Journal of Computers Vol. 28, No. 2, 2017, pp. 145-164
doi:10.3966/199115592017042804011

A Cross-Jobs-Cross-Phases Map-Reduce Scheduling Algorithm in Heterogeneous Cloud

Lei Chen[1], Jing Zhang[1], Li-Jun Cai[2], Zi-Yun Deng[1], and Tao Meng[2]

[1] College of Electrical and Information Engineering, Hunan University, Changsha 410082, China

[2] College of Information Science and Engineering, Hunan University, Changsha 410082, China
{chenleixyz123, hetingqin, ljcai, dengziyun, mengtao}@hnu.edu.cn

Received 30 December 2015; Revised 03 June 2016; Accepted 03 July 2016

Abstract. To fast process the large-scale data, map-reduce cloud is viewed as a very reasonable and effective platform. According to the new scheduling challenges in map-reduce cloud, a cross-jobs-cross-phases (CJCP) map-reduce scheduling algorithm is proposed in this paper. CJCP mainly consists of four optimal schemes, and respectively deals with four resource waste scenes of the job scheduling process. In the first scene, based on the job training method, an optimal scheme is designed to shield the interference of heterogeneous resources on job scheduling. In the second scene, we give two definition and develop another optimal scheme to dynamically adjust task number of multiple virtual machines on the same physical host. Through task adjustment, the high-capacity virtual machines deal more tasks than low-capacity ones. In addition, we build the overlapping execution model to overlap map, shuffle and reduce phases. In the third scene, considering the difference of map tasks and push tasks on resource usage needs, an overlapping scheme is formed to optimal the execution of push and map phase. In the last scene, a cross-jobs optimal scheme is proposed, which overlap execution of current job and next job. To avoid the bandwidth confliction between two jobs, a monitor method is used for reasonably resources allocation. Extensive experiments show that our algorithm consumes less job execution time and performs better than other three algorithms.

Keywords: cloud computing, heterogeneous scheduling, map-reduce, overlapping distribution

1 Introduction

Big data era is coming and accompany with some important data characteristics: rapidly increasing, diversity and large-scale. It brings new opportunities and challenges for all walks of life. How to efficiently process and mine the big data is becoming the focus both in industry and academic research. map-reduce [1] is a large-scale distributed data programming model, which was proposed by Google in 2004, it is one of the most popular framework to big data. Currently, many companies and research institutes have set up their own map-reduce clusters, which require a lot of hardware resources (servers and storage) and professional maintenance staff. Otherwise, due to resource independence and decentralization position of the clusters, causing great waste of underlying physical resources. Therefore, to build map-reduce cloud has become a powerful tool for large-scale data processing.

Based on the cloud computing model, map-reduce cloud platform provide a shared cloud services to users for large-scale data processing in "pay on-demand" approach. User can use map-reduce service at anywhere and anytime by submitting online jobs. In map-reduce cloud, the underlying physical resources consist of one or more geographically dispersed heterogeneous data center, the physical hosts configure with different types of CPU, MEM, storages.

The total map-reduce scheduling process includes four phase: data transfer (push phase), map phase, shuffle and reduce phase, as shown in Fig. 1. Before the jobs were submit, user should specify four pa-

rameters, including input data, map function, reduce function the reduce task size. Push is the first phase, after the job is submit, and responsible for dividing input data into a fixed-size data blocks (trunk), default size is 64M. Map is the second phase, according to the map function, it computes the data blocks and generates intermediate key/value results. The association between push task and map task is 1:1, that is to say, one map task handle one data block. Shuffle is the third phase, after the map tasks have been completed, the intermediate key/value results are sorted and sent to corresponding reduce nodes. Otherwise, on one reduce node, it merge multiple map tasks results according to the keys, and duplicate redundancy data. Reduce is the last phase, it processes the map results and generates final results. The association between map task and reduce task is n: 1, one reduce task handle multiple map task results.

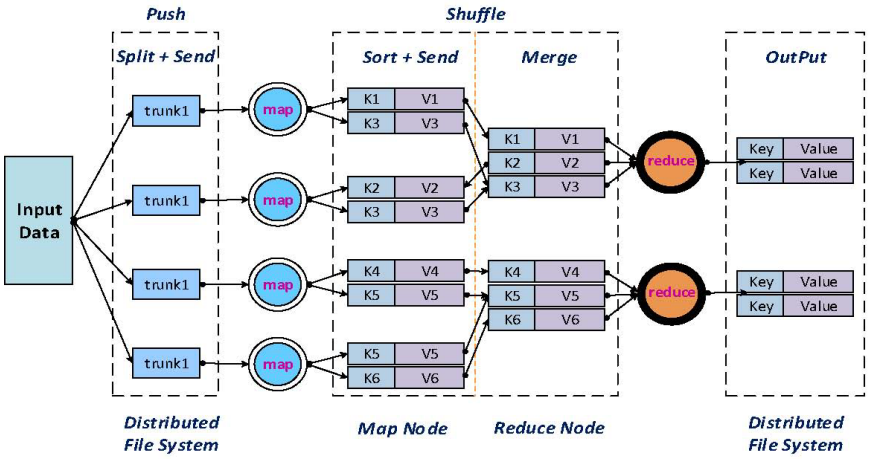


Fig. 1. Scheduling process of map-reduce job

Comparing to the traditional map-reduce scheduling, the process of map-reduce cloud appears new characteristics:

- Massive jobs and users. For the traditional map-reduce cluster, just a few users would submit jobs, there are no need to concern the preemption or insufficient resources on scheduling. However, in map-reduce cloud, because of massive users may random submit multiple jobs at any time, a job may suffer the long time wait or even jobs execution failure from resources insufficient. Therefore, in map-reduce cloud, the scheduling process is more complex.
- Heterogeneous resource nature. In the traditional map-reduce cluster, the resources are usually homogeneous. All compute nodes configure same type of CPU, MEM, DISK, and so on. However, in map-reduce cloud, the clusters usually consists of multiple different types underlying hardware resources. Each type hardware resources have different CPU, MEM, and DISK. Therefore, the map-reduce scheduling on cloud is more difficult. In addition, map-reduce cloud usually hire multiple different geographical data center.
- Virtualization. In traditional map-reduce scheduling, each physical server is a compute node. However, in the map-reduce cloud, virtualization technology has been developed to enhance the utilization of underlying hardware resource. One server will virtual into multiple virtual machines and each virtual machine will be used as a single compute node. The scheduling and management of virtual machines is also a new challenge.

Currently, a lot of research papers about map-reduce scheduling have been published. Based on the research direction, most works can be classified into three aspects, single phase optimization, cross-phases optimization, and parameters adjustment optimization. (1) On single phase optimization aspect, based on the memory perfecting, a high performance scheduling algorithm is proposed for improving the data locality [2]. Focusing on the map phase, a queuing scheduling algorithm is designed to allocate the task based on maximum weight [3]. With an eye to shuffle phase, based on the bandwidth constraints and data locality, two dynamic shuffle scheduling algorithms are developed to reduce the communication overhead in [5-6]. Concerning on the reduce phase, through minimize online makespan time, an optimal scheduling algorithm is designed [7]. Similar, based on the prediction strategy, Tang, Jiang, Zhou, Li and Li put forward an overlapping optimal algorithm to reduce the start execution time [8]. In a cloud envi-

ronment, Zhao, Wei, Zhang and He proposed a two-phase weighted task scheduling algorithm [14]. (2) On cross-phases optimization aspect, an overlay mode is proposed for push and map phases [9]. Using RDMA technology in the high performance environment, an optimal algorithm by overlapping map and shuffle phases is designed [10]. Otherwise, for cloud scheduling, a cross map and reduce scheduling algorithm is proposed [12]. Based on the combination of overlapping in push and map, shuffle and reduce, Heintz, Chandra and Weissman proposed a tow optimal scheme [11]. (3) On parameters adjustment optimization aspect, from job parameter and application view, block device reconfiguration algorithm and MRONLINE algorithm are developed to optimal map-reduce job execution [15-17]. Otherwise, in the cloud environment, Palanisamy, Singh and Liu designed a new map-reduce cloud service model, Cura, it dynamically configures the best hadoop cluster according to the user-submitted needs, so as to reduce the job execution time [13]. In the above algorithms, multiple jobs scheduling optimization is less considered. Otherwise, in cloud environment, heterogeneous resources are rarely researched. So to simultaneously optimal cross phases, cross jobs and heterogeneous resources in cloud is a huge challenge.

The motivation of this paper is to find the scheduling features of multiple phases, multiple jobs and heterogeneous resources so as to develop some optimal schemes to speed up the job execution process, reduce the total running time. According to the new challenges of map-reduce scheduling in cloud, four resource waste scenes are existed among multiple phases (push, map, shuffle, and reduce), multiple jobs and heterogeneous resources, so, respectively four optimal schemes are formed to solve the four resource waste problem. Combining with the four schemes, a cross-jobs-cross-phases map-reduce scheduling algorithm is proposed, which is simply called as CJCP. CJCP takes three overlapping method in three optimal scheme, which saves job execution time and consumption cost. In addition, we develop job training method to shield the heterogeneous feature of underlying physical hosts. The detailed description of CJCP algorithm is showed in section 3.

Other section are organized as follows: The map-reduce scheduling process is formulated and analyzed in section 2. Section 3 presents our algorithm in detail. Section 4 contains an experimental evaluation. We finally conclude the paper in section 5.

2 Problem Definition and Analysis

The main objective of this study is to minimize the execution cost and time of the map-reduce job in cloud environment. In this section, the map-reduce task scheduling process is firstly formulated under heterogeneous cloud environment. Then, the four resource waste scenes existing in the scheduling processing will be analyzed in detail.

2.1 Problem Definition

The scheduling process of map-reduce jobs in heterogeneous cloud simplify as: users usually online submit multiple map-reduce jobs to the cloud. Each job job_j contains three type of tasks, including an input data queue $Q_{job_j}^{data}$, a map task queue $Q_{job_j}^{map}$, a reduce task queue $Q_{job_j}^{reduce}$. The input data queue contains $C_{job_j}^{data}$ same size of data trunk, the default size is 64M. The input data queue will be stored at public storage server after uploading by users, when the tasks processing begin, the data would be transmitted to data nodes. The map queue $Q_{job_j}^{map}$ contains $C_{job_j}^{map}$ map tasks. A map task $task_{map}$ processes one single data trunk $task_{data}$, so, the number of map queue $C_{job_j}^{map}$ is same as $C_{job_j}^{data}$. The reduce queue $Q_{job_j}^{reduce}$ includes $C_{job_j}^{reduce}$ reduce tasks. One reduce task deals with the results of multiple map tasks.

Once a job has been submitted to the cloud, the task scheduler starts to allocate the three type tasks to underlying compute resources (virtual machine: VM). The underlying physical resources are heterogeneous in the cloud, it consists of different type physical hosts and VMs. In the cloud, virtual machine is a basic unit for task computing, so, the computing resources mentioned in the following are referred to virtual machines. More specifically, each virtual machine configures with four different resources (CPU, MEM, DISK and BW). For each type of VM, the rental cost $Cost^{rent}$ of unit time and the communication cost $Cost^{com}$ of unit data are different. In addition, one virtual machine may continuously executes multi-

ple map or reduce tasks in one task scheduling.

To better formulate the job execution time and costs, we suppose one job job_j map to $C_{job_j}^{vm}$ virtual machines, each virtual machine vm_i deals with C_{vm_i,job_j}^{data} input data trunks (push task), C_{vm_i,job_j}^{map} map tasks, C_{vm_i,job_j}^{reduce} reduce tasks and migrating map results from the other virtual machines. In map-reduce process, one push task transmits one data trunk from the public storage to vm_i and the time-consuming is t_{vm_i,job_j}^{data}; one map task process a data block of job_j on virtual machine vm_i and the time-consuming is t_{vm_i,job_j}^{map};one reduce task handles results of $C_{job_j}^{pair}$ map tasks, for one map task, the time-consuming is t_{vm_i,job_j}^{reduce}. Furthermore, the consuming time on migrating all map results from the other virtual machines is $t_{vm_i,job_j}^{transfer}$

In summary, the total task execution time t_{vm_i,job_j} of job_j on a virtual machine vm_i is formulated as:

$$\begin{aligned} t_{vm_i,job_j} = & \sum C_{vm_i,job_j}^{data} * t_{vm_i,job_j}^{data} + \sum C_{vm_i,job_j}^{map} * t_{vm_i,job_j}^{map} \\ & + t_{vm_i,job_j}^{transfer} + \sum C_{vm_i,job_j}^{reduce} * t_{vm_i,job_j}^{reduce} \end{aligned} \tag{1}$$

In Equation (1), task execution time on one virtual machine consists of four parts, including push data transmission time, map task execution time, map result shuffle time and reduce task running time.

Based on this, the eventually execution time of a map-reduce job is as follows:

$$t_{job_j} = \max\left\{ t_{vm_i,job_j} \middle| 0 \le i \le C_{job_j}^{vm} \right\} \tag{2}$$

In Equation (2), the eventually execution time of a job is the maximum running time of all virtual machines which is mapped to multiple tasks of job_j.

Similarly, the running cost of job_j on virtual machine vm_i is as follows:

$$\begin{aligned} cost_{vm_i,job_j} = & t_{vm_i,job_j} * cost_{vm_i}^{rent} + C_{vm_i,job_j}^{data} * \left| task_{vm_i}^{data} \right| * cost_{vm_i}^{com} \\ & + \left\| transfer_{vm_i}^{data} \right\| * cost_{vm_i}^{com} \end{aligned} \tag{3}$$

In the above formulation, the execution costs consist of three parts, including virtual machine rent costs, push data transmission costs, shuffle data migration costs. In addition, $|\bullet|$ indicates the single data trunk size of job_j and $\|\bullet\|$ indicates the total migrating data size of map results from other virtual machines.

So, the total execution cost of job_j can be defined as follows:

$$cost_{job_j} = \sum_{i=1}^{C_{job_j}^{vm}} cost_{vm_i,job_j} \tag{4}$$

Therefore, the objective function of map-reduce task scheduling model on heterogeneous cloud as follows:

$$\begin{aligned} & min\ t_{job_j}\ \ and\ cost_{job_j} \\ & st. \\ & \sum_{i=1}^{C_{job_j}^{vm}} C_{vm_i,job_j}^{data} = C_{job_j}^{data}\ ,\ \sum_{i=1}^{C_{job_j}^{vm}} C_{vm_i,job_j}^{map} = C_{job_j}^{map}\ ,\ \sum_{i=1}^{C_{job_j}^{vm}} C_{vm_i,job_j}^{reduce} = C_{job_j}^{reduce} \end{aligned} \tag{5}$$

2.2 Problem Analysis

In the above description of map-reduce scheduling process under heterogeneous cloud, four resource waste scenes are existing, including map-shuffle-reduce waste scene, push-map waste scene, heterogeneous resource waste scene and cross jobs waste scene.

Scene 1 (heterogeneous resource waste). In the traditional map-reduce scheduling, the resources are homogeneous, all node configure with the same bandwidth, CPU and MEM, the execution time of same type of task could be regard as same, and usually take evenly allocation strategy. However, in heterogeneous cloud environment, the computing nodes configure with different types of CPU and MEM, the time for dealing with same types of tasks are different. The evenly allocation strategy will inevitably lead to virtual machines resources waste, this scene is called "heterogeneous resource waste," as shown in Fig. 2.

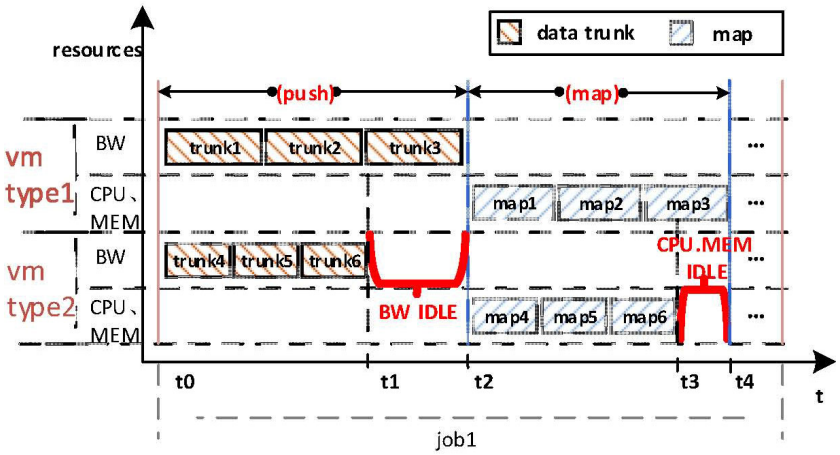


Fig. 2. Scene of heterogeneous resource waste

In Fig. 2, two types of virtual machine process *job1* using evenly distributed strategy, each virtual machine transfers three blocks and calculates three map tasks. At *t0*, two virtual machines simultaneously execute push phase, at *t2*, execute map phase, at *t4*, execute shuffle phase. However, the capabilities of two virtual machines are different, so *vmtype2* completes push phase at *t1*, at *t3*, completes map phase. Thus, *vmtype2* bandwidth resource is idle from *t1* to *t2*. CPU and MEM resources are idle from *t3* to *t4*.

Scene 2(map-shuffle-reduce waste). In general, map tasks and reduce task may not always on the same compute node and the results of multiple map tasks are processed by one reduce task, reduce task needs to wait until shuffle phase finishes, shuffle phase also needs to wait until map tasks implement on different virtual machines, this serial execution model causes resource waste. Such that reduce tasks on virtual machine has to wait when map tasks on other virtual machines are still running. This scene is **map-shuffle-reduce waste**. Let us take an example (see Fig. 3). In the figure, three map tasks of *job1* execute on *vm1*, complete at *t1* time point. From *t1* and *t2*, sort the results of three map on *vm1* and migrates data to *vm2*. At *t3* reduce task begins and completes at *t4*. In the whole process, reduce task on *vm2* must wait for the running of map tasks on *vm1*. So, from *t0* to *t1*, bandwidth (BW) resources on *vm2* are idle. Similarly, from *t0* to *t2*, CPU and MEM resources are also idle on *vm2*.

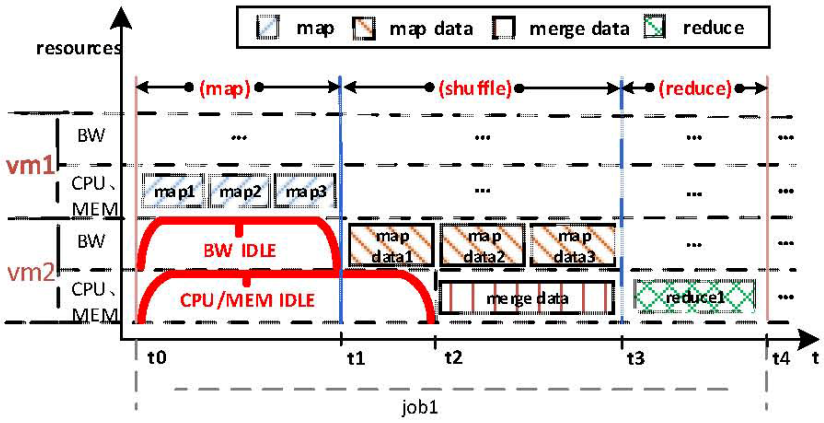


Fig. 3. Scene of map-shuffle-reduce waste

Scene 3 (push-map waste). Under the traditional scheduling process, push phase and map phase belongs

to sequentially connection. That is, map tasks must wait until push phase finishes, thus causing **push-map waste** problem, as shown in Fig. 4. In the figure, *job1* occupied *vm1* to execute push and map tasks. From *t1* to *t2* execute push phase. From *t2* to *t3* execute map phase. However, because of the sequentially execution, from *t1* to *t2*, CPU and MEM resources has been in idle status and causing waste.

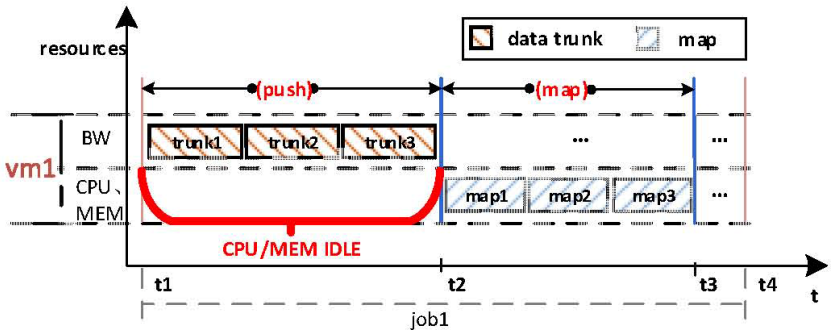


Fig. 4. Scene of push-map waste

Scene 4 (cross jobs waste). In general, one virtual machine is allocated for one job in one scheduling process. In two scheduling, one virtual machine carried out multiple tasks of two different jobs and the running process of two different jobs is a flow model. This flow model causes the resource waste of virtual machine in two different jobs, we calls this situation as **cross jobs waste** (see Fig. 5). As shown in the figure, virtual machines *vm2* and *vm1* process *job1*, *job2*, *job3* and *job4* using FIFO strategy. It can be seen from Fig. 5 (a), from *t0* to *t2*, *job1* occupy *vm1* in first scheduling. From *t2* to *t3*, *job2* occupy *vm1* in second scheduling. At *t1*, push phase of *job1* completes, map tasks begin. At *t2*, map task of *job1* completes, push phase of *job2* begins. So, from *t1* to *t2*, BW resources of *vm1* are idle. Similarly, from Fig. 5 (b), *job3* and *job4* occupied *vm2* resource from *t4* to *t7*. At *t5*, reduce task *job3* begins. At *t6*, *job3* completes, *job4* start to transfer data and begin push phase. From *t5* to *t6*, BW resources on *vm2* are idle.

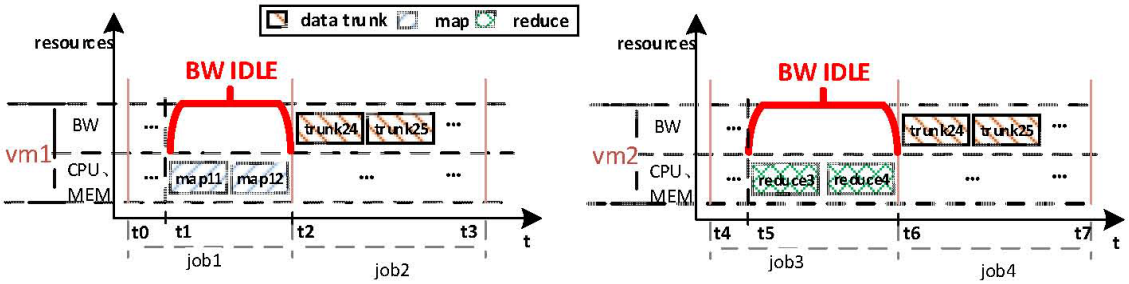


Fig. 5. Scene of cross jobs waste

In the Fig. 5, when two jobs execute on the same virtual machine in different scheduling process, no matter the last task is map (Fig 5. (a)) or reduce (Fig 5. (b)), it will inevitably lead to bandwidth resources waste. The main reason is that the different type resource are used by two different jobs in different scheduling process. Therefore, if the two jobs on same virtual machine do not occupy the same resources in different scheduling process, then the two jobs can be overlapped.

3 Algorithm design

In this section, a cross-jobs-cross-phases map-reduce scheduling algorithm CJCP is proposed in heterogeneous cloud environment. Four optimal schemes are developed for four waste scenes in section 3.1. The CJCP algorithm is presented in section 3.2.

3.1 Optimal Schemes for Four Problems

Heterogeneous resource optimization for scene 1. In the scene 1, due to the heterogeneous feature of

underlying computing nodes, same type task (push, map and reduce) of one job on different virtual machines perform different running time, high ability node consume less times and low ability node needs more times, If take evenly task distributed strategy, evenly distribute same size map tasks to different type of virtual machines, which will cause resource idle of high ability node and increases the total job execution time..

To decrease the influence of heterogeneous feature, an optimal scheme is to reduce the job execution time. In this scheme, the high ability nodes undertake more tasks, and low ability nodes undertake less tasks, thus that heterogeneous computing node could achieve a similar task execution time on same job. That is to say, the optimal scheme must be able to measure the capability of all virtual machines in the heterogeneous cloud. Otherwise, the capability of one virtual machine is different on dealing with different type jobs or tasks.

To measure the capability of one virtual machine for different jobs or tasks, firstly is to build a baseline for different jobs. A training approach is used to construct the baseline for different jobs on different virtual machines. The base idea of training approach is to save the historical average execution time of three type of tasks (push, map and reduce) on different virtual machines, the historical average execution time is treated as a baseline. Then test the three type tasks execution time on fixed test virtual machine, the execution time of current virtual machine can be calculate according to the proportional relation between current virtual machine and fixed test virtual machine. The proportional relation means the time ratio between task execution time of current job and historical average running time on one job which is equal to the time ratio between test execution time of fixed test virtual machine and his historical average execution time on same job. The definition of proportional relation is as follows:

$$\begin{aligned} &\because \quad \frac{time^{type}_{vm_i,job_j}}{avg\_time^{type}_{vm_i}} = \frac{train\_time^{type}_{vm_{test},job_j}}{avg\_time^{type}_{vm_{test}}} \quad \left| type \in (push, map, reduce) \right. \\ &\therefore \quad time^{type}_{vm_i,job_j} = \frac{time^{type}_{vm_{test},job_j}}{avg\_time^{type}_{vm_{test}}} * avg\_time^{type}_{vm_i} \quad \left| type \in (push, map, reduce) \right. \end{aligned} \tag{6}$$

In the above equation, $time$ is the execution time of one type task of job_j on vm_i. $train\_time$ is the test running time of same type task of job_j on fixed test virtual machine. vm_{test}. avg_time is the historical average execution time on virtual machine. Otherwise, $time$ is objective variable and three others are constant.

In summary, the process of optimal scheme consists of three steps. Firstly, a training library is built to save the historical average execution time of different type task on different type virtual machines. When each job completes, the update method begins to improve the historical average value. Secondly, a fixed type virtual machine is selected as a test node. Once user submits job to cloud, three single tasks of three types (push, map and reduce) are trained on test node and the test execution time of job stored into training library. Thirdly, according to job information and current virtual machine type, the execution time of three type of tasks can be calculated through Equation (6) respectively. The pseudo code of optimal scheme is descripted in algorithm 1.

In algorithm 1, the heterogeneous nodes capacity is determined by the execution time of different types of tasks for each type node. The smaller the time is, indicating that the tasks process ability is stronger. Conversely, weaker. In order to be fairer to measure ability of different types of computing nodes, a large number of test jobs should be execute to avoid the impact of network or node error.

Algorithm 1: Heterogeneous resources optimization on virtual machine level

Input: job information job_j and virtual machine type $type_{vm_i}$.

Output: (i) the estimate execution time $time^{map}_{vm_i,job_j}$ of single map task slice on $type_{vm_i}$,

(ii) the estimate data transmission time $time^{push}_{vm_i,job_j}$ of single push task slice on $type_{vm_i}$,

(iii) the estimate execution time $time^{reduce}_{vm_i,job_j}$ of single reduce task slice on $type_{vm_i}$.

Procedure heterogeneous_resources_optimization (job_j , $type_{vm_i}$)

1: get fixed training virtual machine type $type_{vm_{test}}$.

2: read the train push time $train\_time^{push}_{vm_{test},job_j}$, train map execution time $train\_time^{map}_{vm_{test},job_j}$, train reduce execution time $train\_time^{reduce}_{vm_{test},job_j}$ of single task slice from the train library by $type_{vm_{test}}$ and job_j .

3: read the average data transmit time $avg\_time^{push}_{vm_{test}}$, average map execution time $avg\_time^{map}_{vm_{test}}$, average reduce execution time $avg\_time^{reduce}_{vm_{test}}$ from train library which is the historical average job execute results for $type_{vm_{test}}$.

4: read the average data transmit time $avg\_time^{push}_{vm_i}$, average map execution time $avg\_time^{map}_{vm_i}$, average reduce execution time $avg\_time^{reduce}_{vm_i}$ from the train library for $type_{vm_i}$.

//calculate three task slice estimate execute time

5: if ($train\_time^{push}_{vm_{test},job_j}$!=null $\cap$ $train\_time^{map}_{vm_{test},job_j}$!=null $\cap$ $train\_time^{reduce}_{k,j}$!=null)

6: calculate the estimate execution time $time^{push}_{vm_i,job_j}$ according to the Equation (6).

7: calculate the estimate data transmission time $time^{push}_{vm_i,job_j}$ according to the Equation (6).

8: calculate the estimate execution time $time^{reduce}_{vm_i,job_j}$ according to the Equation (6).

9: end if

10: return $time^{push}_{vm_i,job_j}$, $time^{push}_{vm_i,job_j}$, $time^{reduce}_{vm_i,job_j}$.

Map-shuffle-reduce optimization for scene 2. In scene 2, underlying physical resources are idle because of three phase tasks (map, shuffle and reduce) serial execution and three phase tasks wait for each other. In order to decrease the resource waste and speed up the job execution time, the key is to reduce the wait time. To reduce the wait time, shielding the sequential execuion way is a feasible method. In this paper, an overlapping optimal scheme is proposed for the map-shuffle-reduce waste scene. The goal of this scheme includes two aspects, one is to overlap three phases of one job on multiple virtual machines and another one is to dynamically adjust tasks size of multiple virtual machines on same physical host (as algorithm 1), such to achieve the unevenly tasks allocation and assure the similar execution time on all virtual machines.

To better descript the overlapping scheme, we introduce two new definition firstly.

Definition 1: task pairwise. In the scheduling process, one map task deals with one data trunk (one push task) and one reduce task deals with results of multiple map tasks. The ratio between reduce task and map tasks is the task pairwise. The task pairwise implies the work strength of current job. The higher the ratio is, the stronger the strength is. In general, the pairwise is the basic standard for evenly task allocation strategy and also is the initial task allocation strategy of optimal scheme in this paper.

In general, the ratio is real, it's not always integer. This situation causes some difficulties for initial evenly task allocation, because the task number for each virtual machine must be integer. So, a remainder append method is developed to assign initial tasks for all virtual machines. The main idea of remainder append method is to add the task pairwise remainder of one virtual machine to next task pairwise for another virtual machine, thus to ensure that all task pairwise are integer. For example, when the original ratio of map and reduce is 3.25. The methods will add the remainder of first three task pairwise to the fourth task pairwise, so the first three task pairwise is "<1, 3>", the fourth is "<1, 4>".

Definition 2: task slice. Due to relationship between map task number and reduce task number is n to 1 and the reduce task number is only 1, so it is inconvenient to adjust the task number of multiple virtual machine of same physical host. To better normalize the different type task number, task slice is defined to assure the relationship of three type task is one to one. Through transformation, each virtual machine holder multiple map task slices and same size reduce task slices, which provides a great convenience for task number adjustment of multiple virtual machines in same physical host.

Based on the task pairwise and task slice, the optimal scheme needs to dynamically adjust the task number between two virtual machine on same physical host. The goal of task adjustment is re-distributed task number according to virtual machine capability. To better introduce the adjustment process, we suppose virtual machine vm1 and vm2 are allocated to execute the job. Through initial task allocation and

task slice transmission, vm1 processes M1 push task, map task slice and reduce task slice, vm2 processes M2 push task, map task slice and reduce task slice. Based on algorithm 1, the single data transmission time (push task) on vm1 is t1, map execution time is t2 and reduce running time is t3. Also, the single data transmission time (push task) on vm2 is t4, map execution time is t5 and reduce running time is t6. According to the overlapping scheme, x map slices are moved from vm2 to vm1and y reduce task slices are moved from vm1 to vm2. This task adjusting process is must meet the following constraints:

$$\begin{cases} (M_1+x)(t_1+t_2)+(M_1-y)t_3 \le *\max\left(M_1(t_1+t_2+t_3), M_2(t_4+t_5+t_6)\right) \\ (M_2-x)(t_4+t_5)+(M_2+y)t_6 \le \max\left(M_1(t_1+t_2+t_3), M_2(t_4+t_5+t_6)\right) \\ x(t_1+t_2) \le (M_2-x)(t_4+t_5)+(M_2-1)t_6 \\ (M_1+x)(t_1+t_2) \le (M_2-x)(t_4+t_5)+(M_2+y-1)t_6 \end{cases} \tag{7}$$

In Equation (7), the first inequality means, after adjustment, task execution time of vm1 cannot exceed the original maximum execution time of two nodes. The second inequality also indicates task execution time of vm2 cannot exceed the original maximum execution time of two nodes. Third inequality implies all map slice should complete before the last reduce slice, which corresponds to the last map when move x map on vm2. The last inequality indicates all map slices should complete before the last reduce slice. To decrease the computational size, descending sort the factors according to the execution time, the Equation (7) can be converted into:

The goal of optimal shame is to minimize execution time of multiple virtual machine, which is to balance task compute time as much as possible. So, an adjustment objective function can be set to:

$$\begin{aligned} f(z) &= \left|t_{vm1}-t_{vm2}\right| \\ &= \left|(M_1+x)(t_1+t_2)+(M_1-y)t_3-(M_2-x)(t_4+t_5)-(M_2+y)t_6\right| \\ &= \left|M_1(t_1+t_2+t_3)+x(t_1+t_2+t_4+t_5)-y(t_3+t_6)-M_2(t_4+t_5+t_6)\right| \end{aligned} \tag{8}$$

Set $f(z)=0$, then:

$$\begin{aligned} &\because \quad M_1(t_1+t_2+t_3)+x(t_1+t_2+t_4+t_5)=y(t_3+t_6)+M_2(t_4+t_5+t_6) \\ &\therefore \quad y=\frac{t_1+t_2+t_4+t_5}{t_3+t_6}x+\frac{M_1(t_1+t_2+t_3)-M_2(t_4+t_5+t_6)}{t_3+t_6} \end{aligned} \tag{9}$$

The number of task slice adjustment could be calculated according to formula (7) and (9), as shown in Fig. 6. The map and reduce task slice adjustment value is the integer x, y value on red line.

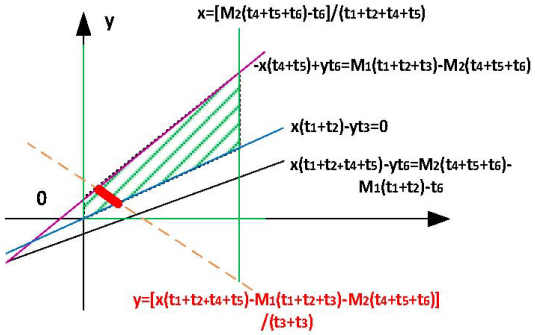


Fig. 6. Task adjustment function and range distribution

In summary, the process of overlapping optimal scheme for scene 2 consists of four steps. Firstly, the task pairwise is calculated according to the jobs information. For example, one user submits 6.4GB input data, 180 reduce tasks. Assuming the default data trunk size is 64M, so the number of push task and map task is both 1000, 1 reduce task need to deal with 5.6 map tasks, the task pairwise is "<1, 5.6>". Secondly, initial distributed process is begin to allocate tasks for all the virtual machines using the remainder append method according to task pairwise. After initial distributed process, task slice transformation is start

to assure the number of three task number is same. Thirdly, according the algorithm 1, the estimate task execution time of each virtual machine for current job has been calculated. Finally, using the physical host as unite, based on the estimated task execution time of three type task and task slice number, adjustment process is executed according to the Equation (8) and Equation (10) for all virtual machines of current host, such that minimizing the job execution time. The pseudo code of overlapping optimal scheme for scene 2 is described in algorithm 2.

Algorithm 2: Cross-phase optimization among map, shuffle and reduce in physical machine level

Input: virtual machine list $vlist$ of one physical machine and job information job_j.

Output: the task queue Q^{wait} of each virtual machine in $vlist$.

Procedure cross_phases_map_shuffle_reduce_optimization ($vlist$, job_j)

1: calculate the initial task pairwise $\langle C_m, 1 \rangle$ of job_j.
//initial task allocation
2: for each $vm_i \in vlist$
3: allocate C_m push, C_m map, 1 reduce task into Q_i^{wait} to assure that the task number is an integer using remainder append approach and transform the task to task slices.
4: use **Algorithm 1** heterogeneous_resources_optimization (job_j , $type_{vm_i}$) to calculate estimate execution time of push, map and reduce task slice.
5: end for
//task adjustment among virtual machines on same physical machine
6: initial virtual machine adjustment queue $vlist_{adjust} = \varnothing$.
7: for each $vm_i \in vlist$
8: if ($vlist_{adjust}.length > 1$)
9: for each $vm_k \in vlist_{adjust}$
10: compute task adjustment number $\langle x, y \rangle$ on vm_i and vm_j according to formulation (7) and (9).
11: if ($x != 0 \bigcup y != 0$)
12: update the map and reduce task slice of Q_i^{wait} and Q_j^{wait}.
13: end if
14: end for
15: end if
16: add vm_i into $vlist_{adjust}$.
17: end for
18: return the task queue Q^{wait} of each virtual machine in $vlist$.

Push-map optimization for scene 3. In scene 3, the main reason of push-map waste is that map tasks must wait until push phase completes. An optimal scheme for this scene is feasible developed to overlap push task and map task. Due to the different resource need of two type tasks, push task use network to transmit data trunk to virtual machines, while map tasks execute data on CPU or MEM, so the overlapping model between push and map task is reasonable. Through overlapping model, while the push task transmits the data trunk, the map task could still compute and analyze data. However, as the data from push task is the prerequisites of map task, portion overlapping is more realistic than fully overlapping. Therefore, overlapping degree is the key element of optimal scheme.

To better describe the method of how to get the overlapping degree, we suppose, some push tasks and map tasks are mapped to virtual machine *vm1*. At *t0* virtual machine *vm1* begin to execute tasks. In the push phase, *vm1* needs to transfer *M* data blocks, execute *N* map tasks in map phase. *M* and *N* may differ because of the influence of *algorithm 4*(cross-jobs optimal scheme). Based on the algorithm 1, the transmission time of each data block is *t1*, the execution time of each map is *t2*. Let us set *x* as the overlapping time for map task, in order to avoid waiting for the data transfer of push phase when map task begins, the overlapping x must meet the following constrains:

$$\begin{cases} t_0 & |\, t_1 \le t_2, N > M \\ t_0 + t_1 & |\, t_1 \le t_2, N = M \\ \left(M - \left(\dfrac{x - t_0}{t_1} \right) \right) * t_1 \le (N-1) * t_2 & |\, t_1 > t_2 \end{cases} \Rightarrow \begin{cases} x = t_0 & |\, t_1 \le t_2, N > M \\ x = t_0 + t_1 & |\, t_1 \le t_2, N = M \\ x \ge M t_1 + t_0 - (N-1) * t_2 & |\, t_1 > t_2 \end{cases} \quad \textbf{(10)}$$

In Equation (10), three constrains are correspond to three situations:

- When *t1* is less than or equal *t2* and *N* is greater than *M*, this situation indicates, before task begins, some data trunks are already transferred to virtual machine according to algorithm 4. Furthermore, the data transmission speed is faster than the computing speed of one map task. That is to say, when one map task does not complete yet, another input data has already been transferred, so the map task can start at *t0*.
- When *t1* is less than or equal *t2* and *N* is equal *M*, this situation implies that, before task begin, no data has been transferred. In addition, due to the data transmission speed is faster than map computing speed, so the overlapping time x of map task is the time of first data block transfer has been completed.
- When *t1* is greater than *t2*, this situation indicates that the data transfer speed is slower than the map computing speed, one map is completed while another push data is not completed. In this case, the remaining data blocks transmission must complete before the last map task slice execute. Furthermore, $(x - t_0)/t_1$ is must an integer because the overlapping time x of map task is the transfer competing time of some data blocks.

In summary, the process of optimal scheme for scene 3 is relatively easy. Firstly, according to the algorithm 1, the single push data transmission time and single map task execution time are calculated based on the type of current virtual machine. Secondly, the push data number and map task number are counted in task queue of current virtual machine. Thirdly, the overlapping time is calculated according to Equation (10). The Pseudo code of this optimal scheme is described in algorithm 3.

Algorithm 3: Cross-phases optimization between push and map

Input: (i) task queue Q_{task}^{hold}, (ii) current virtual machine type $type_i^{vm}$, (iii) the job start time t_{task}^{start} of first task.

Output: the start execution time of map phase t_{map}.

Procedure cross_phases_push_map_optimization (Q_{task}^{hold}, $type_{vm_i}$, t_{task}^{start})

1: calculate transmit data block number *M* of push task in Q_{task}^{hold}.
2: if ($M != 0$)
3: calculate map task number *N* in Q_{task}^{hold}.
4: get the job type and job information job_j by Q_{task}^{hold}.
5: compute data transmission time, map running time of single data trunk according to **the Algorithm 1** heterogeneous_resources_optimization **(** job_j **,** $type_{vm_i}$ **)**;
6: calculate the first map task start execution time t_{map} using the Equation (11).
7: else
8: get current time t_{now} as first map task start execution time $t_{now} \to t_{map}$.
9: end if
10: return t_{map}.

Cross-jobs optimization for scene 4. In scene 4, the main reason of resource waste is the flow-style scheduling model, that is to say, on the same virtual machine, the next round scheduling will be executed until the current tasks scheduling has been completed. However, one virtual machine consists of multiple type resources, the bandwidth resource is not fully used in current tasks because the next round push task

does not start. The bandwidth resource is in idle status at most time. One feasible optimal method is to start push data of the next job before the completion of previous job.

To overlap two different jobs, how to determine the overlapping time is the first key for the optimal scheme of scene 4. In this paper, we define an overlapping coefficient λ to measure the overlapping degree of two jobs. The overlapping coefficient is a percentage of the start scheduling time of next job and the total execution time of current job in the same virtual machine. That is to say, when the overlapping coefficient λ is set, virtual machine sends the idle command to scheduler for allocating next job tasks. Through changing the overlapping coefficient λ, the overlapping degree will be corresponding changed.

Otherwise, based on the overlapping coefficient, current job will be able to use the bandwidth resource at any time. So, the overlapping model of two different job could not affect the current job execution. That is, the two jobs will not use the bandwidth resource at same time, the next job task start only when the bandwidth is idle for current job. For assuring the no-conflict model of two different jobs, a monitor method is proposed in the optimal scheme for scene 4. The main idea is to monitor the bandwidth resource status before next job executes push task slice. When a new push task of next job starts, monitor the status of bandwidth resources. If the bandwidth is occupied, wait a fixed time, then listen again. If bandwidth is not occupied, begin data block transmission. When the data block transfer is completed, also sleep the fixed time, and listen again to judge whether to transmit the next push task. The monitor method guarantees the normal execution of previous job and avoids the long wait situation.

In summary, the optimal scheme of scene 4 consist of three steps. Firstly, according to the task queue, get the single task execution time of different type through the algorithm 1, and calculate the total execution time of all tasks of current job. Secondly, based on the overlapping coefficient and the total execution time, calculate the overlapping time. When achieve the overlapping time, an idle command is sent to the scheduler. When scheduler has received all the virtual machine idle commands, then schedules new tasks to virtual machines according to algorithm 2. Thirdly, the monitor method has been used to overlap execution the tasks of two different jobs. The pseudo code of this optimal scheme is described in algorithm 4.

Algorithm 4: Cross-jobs optimization

Input: (i) holding task queue Q_{task}^{hold}, (ii) waiting task queue Q_{task}^{wait},
(iii) overlapping coefficient λ, (iv) transmission sleep time t_{sleep}.

Output: the first task of next job start time t_{task}^{start}.

Procedure cross_jobs_optimization (Q_{task}^{hold}, Q_{task}^{wait}, λ, t_{sleep})

1: calculate the task total execute time $t_{excute}^{Q_{task}^{hold}}$ according to **Algorithm 1**.
2: compute the overlapping time by $t_{overlap} = t_{excute}^{Q_{task}^{hold}} * \lambda$.
4: while (current time $t_{now} < t_{overlap}$)
5: sleep;
6: end while
7: send idle command and fill new tasks into waiting task queue of next job.
// cross-jobs task optimization
8: while (task in Q_{task}^{hold} hasn't been completed)
9: while (bandwidth resource is idle)
10: if (the push task of Q_{task}^{wait} is not null)
11: transmit one data trunk of Q_{task}^{wait} from the public storage into HDFS of the current virtual machine.
12: update wait task queue Q_{task}^{wait} and remove the push task.
13: sleep t_{sleep} seconds after the transmission process of one data trunk has been completed.
14: end if
15: end while

16: sleep t_{sleep} seconds to next listening of bandwidth.
17: end while
18: retrieve the current time t_{now}.
19: return t_{now}.

3.2 CJCP Algorithm Description

According to the four resource waste scene of map-reduce job scheduling process in heterogeneous cloud, a cross-jobs-cross-phases (CJCP) map-reduce algorithm is developed to improve the utilization of underlying physical resources. For each resource waste scene, a corresponding optimal scheme is designed for minimizing the job execution time and costs, as shown in subsection 3.1. The CJCP algorithm is combination of four optimal scheme for four waste scene.

The process of CJCP algorithm consists of four steps. Firstly, when user submits a job to cloud, the input data are stored to the public storage and wait for the scheduler command. Then one data trunk, map task and reduce task are submitted to test virtual machine for training task execution, so as to get the single task test time of three type of task according to algorithm 1. Secondly, according to the algorithm 2, using physical host as unite, initial evenly distribute tasks for all virtual machines and dynamically adjust task number of any two virtual machines. Thirdly, algorithm 3 is used to parallel overlapping execute data transformation and map task for all virtual machines. After the map task begins, overlap the shuffle phase and reduce phase. Finally, when reach the overlapping time, the algorithm 4 begins to improve the utilization of bandwidth among two jobs execution process. The pseudo code of CJCP is described in algorithm 5.

Algorithm 5: cross-jobs-cross-phases task scheduling
Input: job list *jlist* .
Output: the execute result of job.
Procedure cross_jobs_corss_phases (*jlist*)
//task training
1: for each $job_j \in jlist$.
2: upload the file data to public storage.
3: execute the training test for job_j on fixed test virtual machine using the **Algorithm 1**.
4: end for
//task optimization
5: for each $job_i \in jlist$
6: while (physical machine is idle)
7: find virtual machine list *vlist* of physical machine.
// unevenly task allocation
8: use **Algorithm 2** (*vlist* , job_j) to adjust the task number among any two virtual machines.
//overlap push and map phases
9: **parallel,** $\forall vm_i \in vlist$, execute **Algorithm 3** to overlap the running process of push and map task.
//overlap tow jobs
10: **parallel,** $\forall vm_i \in vlist$, monitor the bandwidth status and use **Algorithm 4** to overlap tow jobs.
11: end while
12: end for
13: return the job results.

4 Experiments

In this section, we evaluate our proposed algorithm CJCP. Through the comparison with JOOP, SARS, CPOA algorithms, we run extensive test jobs and analyzed the performance of CJCP algorithm from three factors of multiple phases of single job, multiple jobs and systematic parameter.

4.1 Experimental Environment

To more truly simulate map-reduce cloud environment, four different type 17 physical hosts from National Supercomputing Changsha China Center, are selected to build a heterogeneous hadoop cluster. The 17 physical nodes include 5 Dell OptiPlex3010 hosts, 4 Sugon A440-G hosts, 4 HP DL320E Gen8 hosts, and 4 HP ProLiant DL160 hosts. Otherwise, the 17 physical hosts are connected by 1000M high-speed switch, in which are 16 compute nodes and 1 job submission node. The total cluster configures with Ubuntu12.04, hadoop1.20, and JDK 1.645. Table 1 shows the details of the physical hosts, including the host type, amount, CPU and MEM capacity, operating system and etc.

Table 1. Physical machine parameter

type	number	cores	MEM (g)	Disk (g)	system	role
G1:Dell OptiPlex3010	1	4 CPUs,Core3.3GHz	4	500	Ubuntu12.04	Name node
G1:Dell OptiPlex3010	4	4 CPUs,Core3.3GHz	4	500	Ubuntu12.04	Data node
G2:Sugon A440-G	4	8 CPUs,Xeon2.6GHz	8	500	Ubuntu12.04	Data node
G3:HP DL320E Gen8	4	16 CPUs,Xeon2.6GH	16	500	Ubuntu12.04	Data node
G4:HP ProLiant DL160	4	24CPUs,Xeon2.7GHz	24	500	Ubuntu12.04	Data node

To better simulate the jobs scheduling process, we extended the hadoop source code and customized five classes (JobDataTransmission, TaskSynchronizer, JobTrainer and CloudConfig), and modified JobTracker, TaskTracker and LaunchTaskAction three class on hadoop1.20, so as to achieve cross-jobs and cross-phases scheduling optimization. Where, JobDataTransmission refers to transfer data between the public storage and HDFS file system. TaskSynchronizer collects status of each compute node to provide decision support for the optimization of the next round scheduling. JobTrainer refers to test job and saving job history data. CloudConfig manages cloud scheduling information. A job trigger was added to monitor job status on JobTracker.

The process of experiments is that when new jobs have been submitted, the input data will be firstly stored into public storage. When the underlying physical resource idles, the input data are transited from public store to data node (HDFS), and the map and reduce task are mapped to virtual machine for execution, the final results will be written back to public storage. To more realistic, 9 different types of jobs are executed. Otherwise, in the 120 test jobs, the task number of map ranges from 160 to 800 and the size of input data varies from 10G to 50G.

4.2 Comparison Algorithm Description

To clearly present the performance of CJCP and show the optimization results on map, push, shuffle and reduce phases, several algorithms (JOOP, SARS, CPOA) are selected to compare with CJCP, and the details are described in Table 2. JOOP algorithm overlap map and shuffle phase for optimization, SARS algorithm optimal shuffle and reduce phases, CPOA concern on both overlapping of push-map and map-shuffle. JOOP establish a new scheduling model to overlap tasks queue for shuffle and map overlapping. SARS predict task execution time to overlap reduce task. CPOA takes feedback policy, the implementation of the results impact push transfer, the reduce execution results impact map process.

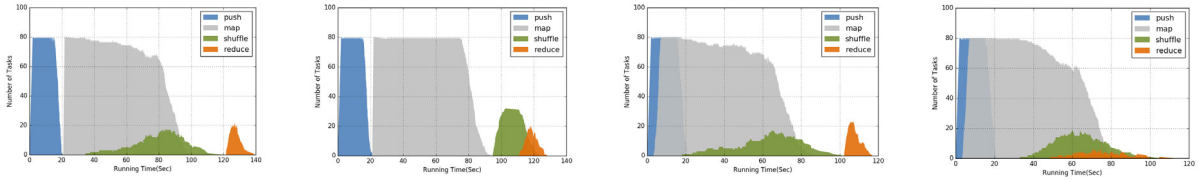
Table 2. The illustration of comparison algorithms

Short Name	Illustration
JOOP	Joint Optimization of Overlapping Phases in MapReduce [9]
SARS	A Self-Adaptive Scheduling Algorithm for Reduce Start Time [8]
CPOA	Cross-Phase Optimization in MapReduce [11]
CJCP	A Cross-job-Cross-phase map-reduce Scheduling Algorithm in Heterogeneous Cloud

4.3 Experimental Results

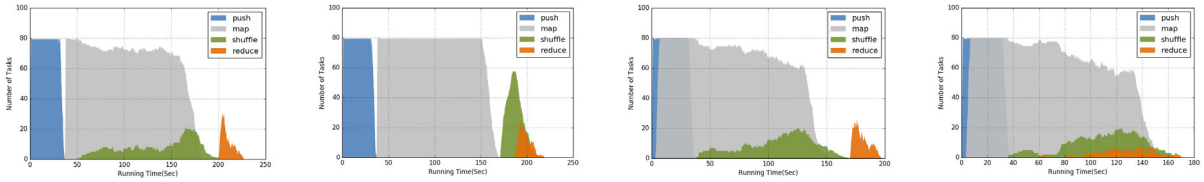
In this subsection, we verify the performance of CJCP algorithms from three factors on a single job, multiple jobs and system parameter.

Comparison of single job optimization. Fig. 7 and Fig. 8 show the different performance of four algorithms on two types of job (map size is 320 and 640 respectively). In the two figure, the relatively similar results are appeared and four algorithms perform different with each other on push, map, shuffle and reduce phases which are represented by four color range of blue, gray, green and orange. On 320 map tasks, JOOP algorithm consumes nearly 140 seconds, overlapping optimizes the map and shuffle two phases, we can see from gray and green area in Fig. 7. (a). SARS algorithm takes more than 120 seconds, the overlapping area is the shuffle and reduce phase, as the green and orange area in Fig. 7. (b). CPOA algorithm spends nearly 120 seconds, it overlaps push, map and shuffle three phases, as blue, gray and green area in Fig. 7. (c). CJCP algorithm costs nearly 110 seconds, optimizes the four phases, as Fig. 7 (d). Similarly, when the size of map tasks becomes 640, JOOP algorithm takes more than 220 seconds, SARS spends 210 seconds, CPOA needs 200 seconds, and CJCP takes 180 seconds. As can be seen from the two figures, CJCP algorithm has better overlapping performance than other three algorithms, and spends less time.



(a) JOOP (b) SARS (c) CPOA (d) CJCP

Fig. 7. Execution results of 320 Map tasks



(a) JOOP (b) SARS (c) CPOA (d) CJCP

Fig. 8. Execution results of 640 Map tasks

Table 3. Execution time of 320 map tasks

Algorithms	Push(s)	Map(s)	Shuffle(s)	Reduce(s)	Job Execution Time(s)	Improvement
JOOP	21	81	76	18	140	16.44%
SARS	22	73	29	19	129	11.34%
CPOA	21	83	85	15	118	8.25%
CPCJ	20	80	68	64	109	/

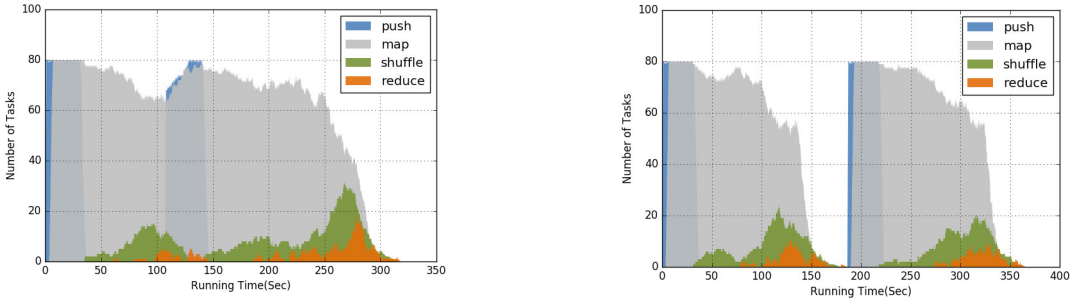
On two types of jobs, Table 4 and Table 5 shows the execution time of four algorithms. On each phase of push, map, shuffle, and reduce, four algorithms show different process time. In which, map and shuffle phase occupy a large portion time. JOOP algorithm is most time-consuming on map and shuffle phase (more than 150s). SARS algorithm takes less than JOOP. CPOA algorithm consumes up to 149 seconds, which is more than JOOP and SARS, but its shuffle phase is the minimum time-consuming. CJCP algorithm takes more time than JOOP and SARS on the map phases, but is minimized in the shuffle phase. On total time, CJCP algorithm is minimum, followed by the CPOA, SARS, and the finally one is JOOP. The last column of two tables shows time improvement of CJCP algorithm comparing to other algorithms. Through repeated experiments, we see that CJCP algorithm enhances the execution efficiency from 8% to 16%.

Table 4. Execution time of 640 map tasks

Algorithms	Push(s)	Map(s)	Shuffle(s)	Reduce(s)	Job Execution Time(s)	Improvement
JOOP	38	146	149	26	227	15.52%
SARS	37	135	42	33	220	12.41%
CPOA	36	149	132	27	198	9.17%
CPCJ	38	147	129	77	170	/

Comparison of Cross jobs optimization. (1) Overlapping Coefficient λ : In this subsection, the influence of overlapping coefficient on job performance of our algorithm is validated. Different overlapping coefficient will seriously affect the execution results of two same jobs. In this paper, two same type of jobs (640 map tasks) are executed under different overlapping coefficient.

Fig. 9 shows the performance changing under two different overlapping coefficient by using CJCP algorithm. In the figure, the overlapping degree of two jobs changed with the value of overlapping coefficient. As we can see, when the value of overlapping coefficient is 0.2, means the current job will overlap with next job and delay to complete because the bandwidth is occupy by the both two jobs. Map huger task size is, the delay time will be longer. Specifically when map tasks number is 640, the completion time of the first job delayed from 172s to 200s. When overlapping coefficient is 1.0, two jobs are flow-style executed, there is no overlapping for each job. The specific runtime under different overlapping coefficient values are shown in Tables 5.



$(a)\lambda = 0.2$ $(b)\lambda = 1.0$

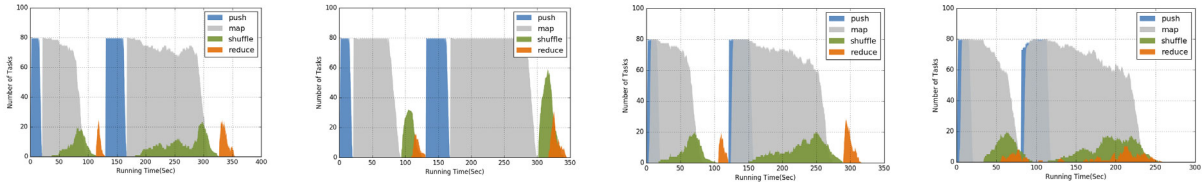
Fig. 9. Execution results with different overlapping coefficient

Table 5. Execution time with different overlapping coefficient

Value	Job1 Execution Time(s)	Job2 Execution Time(s)	Total Execution Time(s)	Job1 Delay Time Percent (no monitor)	Job1 Delay Time Percent (monitor)
0.1	210	187	316	22.09%	7.23%
0.2	200	182	318	16.27%	6.72%
0.3	196	179	320	13.95%	5.54%
0.4	193	185	322	12.21%	4.36%
0.5	186	176	326	8.13%	3.78%
0.6	188	174	336	9.30%	2.92%
0.7	183	177	344	6.39%	2.21%
0.8	179	172	350	4.06%	1.97%
0.9	177	180	359	2.90%	0.63%
1.0	172	173	366	0.00%	0.00%

As can be seen from Table 5, when the overlapping coefficient value is smaller, the delay time for the current job will be longer. From the table, we know that the monitor method of algorithm 4 helps to improve the job performance. By monitoring the bandwidth status of virtual machine, it reduced the bandwidth resource confliction of two jobs. Otherwise, by repeatedly running test instances, it is easy to find that, when the overlapping coefficient value is 0.6, the current job completion delay time is low, the total completion time is optimal, and the delay time ratio is 2.92%.

(2) Job execution time: Taking the overlapping coefficient value 0.6, we execute two type of jobs (the number of map task is 320 and 640), Fig. 10 shows the total execution results of four algorithms. Table 6 shows the total execution time of four algorithms. As can be seen from the figure and table, comparing to the JOOP algorithm, CJCP algorithm saves job execution time by 16.43%. Comparing to SARS algorithm, CJCP algorithm saves job execution time by 14.10%. Comparing to CPOA algorithm, CJCP algorithm saves job completion time about 11.03%.



(a) JOOP (b) SARS (c) CPOA (d) CJCP

Fig. 10. Execution results of two different jobs

Table 6. Execution time of two different jobs

Algorithm	Jobs Total Execution Time(s)	Improvement in total
JOOP	352	16.43%
SARS	346	14.10%
CPOA	320	11.03%
CPCJ	258	/

Systematic comparison. (1) Average job execution time: The average job execution time consists of data migration time, map and reduce running time. We run test jobs 20 times and changing map task number from 160 to 800, the comparison details of average job execution time are shown in Fig. 11. From Fig. 11, we can see that CJCP algorithm has better performance comparing to other three algorithms because the trend line of CJCP is most close to the bottom line. Along with the increasing of map tasks, CJCP algorithm presents better performance.

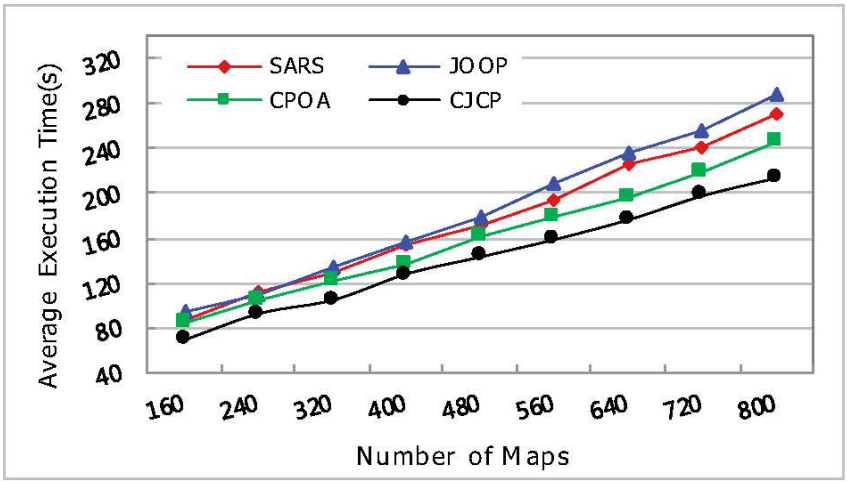
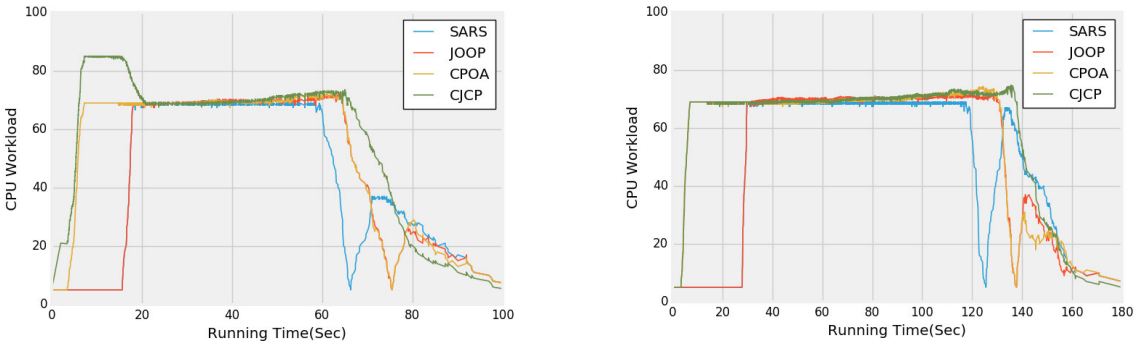


Fig. 11. Average execution time

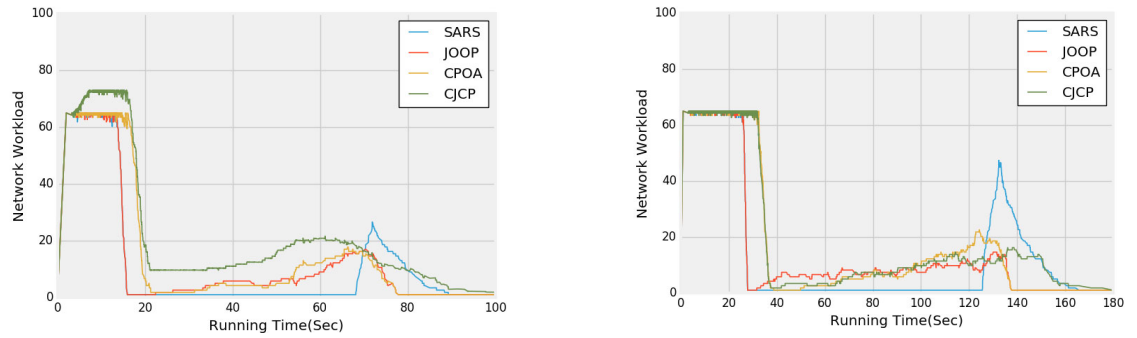
(2) CPU workload: We run two type of jobs on one virtual machines, the map task number is 320 and 640, CPU resource workload of four algorithms is demonstrates in Fig. 12. As is shown in the figure, at the beginning (from 0 to 20), the CPU workload fluctuations from 1% to 80%, because when the push phase runs, the network resources are busy while CPU workload is low. Along with the job execution, map tasks and other phases begin running, CPU resources are used frequently and gradually stabilize. Among the four algorithms, CJCP has highest CPU resource workload, because CJCP has greater overlapping degree than other three algorithms.



(a) 320 map tasks (b) 640 map tasks

Fig. 12. CPU workload

(3) Network workload: To correspond the CPU workload, for same two jobs, the map task number is 320 and 640, Network resource workload of four algorithms is demonstrates in Fig. 13. It is easy to find that two figures show a naturally opposite tendency because the intrinsic features among push, map and reduce tasks. From the Fig.13 (a), at the beginning (from 0 to 20), the Network workload fluctuations from 1% to 70%, because the push phase is running, the network resources are busy. From 20 to 60, Network workload reduce to 2% to10%, because the map task is running, the data transmission number cut down. From 60 to 80, Network workload increase again due to the task execution of shuffle phase. From 80 to 100, Network work load reduce again due to the execution of reduce phase. From two figures, we can see that CJCP algorithm performs more stable on the network workload than other three algorithms.



(a) 320 map tasks (b) 640 map tasks

Fig. 13. Network workload

5 Conclusions

With the urgent social needs on efficient and fast processing large-scale data, map-reduce cloud is becoming a powerful and popular platform. In this paper, we analyze and formulate the process of map-reduce job scheduling and present four resource waste scenes in the heterogeneous cloud. Based on the four waste scenes, we propose a cross-jobs-cross-phases map-reduce scheduling algorithm in this paper. Our algorithm consists of four optimal schemes, and respectively to four resource waste scenes. Firstly, we describe the heterogeneous resource waste scene, and a job training method and corresponding optimal scheme are developed to measure the virtual machine capability and the execution time of three different type tasks. Secondly, we develop another optimal scheme dynamically to adjust task number of multiple virtual machines on same host for unevenly tasks allocation. The goal of this scheme is to allocate more tasks to high-capability virtual machines, less tasks to low-capability virtual machines. Thirdly, considering the difference resource need feature of push and map tasks, an overlapping optimal method is

developed to speed up tasks execution time. Finally, we analyze the relationship between two jobs on the same virtual machine and designed another scheme to overlap execution two jobs. In the final scheme, to shield the interference of two jobs, a monitor method is used to solve the network occupancy problem that two jobs may simultaneously occupy bandwidth resources. The results verify that the CJCP performs better on cross phases, cross jobs scheduling than the JOOP, SARS, and CPOA algorithms.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (61174140, 61472127, 61272395); China Postdoctoral Science Foundation (2013M540628,2014T70767); Natural Science Foundation of Hunan Province (14JJ3107); Excellent Youth Scholars Project of Hunan Province (15B087).

References

[1] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Communications of the ACM 51(1)(2008) 107-113.

[2] M. Sun, H. Zhuang, X. Zhou, K.Lu, C. Li, HPSO: prefetching based Scheduling to improve data locality for MapReduce clusters, in: Sun X. et al. (Eds.), Algorithms and Architectures for Parallel Processing, ICA3PP 2014, Lecture Notes in Computer Science, vol. 8631, Springer, Cham, 2014, pp. 82-95.

[3] W. Wang, K. Zhu, L. Ying, J. Tan, L. Zhang, A throughput optimal algorithm for map task scheduling in MaprRduce with data locality, ACM Sigmetrics Performance Evaluation Review 40(4)(2013) 33-42.

[4] J. Li, X. Lin, X. Cui, Y. Ye, Improving the shuffle of hadoop MapReduce, in: Proc. Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on. IEEE, 2013.

[5] S. Gault, C. Perez, Dynamic scheduling of MapReduce shuffle under bandwidth constraints, in: Proc. Euro-Par 2014: Parallel Processing Workshops, 2014.

[6] W. Yu, Y. Wang, X. Que, C. Xu, Virtual shuffling for efficient data movement in MapReduce, Computers, IEEE Transactions on 64(2)(2015) 556-568.

[7] T. Luo, Y. Zhu, W. Wu, Y. Xu, D.-Z. Du, Online makespan minimization in MapReduce-like systems with complex reduce tasks, Optimization Letters 11(2)(2017) 271-277.

[8] Z. Tang, L. Jiang, J. Zhou, K. Li, K. Li, A self-adaptive scheduling algorithm for reduce start time, Future Generation Computer Systems 43(2015) 51-60.

[9] M. Lin, L. Zhang, A. Wierman, J. Tan, Joint optimization of overlapping phases in MapReduce, Performance Evaluation 70(10)(2013) 720-735.

[10] M.W. Rahman, X. Lu, N.S. Islam, D.K. Panda, HOMR: a hybrid approach to exploit maximum overlapping in MapReduce over high performance interconnects, in: Proc. the 28th ACM international conference on Supercomputing, 2014.

[11] B. Heintz, A. Chandra, J. Weissman, Cross-phase optimization in MapReduce, in: X. Li, J. Qiu (Eds.), Cloud Computing for Data-Intensive Applications, Springer, New York, 2014, pp. 277-302.

[12] D. Dahiphale, R. Karve, A.V. Vasilakos, H. Liu, Z. Yu, A. Chhajer, J. Wang, C. Wang, An advanced mapreduce: cloud mapreduce, enhancements and applications, Network and Service Management 11(1)(2014) 101-115.

[13] B. Palanisamy, A. Singh, L. Liu, Cost-effective resource provisioning for mapreduce in a cloud, Parallel and Distributed Systems 26(5)(2015) 1265-1279.

[14] L.-Y. Zhao, Y.-N. Wei, Y.-F. Zhang, Z.-X. He, Two-circle task scheduling algorithm based on MapReduce in cloud computing, in: Proc. Information Technology and Artificial Intelligence Conference (ITAIC), 2014 IEEE 7th Joint International. IEEE, 2014.

[15] K. Lee, S. Park, H. Lee, Improving MapReduce performance using block device reconfiguration in virtualized clouds, in: Proc. 2012 International Conference on Information Science and Technology (IST'12), 2012.

[16] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A.R. Butt, N. Fuller, MRONLINE: MapReduce online performance tuning, in: Proc. the 23rd International Symposium on High-performance Parallel and Distributed Computing, 2014.

[17] D. Wu, A. Gokhale, A self-tuning system based on application Profiling and Performance Analysis for optimizing Hadoop MapReduce cluster configuration, in: Proc. High Performance Computing (HiPC), 2013 20th International Conference on. IEEE, 2013.