eight racks consist of 40 Dell 3010 servers, four racks are made up of 20 HP DL320 hosts, two racks are constituted by ten HP DL160 hosts, and two racks consist of ten Dell R720 servers.

### 5.2 Experimental results

We compare the performance of TPJS, MAR++, SARS, and EMRSA algorithms from job scheduling time, resource balance rate, rack-to-rack traffic, amount of rack used, and energy usage. For each benchmark, different jobs are repeatedly executed many times to get the average value, where the processing file size (Input Size) and the number of tasks in the different jobs are shown in Table 4.

#### (1) Job scheduling time

Figure 4 shows the scheduling time of four algorithms on the TeraSort instance. From Fig. 4, we can find that the job scheduling time is seriously affected by the amount of tasks. In four algorithms, TPJS algorithm has minimized job scheduling time, next is SARS and EMRSA is the worst. More specifically, when the map tasks are small (0–80), job scheduling time of four algorithms is very similar. But, as the map tasks grow (80–960), the time of four algorithms grows linearly, in which EMRSA and MAR++ increase fastest, and represents they are worst. Figure 5 shows the job execution time of four algorithms on the PageRank instance. Similar



**Table 3** Rack configuration

Rack	G1	G2	G3	G4
R1–R8	40/5	0	0	0
R9–R12	0	20/5	0	0
R13–R14	0	0	10/5	0
R15–R16	0	0	0	10/5

to Fig. 4, EMRSA and MAR++ have higher job scheduling time than SARS and TPJS in all cases. Furthermore, the job execution speed of SARS is stronger than our TPJS when the amount of map tasks is less than 320; when the amount of map tasks is over 320, job scheduling time of our TPJS is shorter than SARS, and the gap between two algorithms becomes bigger and bigger.

**(2) Resource balance rate**

Resource balance rate indicates the percentage between the CPU utilization and memory utilization. The smaller the resource balance rate is, the less the resource waste is. Figure 6 shows the resource balance rate of four algorithms on the instance of TeraSort. As we can see Fig. 6, the resource balance rate of four algorithms changes frequently in different jobs. (The number of map tasks is from 0 to 960). Compared with other three algorithms, resource balance rate of our TPJS algorithm is relatively stable, and the average balance rate is only about 0.2. When the amount of map tasks increases from 320 to 640, the floating range of SARS and MAR++ algorithms is largest, so resource balance performance of these two algorithms is worst; EMRSA algorithm follows and our TPJS algorithm is best. Figure 7 shows the resource balance rate of four algorithms on the instance of PageRank. Based on the figure, TPJS has the smoothest trend line than the other three algorithms in different jobs where the amount of map tasks increases from 0 to 960. In particular, from 240 to 760, the resource balance rate of SARS, EMRSA and MAR++ sharply floats.

**(3) Rack-to-rack traffic**

The rack-to-rack traffic is the percentage between communication sizes of multiple racks and all communication sizes of whole job execution. The bigger the rack-to-rack traffic is, the worse the data locality is. Figure 8 shows the cross-rack

traffic of four algorithms on TeraSort instance. From Fig. 8, we can find that the rack-to-rack traffic of four algorithms changes frequently. But, Fig. 8 implies a unified trend that is the cross-rack traffic of four algorithms is growing with the growth of the amount of tasks (from 80 to 960). In the four algorithms, TPJS has the smallest rack-to-rack traffic and tiniest change. Figure 9 shows the performance of four algorithms on cross-rack traffic in the PageRank instance. The similar phenomenon is found, and the rack-to-rack traffic changes rapidly with the increase in Map tasks (from 0 to 960).

**(4) Amount of rack used**

The amount of rack used is an index to measure the concentration of job mapping. The smaller the used rack is, the higher the concentration of job is. Meanwhile, the rack used is small and the rack-to-rack traffic is also small. Figures 10 and 11 show the performance of four algorithms on the amount of rack used in two different instances, TeraSort and PageRank. In the TeraSort instance (Fig. 10), because the trend line of TPJS is closer to the bottom, TPJS algorithm is significantly better than other algorithms. More specifically, as the number of map tasks (0–960) grows, the used rack grows correspondingly. The SARS and MAR++ use most racks, EMRSA follows, and TPJS is minimized. In the PageRank instance (Fig. 11), when the map tasks are more than 80, EMRSA and TPJS have smaller used racks than SARS and MAR++; when the map tasks are over 320, the TPJS uses smaller racks than EMRSA.

**(5) Host energy usage**

To compare the energy consumption, the energy usage of physical hosts with different types is considered to observe the energy consumption of four algorithms. Figures 12 and 13 show heterogeneous hosts usage of four algorithms. Through multiple repeated experiments on TeraSort and PageRank, the different performances of four algorithms on occupying physical hosts are clearly shown. Based on the two figures, in all cases (the number of map tasks is from 0 to 960), TPJS uses more low-energy (low-profile) physical hosts than other three algorithms. That is to say, the used number of physical hosts Host-G1 and Host-G2 in our TPJS algorithm is bigger than that in other algorithms. In addition, the usage of four physical hosts (Host-G1, Host-G2, Host-G3, and Host-G4) in

**Table 4** Job parameters

Job type	Input size (GB)	Map tasks
TeraSort	5/9.4/15/20/26.8 /31.8/47.5/52.5/60	80/150/240/320/430 /510/640/760/840/960
PageRank	5/9.4/15/20/26.8 /31.8/47.5/52.5/60	80/150/240/320/430 /510/640/760/840/960

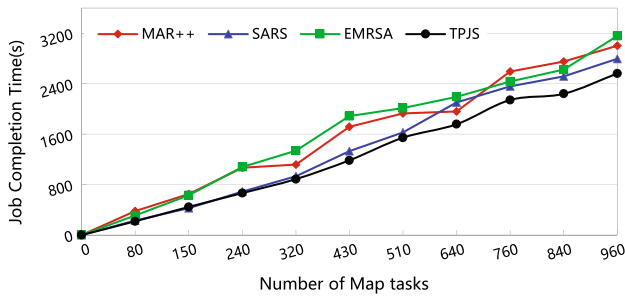


Fig. 4 Job scheduling time of TeraSort

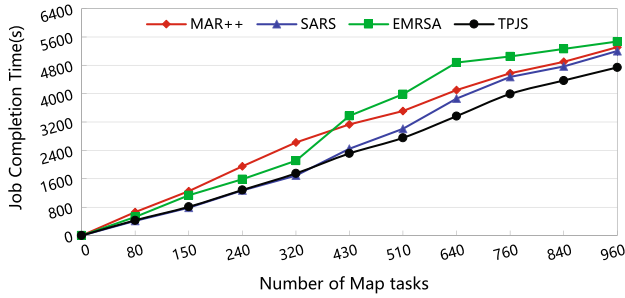


Fig. 5 Job scheduling time of PageRank

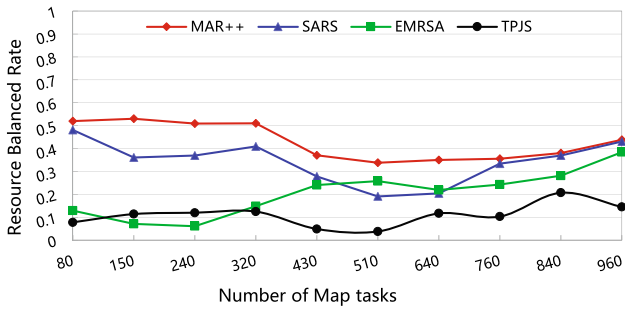


Fig. 6 Resource balance rate of TeraSort

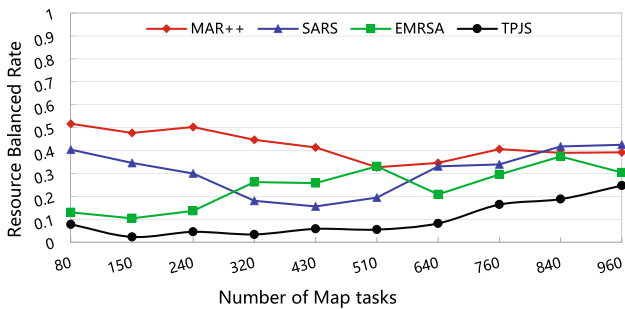


Fig. 7 Resource balance rate of PageRank

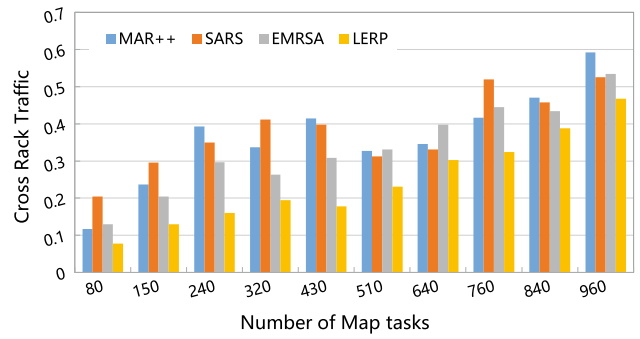


Fig. 8 Rack-to-rack traffic of TeraSort

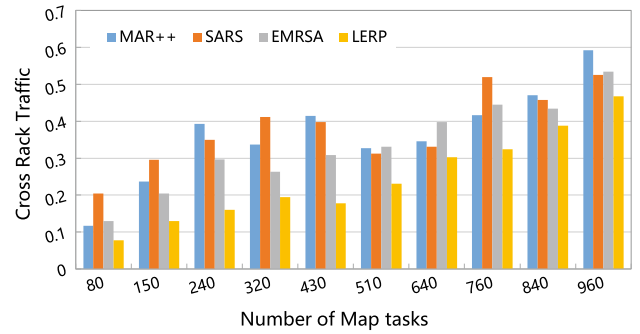


Fig. 9 Rack-to-rack traffic of PageRank

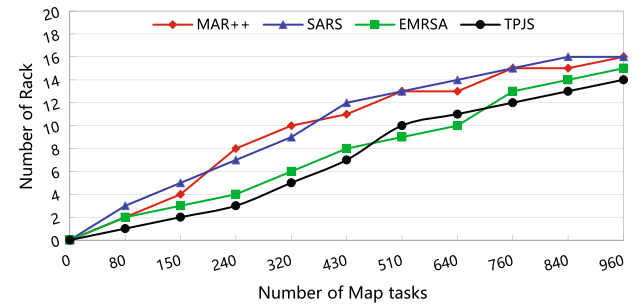


Fig. 10 Amount of rack used in TeraSort

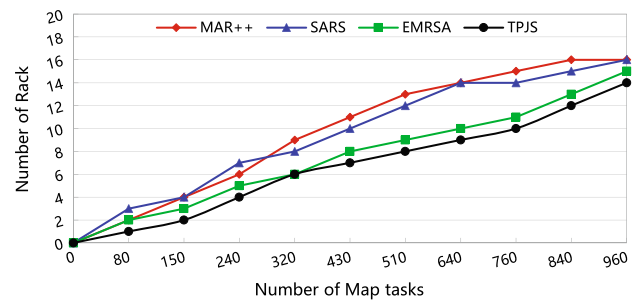
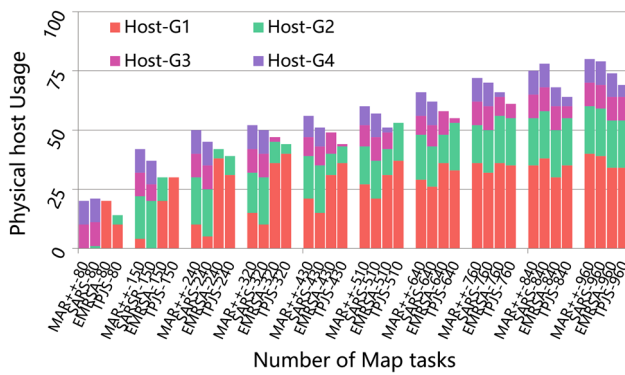


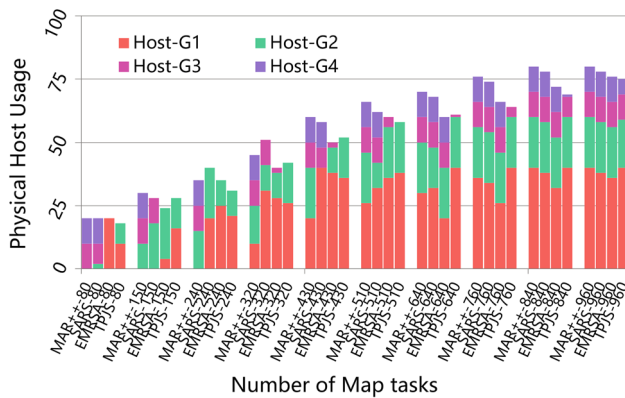
Fig. 11 Amount of rack used in PageRank

EMRSA is more balanced. And MAR++ and SARS prefer to use high-energy (high-profile) physical hosts (Host-G3 and Host-G4). Based on Figs. 12 and 13, we can conclude that TPJS algorithm can save energy usage in the job scheduling process by the rack-level job-centralized mapping.

In summary, based on the above experiments, we can find our TPJS algorithm has better performance with other algorithms from five aspects of job scheduling time, resource balance rate, rack-to-rack traffic, amount of rack used and host energy usage. The main reason has two points. On the



**Fig. 12** Physical host usage of TeraSort



**Fig. 13** Physical host usage of PageRank

one hand, TPJS takes the rack as the basic unit of resource allocation and tries her best to assign all tasks to as few racks as possible. In this way, the number of physical hosts online can be reduced, energy-saving can be achieved by a large margin, and management can be facilitated at the same time. On the other hand, through this centralized mapping strategy, map and reduce tasks of one job can be nearby and data transmission between them will be reduced significantly, while job execution time will be speeded up.

## 6 Conclusion

To jointly optimize the job scheduling time, data transmission, job cost, and energy-saving, an energy- and locality-efficient multi-job scheduling algorithm is developed to improve the performance of MapReduce tasks on heterogeneous datacenter. The main works of our algorithm are as follows: (1) Based on the rack capacity, a multi-job pre-mapping method is designed to enhance the resource utilization and avoid the resource waste. By this way, a job will be centrally allocated to multiple virtual machines of the same rack to minimize the number of online racks, save energy, and reduce the data traffic of the job. (2) After that, all pre-

assigned tasks will be executed in parallel to further improve the data locality and decrease the data communication of the immediate data between map and reduce tasks. Compared with other three algorithms, lots of experimental results prove the advantages of our TPJS algorithm from the five aspects of job scheduling time, resource balance rate, rack-to-rack traffic, amount of rack used and host energy usage.

In the future, we will try to test the execution process of different phases in MapReduce scheduling and further optimize the job scheduling from the idea “overlapping execution in different phases.”

**Acknowledgements** This work was supported by the Science Research Project of Education Department of Hunan Province (18C0296); the Open Project of State Key Laboratory of Advanced Design and Manufacturing for Vehicle Body (31715010); Hunan Provincial Natural Science Foundation of China (2018JJ2134); Hunan Provincial Young Talents Project (2018RS3095); and Ph.D. research startup foundation of Hunan University of Science and Technology (E51863).

## References

1. Hashem IAT, Anuar NB, Marjani M et al (2018) MapReduce scheduling algorithms: a review. *J Supercomput* 18(1):1–31
2. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. *Commun ACM* 51(1):107–113
3. Dahiphale D, Karve R, Vasilakos AV et al (2014) An advanced mapreduce: cloud mapreduce, enhancements and applications. *IEEE Trans Netw Serv Manag* 11(1):101–115
4. Mashayekhy L, Nejad MM, Grosu D et al (2015) Energy-aware scheduling of mapreduce jobs for big data applications. *IEEE Trans Parallel Distrib Syst* 26(10):2720–2733
5. Bampis E, Chau V, Letsios D, Lucarelli G, Milis I, Zois G (2014) Energy efficient scheduling of mapreduce jobs. In: *Euro-Par 2014 parallel processing*. Springer
6. Wang J, Li X, Yang J (2015) Energy-aware task scheduling of mapreduce cluster. In: *2015 international conference on service science (ICSS)*
7. Maheshwari N, Nanduri R, Varma V (2012) Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework. *Future Gener Comput Syst* 28(1):119–127
8. Chen Y, Alspaugh S, Borthakur D, et al (2012) Energy efficiency for large-scale mapreduce workloads with significant interactive analysis. In: *Proceedings of the 7th ACM European conference on computer systems*
9. Palanisamy B, Singh A, Liu L, Jain B (2011) Purlieus: locality-aware resource allocation for mapreduce in a cloud. In: *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*
10. Chen L, Zhang J, Cai L et al (2017) Fast community detection based on distance dynamics. *Tsinghua Sci Technol* 22(6):564–585
11. Tang Z, Jiang L, Zhou J, Li K, Li K (2015) A self-adaptive scheduling algorithm for reduce start time. *Future Gener Comput Syst* 43:51–60
12. Ramanathan R, Latha B (2018) Towards optimal resource provisioning for Hadoop-MapReduce jobs using scale-out strategy and its performance analysis in private cloud environment. *Clust Comput* 2:1–11
13. Lin JW, Arul JM, Lin CY (2018) Joint deadline-constrained and influence-aware design for allocating MapReduce jobs in cloud computing systems. *Clust Comput* 1:1–14

14. Zhu Y, Jiang Y, Wu W, Ding L, Teredesai A, Li D, Lee W (2014) Minimizing makespan and total completion time in mapreduce-like systems. In: 2014 proceedings on INFOCOM. IEEE
15. Palanisamy B, Singh A, Liu L (2015) Cost-effective resource provisioning for mapreduce in a cloud. *IEEE Trans Parallel Distrib Syst* 26(5):1265–1279
16. Lin M, Zhang L, Wierman A, Tan J (2013) Joint optimization of overlapping phases in mapreduce. *Perform Eval* 70(10):720–735
17. Heintz B, Chandra A, Weissman J (2014) Cross-phase optimization in mapreduce. In: *Cloud computing for data-intensive applications*
18. Anjos JC, Carrera I, Kolberg W, Tibola AL, Arantes LB, Geyer CR (2015) Mar++: scheduling and data placement on mapreduce for heterogeneous environments. *Future Gener Comput Syst* 42:22–35
19. Jin H, Yang X, Sun X-H, Raicu I (2012) Adapt: availability-aware mapreduce data placement for non-dedicated distributed computing. In: 2012 IEEE 32nd international conference on distributed computing systems (ICDCS). IEEE
20. Xie J, Yin S, Ruan X, Ding Z, Tian Y, Majors J, Manzanares A, Qin X (2010) Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In: 2010 IEEE international symposium on parallel and distributed processing, workshops and Ph.D. forum (IPDPSW). IEEE
21. Al-Khasawneh MA, Shamsuddin SM, Hasan S et al (2018) MapReduce a comprehensive review. In: 2018 international conference on smart computing and electronic enterprise (ICSCEE) on IEEE
22. Gregory A, Majumdar S (2018) Resource management for deadline constrained MapReduce jobs for minimising energy consumption. *Int J Big Data Intell* 5(4):270–287
23. Elzein NM, Majid MA, Hashem IAT et al (2018) Managing big RDF data in clouds: challenges, opportunities, and solutions. *Sustain Cities Soc* 39:375–386
24. Chen L, Zhang J, Cai L et al (2016) Locality-aware and energy-aware job pre-assignment for mapreduce. In: *International conference on intelligent networking and collaborative systems*

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.