J Grid Computing (2019) 17:419–446
https://doi.org/10.1007/s10723-019-09481-3

Hybrid Cloud Adaptive Scheduling Strategy for Heterogeneous Workloads

Li Chunlin · Tang Jianhang · Luo Youlong

Received: 12 April 2018 / Accepted: 11 March 2019 / Published online: 29 March 2019

Abstract With the advent of the era of big data, many companies have taken the most important steps in the hybrid cloud to handle large amounts of data. In a hybrid cloud environment, cloud burst technology enables applications to be processed at a lower cost in a private cloud and burst into the public cloud when the resources of the private cloud are exhausted. However, there are many challenges in hybrid cloud environment, such as the heterogeneous jobs, different cloud providers and how to deploy a new application with minimum monetary cost. In this paper, the efficient job scheduling approach for heterogeneous workloads in private cloud is proposed to ensure high resource utilization. Moreover, the task scheduling method based on BP neural network in hybrid cloud is proposed to ensure that the tasks can be completed within the specified deadline of the user. The experimental results show that the efficient job scheduling approach can veffectively reduce the job response time and improve the throughput of cluster. The task scheduling method can reduce the response time of tasks, improve QoS satisfaction rate and minimize the cost of public cloud.

Keywords Hybrid cloud · Heterogeneous workloads · BP neural network · Job scheduling

L. Chunlin (✉)
The Key Laboratory of Embedded System and Service Computing, Ministry of Education, Tongji University, Shanghai 200092, China
e-mail: chunlin74@aliyun.com

L. Chunlin · T. Jianhang · L. Youlong
Department of Computer Science, Wuhan University of Technology, Wuhan 430063, People's Republic of China

L. Chunlin
Shaanxi Key Laboratory of Network Data Analysis and Intelligent Processing, Xi'an University of Posts and Telecommunications, Xi'an Shaanxi 710121, China

1 Introduction

With the advent of the big data era, processing large amounts of data requires a lot of computing resources. The self-built private cloud of the enterprise can provide a high level of security and control mechanism for internal users. However, enterprises often need flexible computing resources. If an enterprise builds a sufficient private cloud cluster to meet the high resource demand of the holiday days, the cost of the enterprise is huge and a lot of resources are wasted when the resource demand is low. When the existing private cloud processing capacity of the enterprise is limited, reasonable selection of public cloud resources and hybrid cloud architecture for big data processing are cost-effective choices [1, 2]. Therefore, the key is how to give full play to the unique advantages of hybrid cloud, switch public cloud and private cloud smoothly, and achieve efficient processing of large data business.

With the burst of data (zettabytes predicted by 2020 [3]) and applications becoming more complex, private clouds struggle to adapt to the scale and scope required. Just by relying on private cloud, there is not enough capacity to run the desired analytics, or it is hard to get the results within a given deadline. In addition, the public cloud offers many new opportunities to make new insights beyond the scope of private clouds only. In this case, cloud bursting [4] quickly became popular among big data analytics users. Cloud bursting is an application deployment model in which the application runs in a private cloud or data center. A certain amount of computing or storage capabilities are dynamically requested from the cloud server when the demand for computing power reaches a peak. Hybrid cloud bursting can be an effective way to handle increasingly complex online analytics. There are many case studies including Netflix (https://www.netflex.com) and Pinterest (https://www.pinterest.com).

In a hybrid cloud environment, cloud bursting technology is used to processes data at a lower cost in a private cloud and burst into the public cloud when the resources of the private cloud are exhausted [5]. However, there are still many challenges in hybrid clouds [6, 7]. (1). The diversity of jobs submitted by users. For example, the resource requirements and data volume of big data services may be different in different time periods. (2). Heterogeneity of hybrid cloud resources. At present, enterprise-class IaaS private cloud platform is heterogeneous in physical machine. Public cloud resource types are more diverse, such as CPU, memory and bandwidth, etc. (3). QoS heterogeneity. Different big data services will have different QoS requirements. In short, the emergence and development of hybrid clouds is of great significance to users, especially the medium-sized and small enterprises.

To solve the above problems, the efficient adaptive scheduling strategy for heterogeneous workloads is proposed. The main idea of the proposed method is as follows First, the queuing model is established according to the job type and heterogeneous resources, so that the job selects the optimal heterogeneous private cloud resource. Then, when the job enters the queue of the corresponding resource pool, the task is scheduled to meet the user's constraints and apply for the optimal public cloud resource according to the requirements. The main contributions of this paper are summarized as follows:

(1) First, the jobs are classified by logistic regression, and the resources of heterogeneous private cloud are divided based on resource utilization. Then, the jobs are dispatched to corresponding resource pools in private cloud. In each resource pool, there are a number of job queues. The genetic algorithm is given to choose the optimal queues for jobs to maximize the utilization rate of private cloud.

(2) For reducing the monetary cost of public cloud under the deadline constraint, the execution time of tasks is predicted by Back Propagation neural network. If some tasks cannot meet the deadline in the private cloud according to the prediction results, the public cloud with the lowest cost would be applied.

(3) The extensive experiments show that our proposed efficient adaptive scheduling algorithm in hybrid cloud can improve the utilization rate and improve the throughput of private cloud. Moreover, it can also reduce the monetary cost of public cloud.

The rest of the paper is organized as follows: In Section 2, several related works are presented. Section 3 presents the model and algorithm for efficient adaptive scheduling of heterogeneous workloads. The comparison and analysis of experiment results are presented in Section 4. Section 5 is about the conclusions.

2 Related Work

Many researchers have done lots of work about cloud bursting to obtain elastic cloud resources with the minimal total cost in a hybrid cloud environment. Clemente-Castelló et al. [8] introduced a cost model designed for iterative MapReduce application, which runs in a hybrid cloud bursting scenario. It is a popular large-scale data-intensive application that can provide near real-time response capability. Li et al. [9] proposed the model and algorithm of cloud bursting hybrid cloud scheduling optimization including public cloud optimization, private cloud application optimization and private cloud job optimization routines. Xue et al. [10] proposed a solution to create a hybrid cloud platform, and they focused on bursting aspect and proved that their solution can use the spot price example to adapt the cloud bursting.

Cao et al. [11] proposes an online cost-aware service request scheduling strategy for Cloud Bursting in the hybrid cloud environment, which can make appropriate request placement decisions in real time and minimize the cost of renting public cloud resources with a low rejection rate. Clemente-Castelló et al. [12] presented a performance model for estimating the runtime of an iterative MapReduce application in a mixed cloud bursting scenario. Although there are many methods of cloud bursting in the hybrid cloud mentioned above, none of those approaches considered the heterogeneous jobs and heterogeneous private cloud, which may cause wastage of resources.

Nowadays, more and more enterprises adopt hybrid cloud to meet the demand of peak workload. It is important to study the scheduling algorithms of hybrid clouds. Wei et al. [13] proposed an innovative user request scheduling mechanism, which is dedicated to our system and considers multiple goals of minimizing the average completion time of user requests while ensuring load balancing. Arantes et al. [14] defined the corresponding scheduling optimization to ensure BOT Application requirements in terms of limited latency and time Reliability while minimizing budget. Liu et al. [15] proposed a job scheduling model and a task scheduling algorithm of multi-QoS constrained cost optimization for task scheduling in hybrid cloud, which is divided into three steps, namely task scheduling for private cloud, service selection and task scheduling for public cloud. Balagoni et al. [16] proposed a hybrid cloud scheduling algorithm to solve the problem that heterogeneity, uncertainty and supply time delay will affect the performance of the hybrid cloud environment. Different scheduling methods in hybrid clouds have been widely used. It is important to maximize the utilization of the private cloud and minimize the total cost of the hybrid cloud.

There are currently some works to study the multi-queue scheduling for cloud in terms of Software and System Architecture. Muñoz et al. [17] introduced the definition of the architecture of PaaS and SaaS developed on the federated hybrid cloud computing model, which includes storage resources separate from the computational model. Caballer et al. [18] provided a mechanism to coordinate computing resources across heterogeneous cloud infrastructures, with discussion of real-life use cases from scientific communities. Moreno et al. [19] represented a new cloud network federation framework for automatically providing Layer2 and Layer3 virtual networks to interconnect geographically distributed cloud infrastructure in a hybrid cloud scenario. Marosi [20] proposed a Federated cloud management architecture by using multiple queues to schedule heterogeneous jobs, which serves as the entry point for the cloud federation and combines the concepts of metabrokering, cloud agents and on-demand service deployment. Calatrava et al. [21] studied the further development of the Elastic Cloud Computing cluster. It was a tool which created self-managed cost-aware elastic clusters in hybrid clouds. Singh et al. [22] presented a hybrid adaptive multi-queue approach and Multi-level feedback queue scheduling algorithm to reduce energy consumption. Zuo et al. [23] proposed a scheduling method called interlaced peaks and resources are divided into three queues based on CPU, I/O, and memory load. This method can effectively balance the load and improve the resource utilization. Shorgin et al. [24] developed a cloud analytical model in terms of queuing system with multiple queues and batch arrivals to evaluate the average response time request. Singh et al. [25] proposed a smarter multi-queue job scheduling algorithm for cloud computing. The algorithm adapts to the virtualization characteristics of cloud computing and maintains the best resource fairness in energy efficiency and energy efficiency. Different from the above existing work, we first build the job queuing model based on the job classification and resource classification in this paper. Then the job arrival probability model in queuing theory is applied to select the job queue. We propose the job-adaptive scheduling algorithm based on queuing theory.

We then introduce some work which study the methods of cloud resource management, especially the method including time deadline. Montes et al. [26] presented a theoretical framework combining multiple resources and a single entity to develop large-scale distributed systems management techniques which is designed to optimize system performance, improve reliability and quality of service (QoS). Pop et al. [27] proposed a new approach for resource management to estimate the amount of resources required to schedule a set of aperiodic tasks, considering the deadline as the primary constraint while considering execution and data transmission costs. Yuan et al. [28] presented a profit maximization algorithm in hybrid clouds. This method can maximize the profit of the private cloud while meeting the deadline of tasks. Zuo et al. [29]

proposed a task-oriented multi-objective scheduling method based on ant colony optimization (MOSACO) to optimize the finite pool of public and private computing resources in a hybrid cloud computing environment according to deadline and cost constraints. Zuo et al. [30] proposed a self-adaptive learning Particle Swarm Optimization (SLPSO) to allocate the tasks of users in order to maximize the profit of Infrastructure as a Service provider while guaranteeing QoS in the hybrid clouds. Wang et al. [31] focused on the typical scenario of scheduling requests in a hybrid cloud and developed a hybrid cloud scheduler application to strike a good balance between the cost of the public cloud and the deadline for calculating requests. There are many resource management methods that consider deadlines, but we consider both deadlines and cost constraints in this paper. In the MapReduce task scheduling process, the tasks are scheduled to be dispatched to the private cloud as long as the deadline is met. Otherwise, the optimal public cloud resource needs to be applied to meet the deadline and cost constraints of the jobs.

3 Efficient Adaptive Scheduling Strategy for Heterogeneous Workloads in Hybrid Cloud

The architecture of scheduling in hybrid clouds is shown in Fig. 1. The hybrid cloud architecture mainly includes private cloud and public cloud. First, the user submits the job to the private cloud. The private cloud dispatches jobs to the appropriate queues based on job type and resource type. Then we perform task scheduling, check whether the deadline constraint is met after the private cloud scheduling. Otherwise the appropriate public cloud resources are applied, and then tasks are scheduled in the public cloud. Finally, the job execution results are fed back to the users.

When the existing scheduling method dealing with heterogeneous cluster resources, multiple types of jobs and different user demands, there may be problems such as low resource utilization and low user satisfaction of the cluster. To solve the above problem, we have proposed efficient adaptive scheduling strategy for heterogeneous workloads in hybrid cloud. First, the efficient job scheduling approach in private cloud is proposed. The queuing model is built according to the job type and heterogeneous resources, so that the job selects the optimal heterogeneous private cloud resource. The approach achieves the goal of reducing job response time and increasing the throughput of private cloud cluster operations. Then, we propose the task scheduling method based on BP neural network in hybrid cloud. When the jobs join the job queue of the corresponding resource pool, the task is scheduled to meet the user's constraints and apply for the optimal public cloud resources according to the requirements. The approach improves private cloud

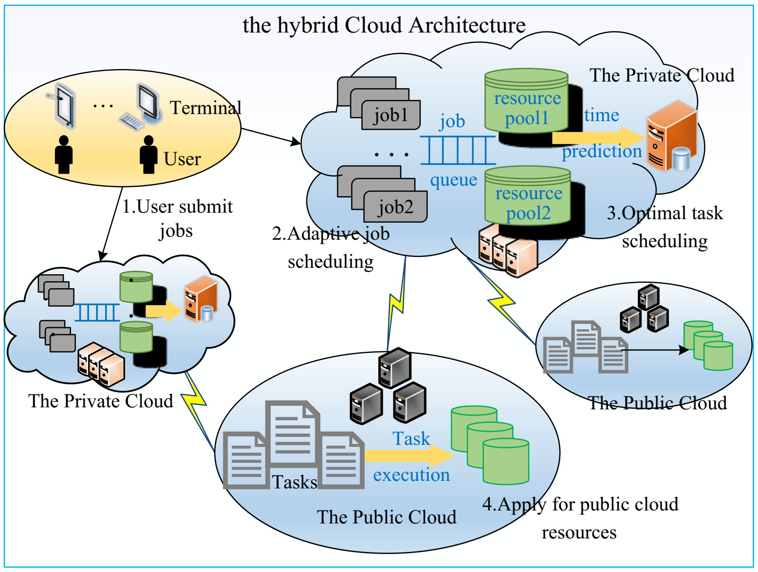


Fig. 1 Architecture of scheduling in hybrid clouds

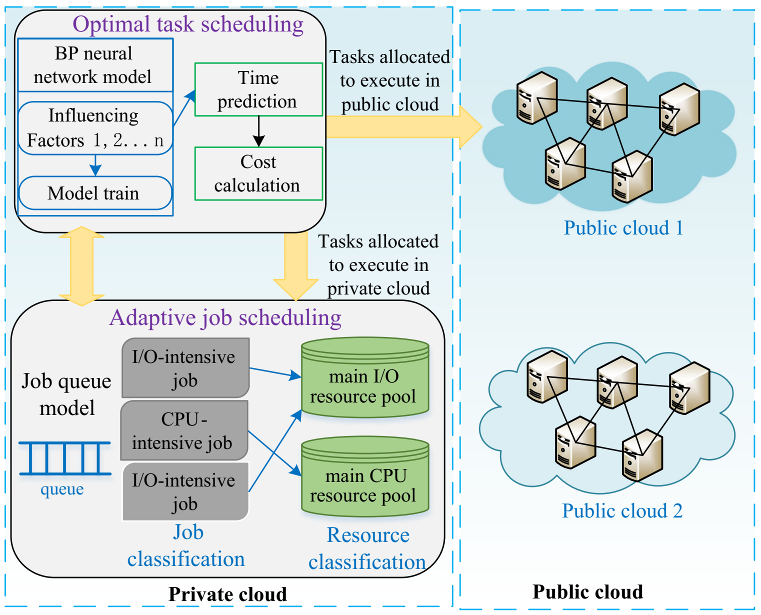


Fig. 2 Efficient adaptive scheduling model

resource utilization, reduce task response time and save public cloud costs. The efficient adaptive scheduling model for heterogeneous workloads is shown in Fig. 2.

3.1 Efficient Job Scheduling Approach in Private Cloud

In hybrid cloud environment, jobs submitted to hybrid clouds should be allocated to the private cloud first. If the private cloud could not meet the demands of users, the public cloud would be applied. In order to maximize utilization rate of the private cloud, efficient job scheduling approach based on queuing theory in heterogeneous private cloud is proposed. Firstly, jobs are classified into I/O-intensive and CPU-intensive by logistic regression; Second, the heterogeneous private cloud resources are divided into main I/O resource pool and main CPU resource pool according to resource utility ratio; Finally, the optimal choices of job queues are obtained according to the genetic algorithm, as shown in Fig. 3.

3.1.1 Efficient Job Scheduling Model in Private Cloud

Job Classification Tian et al. [32] divide map tasks into I/O-intensive and CPU-intensive based on job

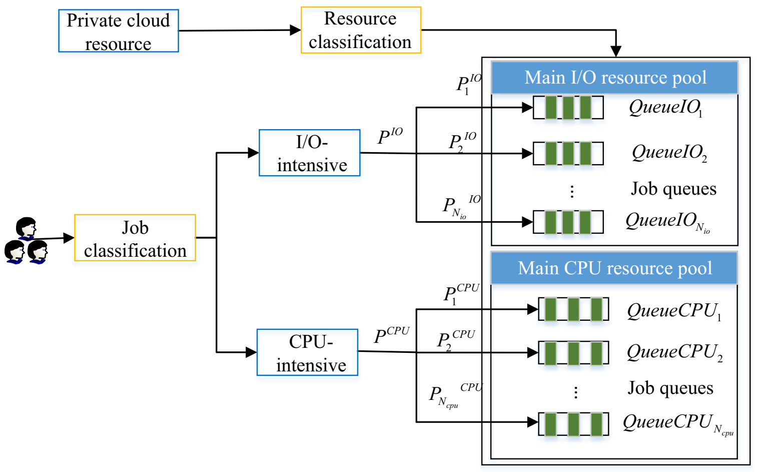


Fig. 3 Efficient job scheduling approach in private cloud

load. CPU-intensive jobs are characterized by a large amount of computation and consume CPU resources. In a multi-core architecture, CPU-intensive jobs can take advantage of multi-core for high performance. However, IO-intensive jobs are characterized by low CPU consumption and most of the job execution time is waiting for IO operations completed. For I/O intensive jobs, tasks assigned to a node need to access shared memory frequently, so the system performance will degrade and work execution time will be extended due to memory contention. Therefore, this paper considers the division of operations into I/O-intensive and CPU-intensive jobs based on workloads, thereby minimizing system memory contention, improving system performance and resource utilization.

The jobs with different types would be dispatched to corresponding resource pools in order to reduce response time. The Map tasks are divided into IO-intensive and CPU-intensive tasks. The ratio of Map input data denoted by MID to Map output data denoted by$MOD$is represented by ρ, and $\rho = MOD/MID$. $MTCT$ denotes the completion time of the map task. $DIOR$ denotes the I/O rate of the disk.$n$indicates the number of tasks. The I/O-intensive and CPU-intensive jobs are shown as follows.

$$n * (MID + MOD) \Big/ MTCT = n * ((1+\rho)MID) \Big/ MTCT \geq DIOR \quad (1)$$

$$n * (MID + MOD) \Big/ MTCT = n * ((1+\rho)MID) \Big/ MTCT < DIOR \quad (2)$$

From Formula (1) and (2), it can be seen that there are map input data, map execution time, disk I/O rate and other factors that affect the job classification. Most of the factors affecting job classification need to be determined after the job is executed. In this paper, we need to classify the job before the job being executed, and then dispatch it to the appropriate resource pool according to the job type. However, the map execution time is unknown, so the above formula classification cannot be directly applied. By analyzing the historical information, the principal component analysis method can be used to find that the two factors affecting the IO-intensive and CPU-intensive job classification. The two factors are the data size of the job ($DataSize$) and the computational complexity of the job ($Complexity$).The logistic regression is used to classify the jobs, and then the formula (1) and formula (2) are used to verify the type of the job. The calculation formula of the Sigmoid function of Logistic regression is shown as follows.

$$\sigma(z) = (1 + e^{-z})^{-1} \quad (3)$$

where $z = w_0 DataSize + w_1 Complexity$. Parameters ω_0 and ω_1 are estimated by the gradient ascent method.

Resource Classification We have studied the load type classification of MapReduce jobs. There are I/O resource requirements and CPU resource requirements for most jobs. The jobs can be executed in resource nodes, and their execution time mainly includes I/O time and CPU time. The I/O resource requirements of I/O-intensive jobs are significantly larger than CPU resource requirements. Most existing job scheduling algorithms rarely consider the job type to match the corresponding resource type. I/O-intensive jobs may be allocated to resources with high CPU performance, so that CPU performance is not better utilized. The job response time and the job throughput of the cluster are affected. Different types of jobs are allocated to the heterogeneous resource service queue, in order to avoid mismatching the job type and resource performance, resulting in unreasonable resource allocation. Therefore, the resources are divided into different resource pools according to the classification of heterogeneous resources, and then the jobs are dispatched to the corresponding resource pool according to the job type.

The computing resources and storage resources of the private cloud are mostly heterogeneous resources. The physical factors affecting the efficiency of job execution in heterogeneous private clouds mainly include disk IO performance, memory and CPU speed. The CPU performance is denoted by MIPS (Million instructions per second), and IOPS (Input/output operations per second) is used to represent I/O performance. In this paper, if the CPU performance of a node is better than IO performance, then the node is classified into main CPU resource pool. If the IO performance is better than CPU performance, the node is classified into main IO resource pool. S_{ir} denotes the average time of the$r$type jobs spent on$i$resources. $r = \{IO, CPU\}$ indicates IO-intensive and CPU-intensive jobs, and $i = \{IO, CPU\}$ indicates the IO time and CPU time of the job. For example, $S_{IO,IO}$ means the average time of I/O-intensive jobs spent on I/O resources. The average service time matrix can be calculated as follows.

$$[S_{ir}] = \begin{bmatrix} S_{IO,IO}, S_{IO,CPU} \\ S_{CPU,IO}, S_{CPU,CPU} \end{bmatrix} \tag{4}$$

Literature [33] proposes the utility calculation of heterogeneous resources and selects the resources with the minimum response time. In this paper, we improve the method and calculate the IO and CPU resource performance utility ratio of the node. Then we judge the resource type of the node for classifying the node resources. $\gamma_{i,r}$ denotes the physical performance influencing factors of $i$type resources of node r. The IO type resource utility ratio and CPU type resource utility ratio of the compute node k can be expressed by $\ell_{IO,k}$ and $\ell_{CPU,k}$ respectively, and the calculation formulas are shown as follows.

$$\ell_{IO,k} = \log \frac{S_{IO,IO} \times \gamma_{IO,k}}{S_{CPU,IO} \times \gamma_{CPU,k}} \Big/ \log \frac{S_{IO,IO} \times S_{CPU,CPU}}{S_{IO,CPU} \times S_{CPU,IO}} \tag{5}$$

$$\ell_{CPU,k} = 1 - \ell_{IO,k} \tag{6}$$

If $\ell_{I/O,k} > \ell_{CPU,k}$, node k will be classified into main I/O resource pool. Otherwise, the node will be classified into main CPU resource pool.

Job Queue Selection Based on Adaptive Genetic Algorithm As the cluster resources become larger and larger, the number of job queues per resource increases accordingly. The optimal job allocation probability calculation in the queuing model with a large amount of job submissions and a large number of job queues has a large search space. The basic genetic algorithm has the advantages of quickness, simplicity, and strong fault tolerance, but it has a slow convergence rate, is easy to fall into a local optimal solution, and has a long running time. Therefore, considering the time sensitivity and high quality of service requirements of most big data application services, this paper proposes an improved adaptive genetic algorithm to solve the optimal job arrival probability, so that the total job response time of the resource pool is minimized.

The Poisson process is applicable to the occurrence in a certain period of random time, which is a commonly used random process. In this paper, it is assumed that the user submission jobs of the hybrid cloud environment are independent of each other and conform to the Poisson distribution. That is, the arrival time interval of the jobs conforms to the negative exponential distribution. The process of submitting a multi-type job to a heterogeneous private cloud cluster and waiting for the cluster resources to serve the job can be regard as a mathematical model, that is, a queuing model. μ_j indicates the average service rate. $1/\mu_j$ denotes the average execution time of jobs in queue j. We assume that there areN_{rp}queues in private cloud, each of which is a $M/M/1$ queuing model. The average number Q_{avg} of jobs arriving in the job queue j can be obtained as follows.

$$Q_{avg} = (\rho_j)^2 \big/ (1 - \rho_j) \tag{7}$$

where $\rho_j (j = 1, 2, \ldots, N_{rp})$ indicates the service intensity of queue j and $\lambda_j = P_j \lambda$ indicates the arrival rate of job queue j. In this paper, the goal of job scheduling is to minimize the total response time of the private cloud, and the job scheduling problem can be expressed as the following programming problem.

$$\begin{aligned} \min T_{total}^{r} &= \min \sum_{j=1}^{N_{rp}} P_j T_j^r = \min \sum_{j=1}^{N_{rp}} \frac{\lambda_j}{\lambda} \\ &\times \left(\frac{Q_{avg} + Q_{init}}{\lambda_j} + \frac{1}{\mu_j} \right) \text{s.t.} \lambda_1 + \lambda_2 \\ &+ \cdots + \lambda_{N_{rp}} = \lambda \end{aligned} \tag{8}$$

where T_{total}^{r} denotes the total response time of private cloud. $P_j$denotes the probability of assigning a job to queue$j$. The average response time T_j^r is the sum of the waiting time and the service time of queue j. Q_{init} indicates the initial number of jobs in the queue.

With the development of private cloud, the number of job queues is increasing. Then, the job which has been submitted to private cloud should choose one of job queues based on the probability P_j of assigning a job to the queue j. The job queue is selected based on optimal probabilities $P_j \left(j = 1, 2, \ldots, N_{rp} \right)$ to minimize job response time T_{total}^{r} and balance the private cloud load. The optimal job allocation probability calculation in the queuing model with a large amount of job submissions and a large number of job queues has a large search space. Therefore, in this paper, we propose an improved adaptive genetic algorithm to solve the optimal job arrival probability, so that the total job response time of the resource pool is minimized. There are four factors about the job queue load: the initial number of jobs in the job queue, the resource utilization of the job queue, the amount of job data, and the complexity of the job calculation. Currently, the Hadoop YARN resource allocation unit mainly includes the CPU virtual core number and memory.

The i-th queue resource utilization ratio Ru_j is the sum of the CPU virtual core number and the memory utilization ratio, shown as follows.

$$\begin{aligned} Ru_j &= \delta_1 \cdot usedVCPU_j \Big/ VCPU_j \\ &+\delta_2 \cdot usedMem_j \Big/ Mem_j \end{aligned} \tag{9}$$

where $usedVCPU_j$ and $VCPU_j$ represent the number of CPU virtual cores used and the total number of CPU virtual cores respectively. $usedMem_j$ and Mem_j denote the used memory and the total memory of queue$j$respectively. And $\delta_1/\delta_2 = VCPU/Mem.\eta_1^{(j)}, \eta_2^{(j)}, \eta_3^{(j)}$ and $\eta_4^{(j)}$ represent the efficiency factor of the initial job numbers, resource utilization, job data volume, and the job calculation complexity of job queue j respectively. The calculation formulas are shown as follows.

$$\begin{cases} \eta_1^{(j)} = \frac{InitQ_{\max} - InitQ_j}{InitQ_{\max} - InitQ_{\min} + 1} \\ \eta_2^{(j)} = \frac{Ru_{\max} - Ru_j}{Ru_{\max} - Ru_{\min} + 1} \\ \eta_3^{(j)} = \frac{DataSize_{\max} - DataSize_j}{DataSize_{\max} - DataSize_{\min} + 1} \\ \eta_4^{(j)} = \frac{Complexity_{\max} - Complexity_j}{Complexity_{\max} - Complexity_{\min} + 1} \end{cases} \tag{10}$$

where $InitQ_{\max}$ and $InitQ_{\min}$ is the maximal and the minimal initial amount of jobs in private cloud. $InitQ_j$ is the initial amount of jobs in queue j. $Ru_{\max}$ and $Ru_{\min}$ denote maximal and minimal resource utilization rate of queues in private cloud respectively. Ru_jdenotes the resource utilization rate of queue j. $DataSize_{\max}$ and $DataSize_{\min}$ denote the maximal and minimal data size of queues in private cloud. $DataSize_j$ represents the data size of the queue $j.Complexity_{\max}$ and $Complexity_{\min}$ indicate the maximal and minimal computational complexity of queues in private cloud. $Complexity_j$ indicates computational complexity of queue j. The total efficiency coefficient $\eta^{(j)}$ of the j-th job queue is shown in formula (11).

$$\eta^{(j)} = \left(\prod_{i=1}^{4} \eta_{\mathrm{i}}^{(j)}\right)^{1/4} \tag{11}$$

The initial queue selection probability calculation formula according to the job queue load is shown as follows. The initial arrival rate λ_k of the job queue is shown as follows.

$$P_k = \eta^{(k)} \Big/ \sum_{k=1}^{\mathrm{n}} \eta^{(k)} \tag{12}$$

$$\lambda_k = \eta_k \times \lambda \Big/ \sum_{k=1}^{n} \eta_k \tag{13}$$

the similarity coefficient$factor$is shown in formula (14).

$$factor = (\mu + 1) \big/ \sigma \tag{14}$$

where the average value of the population fitness is reflected by the expectation μ, and the variance σ reflects the degree of individual fitness dispersion according to the probability theory. Therefore, the adaptive formulas for improving the crossover probability$p_c$and the mutation probability p_m based on the Logistic function are shown as follows. G_1, G_2, G_3, G_4 are constants.

$$p_c = \left(1 + e^{-G_1/factor}\right)^{-1} - G_2 \tag{15}$$

$$p_m = G_3 \left[G_4 \left(1 + e^{1/factor}\right)\right]^{-1} \tag{16}$$

The job queue selection algorithm based on the adaptive genetic algorithm is shown as follows.

(1) We initialize the queue job arrival probability P_k and job arrival rate λ_k for the two types of resource pools according to formula (12) and formula (13). The fitness function is $1 \big/ T_{total}^r$.
(2) We encoding the initial population individual according to the initial job arrival rate of the main I/O resource pool.
(3) The individual fitness values are calculated.
(4) The selection operator is applied to the population according to the fitness value of the individual, and the individual is selected to the next generation by the roulette selection method.
(5) The crossover probability P_c is calculated according to formula (15). Then we select two individuals to cross-pair and generate two new individuals. The mutation probability P_m is calculated according to formula (16). Then we perform a mutation operation on the paired individuals to generate new individuals, and reassess the chromosomal properties of the new population.
(6) Steps (3) to steps (5) are repeated until the iteration is terminated. The iterative termination condition is that the number of iterations exceeds the maximum number of iterations N_0 or the maximum fitness value and the average fitness value tend to be stable.
(7) The initial population individual is coded according to the initial job arrival rate of the main CPU resource pool. Then we perform steps (3) to step (6).

3.1.2 Efficient Job Scheduling Algorithm in Private Cloud

Algorithm 1 shows the pseudo-code of adaptive job scheduling algorithm in private cloud. First, the nodes are classified into main I/O resource pool or main CPU resource pool according to resource utility ratio (Algorithm 1 Line 1 $\sim$ 8). Then, the optimal queue choice probability P_j is calculated by genetic algorithm (Algorithm 1 Line 9 $\sim$ 11). The jobs are classified into CPU-intensive or I/O-intensive by logistic regression. Finally, the jobs are dispatched into a queue of main CPU resource pool or main I/O resource pool according to P_j respectively (Algorithm 1 Line 12 $\sim$ 19).

3.2 Task Scheduling Method Based on BP Neural Network in Hybrid Cloud

When the jobs have been allocated to corresponding resource pools and specific queues, the tasks of these jobs would be dispatched. The tasks would just utilize the private cloud resources if the private cloud can meet the demands of users. Otherwise, the public cloud with the minimal cost would be applied. In this section, the task scheduling method and algorithm based on BP neural network in hybrid cloud are proposed. First, BP neural network is given to predict the execution time of tasks. Then, task scheduling method would be given according to the prediction results.

In order to better illustrate the task scheduling process, we shown an example of task scheduling model in Fig. 4. First the user submits the tasks, and the tasks are waiting for scheduling in the queue. Then, the BP neural network is used to predict the task execution time. According to the history execution information, the influencing factors of the task execution can be obtained to improve the accuracy of the neural network prediction task execution time. If the current task execution time meets the deadline, the task is prioritized to the private cloud. Otherwise, the cost of the public cloud is further calculated. Under the condition that the deadline and cost constraints of the job are satisfied at the same time, the optimal public cloud resource is applied to execute the tasks. When the tasks execution is completed, the execution result will be returned, and the task scheduler will finally feed the results back to the users.

3.2.1 Task Scheduling Model Based on BP Neural Network in Hybrid Cloud

The Execution Time Prediction Most of jobs would be performed cyclically in large-scale systems. This means that each job with different data will be executed again and again. Therefore, it is important to analyze the historical information of these tasks. Factors that may affect the map task or reduce the task execution time are obtained from the parameters in the configuration file, such as “mapred-site.xml”.

In this paper, a MapReduce task prediction method based on BP neural network is proposed. We first extract the related factors of MapReduce task in historical job execution information as sample input. Then we iterative learning through BP neural network algorithm constantly to adjust the parameter weights until all samples converge. After determining parameters of the BP neural network model, the MapReduce task execution time is predicted according to the BP neural network model.

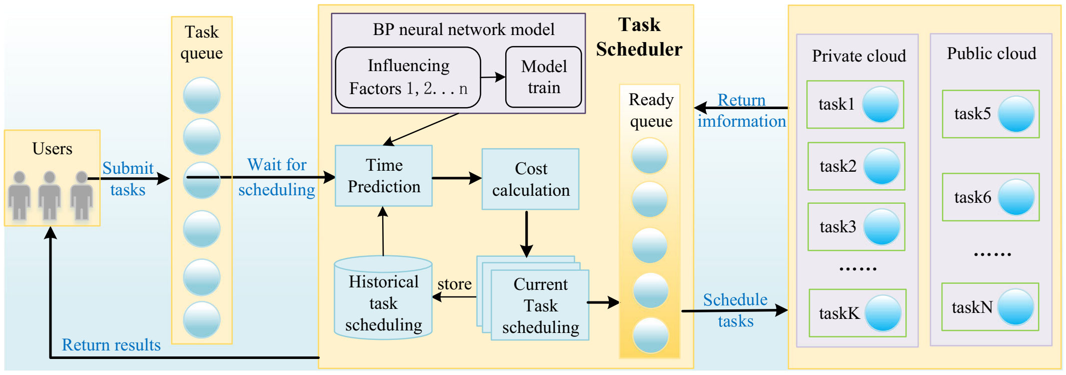


Fig. 4 An example of task scheduling model

Algorithm 1 Efficient job scheduling algorithm in private cloud.

```
Input: Job [$1, \dots, I$], Queue [$1, \dots, J$], resource nodes [$1, \dots, N$]
Output: Job scheduling result HahMap<Job [$1, \dots, I$], Queue [$1, \dots, J$] >
 1: for n $\in [1, \dots, N]$
 2:     Calculate $l_{IO,n}$ and $l_{CPU,n}$ according to Formula (5) (6)
 3:     if ($l_{IO,n} \geq l_{CPU,n}$) then
 4:         Classify the node $n$ into main I/O resource pool
 5:     else
 6:         Classify the node $n$ into main CPU resource pool
 7:     end if
 8: end for
 9: for each queue $\in$ Queue[1, ..., J] do
10:     $P_j \leftarrow Genetic(Queue[1, \dots, J])$
11: end for each
12: for each  job $\in$ Job[1, ..., I] do
13:     $\sigma(Z_j) > Logistic$(z)
14:     if $\sigma(Z_j) \leq 0.5$ then
15:         The job is classified into I/O-intensive and dispatched to corresponding queue of main I/O resource
            pool
16:     else
17:         The job is classified into CPU-intensive and dispatched to corresponding queue of main CPU resource
            pool
18:     end if
19: end for each
20: return HashMap<Job[$1, \dots, I$], Queue[$1, ldots, J$] >
```

In this paper, the MapReduce task prediction model is a three-layer BP neural network with a single hidden layer. There are n neuron in the input layer, and the input vector is $x = (x_1, x_2, ..., x_n)$, that is, the influencing factors of MapReduce task execution time. There is a neuron in the output layer, that is, the predicted time. The input vector of the output layer is $yi = (yi_1, yi_2, ..., yi_m)$, and the output vector of the output layer is yo, which indicates the predicted execution time of the task. The expected output vector is y, that is, the actual execution time of the task. The BP neural network learns through the forward propagation, predicts the task execution time, and judges whether the error value satisfies the requirement. Otherwise, the back-propagation learning adjustment weight is performed until the iteration is terminated.

The MapReduce task execution time prediction process is shown as follows:

(1) According to the MapReduce task execution characteristics, the influencing factors of the MapReduce task execution time is determined. The historical job samples are used to determine the factors that affect the MapReduce task execution time.

(2) The weight matrices W_A and W_B are generally initialized to small random non-zero values

(3) The training samples are inputted from $t = 1$ to $t = N$. Then we repeat steps (4) to (8) until all samples converged and go to step (9).

(4) The forward propagation is calculated. The input value A of implicit layer is calculated as $A = W_A^T \times X$, and the input value B of implicit layer is calculated as $B = W_B^T \times f_1(A)$ Thus, the calculation formula for the output value Y' of entire network is shown as follows.

$$Y' = f_2\left(W_B^T \times f_1\left(W_A^T \times X\right)\right) \tag{17}$$

(5) The sample error E_k is calculated. If the error meet requirements, the sample converges and the next sample is trained. Otherwise, we go to step (6), calculate back propagation and correct the

weight. The sample error calculation formula is shown as follows.

$$E_k = \frac{1}{2}\sum_{j=1}^{m}(d_{jk} - y'_{jk})^2 \tag{18}$$

(6) The back propagation of the output layer is calculated, and the weight $W_{B_{ij}}^{(t+1)}$ is adjusted. η denotes the learning rate, α denotes the momentum coefficient, and t represents the number of iterations. The calculation formula for weight adjustment in output layer is shown as follows.

$$\begin{aligned} W_{B_{ij}}^{(t+1)} &= W_{B_{ij}}^{(t)} - \eta \times (-(d_i - y'_i) \times f_2(B_j) \\ &\quad \times (1 - f_2(B_j))) \times f_2(A_i) + (W_{B_{ij}}^{(t)} \\ &\quad - W_{B_{ij}}^{(t-1)})\,(i = 1, ...q, j{=}1) \end{aligned} \tag{19}$$

(7) The weight $W_{A_{ij}}^{(t+1)}$ adjustment formula in output layer is shown as follows.

$$\begin{aligned} & W_{Aij}^{(t+1)} = W_{Aij}^{(t)} - \eta \times \sum\nolimits_{j=1}^{m}\Big(W_{Bij} \times (-(d_i - y'_i) \\ & \times f_2(B_j) \times (1 - f_2(B_j)))\Big) \times f_2(A_i) \times (1 - f_2(A_i)) \\ & + \alpha(W_{Aij}^{(t)} - W_{Aij}^{(t-1)}\Big) \qquad (i = 1, ...q, j = 1) \end{aligned} \tag{20}$$

(8) After correcting the weight W_A and W_B, we repeat step (4) to step (7) until the sample converges.
(9) We input the influencing factors of the predicted task and predict the execution time of the MapReduce task according to the BP neural network model.

Task Optimization Scheduling Model In hybrid clouds, when users submit jobs, there are some corresponding execution requirements, such as deadlines and cost constraints. In order to meet the individual requirements of different types of jobs, we will consider the job load, job deadline and job cost constraints to determine the job scheduling priority of the same job queue. The jobs in each queue would be sorted according to the priority. Let J_i be the job$i$with deadline D_i and cost constraint C_i. The data size is regarded as the main factor which influences the priority of I/O-intensive jobs. The computational complexity is chosen for CPU-intensive jobs. The I/O intensive job priority $JobP_i^{\mathrm{I/O}}$ and CPU intensive job priority $JobP_i^{\mathrm{CPU}}$ can be defined as follows. k_1, k_2, k_3 and k_4 are constants. $k_1 + k_2 = 1$ and $k_3 + k_4 = 1$.

$$\begin{cases} JobP_i^{\mathrm{I/O}} = k_1 \times DataSize_i \,/ D_i + k_2 \times C_i \\ JobP_i^{\mathrm{CPU}} = k_3 \times Complexity_i \,/ D_i + k_4 \times C_i \end{cases} \tag{21}$$

In a hybrid cloud environment, tasks are executed as much as possible in a private cloud to increase private cloud resource utilization. However, in order to ensure the user's constraints, when the private cloud resources are insufficient, we must apply for the optimal public cloud resources.

$J_i = \{m_{i1}, m_{i2}, ..., m_{im}, r_{i1}, r_{i2}, ..., r_{ir}\}$ denotes the i-th MapReduce job, which includes m map tasks and r reduce tasks. The triad $U = \{J_i, D_i, C_i\}$ denotes the job submitted by the user. Each job consists of multiple tasks. The tasks are indivisible, independent and non-preemptible. D_i indicates the deadline of the job, which is the latest completion time of the job, or the task of the latest completion time in the MapReduce task set. C_i indicates the maximum cost of completing job J_i. In the hybrid cloud scheduling model of this paper, we suppose that the cost of the private cloud is negligible, which means that C_i denotes the maximum cost of the public cloud for job J_i. The MapReduce tasks are defined as $m_{ik} = \{mW_{ik}, mD_{ik}\}$ and $r_{ik} = \{rW_{ik}, rD_{ik}\}$.$mW_{ik}$ and rW_{ik} respectively represent the workload of the Map task and the Reduce task. mD_{ik} and rD_{ik} represent the input data size of the Map task and the Reduce task. The relationship between m_{ik} and r_{ik} is shown as follows.

$$rD_{ik} = f \times \sum\nolimits_{u=1}^{W} mD_{iu}, 0 < f < 1 \tag{22}$$

where$f$denotes the conversion rate of input data size of reduce task to that of map task. W denotes the number of map tasks processed by a Reduce task.

The MapReduce tasks are assigned to execute on the corresponding hybrid cloud resource. The Rth resource of the hybrid cloud is defined as $Container_R = \{Cost_R, Stg_R, Cin_R, Cout_R, Band_R, Rft_R\}$.$Cost_R$ indicates the cost($/MIs) of public cloud, and the private cloud cost is negligible. Stg_R denotes the storage cost ($/MBs). Cin_R denotes input transmission cost($/s). $Cout_R$ denotes output transmission cost ($/s). $Band_R$ denotes the network bandwidth (MB/s). Rft_R represents the completion time of the task in resource R. In a hybrid cloud, some resources hold replicated data of MapReduce tasks. If there is not any replicated data of tasks in a resource, the data needs to be downloaded from other resources. Therefore, data transmission will occur. The transmission time $mDtt_{ik}$ of map task k of job i and the transmission time $rDtt_{ik'}$ of reduce task k' of job i can be achieved as follows.

$$\begin{cases} mDtt_{ik} = mD_{ik} \Big/ Band_R \\ rDtt_{ik'} = rD_{ik'} \Big/ Band_R \end{cases} \tag{23}$$

The enterprises that provide public cloud services would charge the service fee that includes computing cost, storage cost and data transmission cost. Therefore, the total monetary cost can be calculated by addition of computing cost, storage cost and data transmission cost. The cost $mCost[i, k, \mathrm{R}]$ of map task k of job i in hybrid cloud resource R and the cost $rCost[i, k', \mathrm{R}]$ of reduce task k' in public cloud can be achieved in (**??**).

$$\begin{cases} mCost[i, k, R] = Cost_R \cdot mW_{ik} + Stg_R \cdot mD_{ik} + mDtt_{ik} \cdot (Cin_R + Cout_R) + Cout_R \cdot \rho \cdot mD_{ik}/Band_R \\ rCost[i, k', R] = Cost_R \cdot rW_{ik'} + Stg_R \cdot rD_{ik'} + rDtt_{ik'} \cdot (Cin_R + Cout_R) + Cin_R \cdot rD_{ik'}/Band_R \end{cases} \tag{24}$$

$mEEt[i, k, R]$ represents the estimated execution time of map task k of job i in $Container_R$ and $rEEt[i, k', R]$ represent the estimated execution time of reduce task of reduce task k' of job i in $Container_{R'}$. The estimated execution time can be calculated by addition of the predicted execution time and data transmission time $mDtt_{ik}$ or $rDtt_{ik'}$.

$$\begin{cases} mEEt[i, k, R] = T_{map} + mDtt_{ik} \\ rEEt[i, k', R] = T_{reduce} + rDtt_{ik'} \end{cases} \tag{25}$$

The last task completion time of job J_i denotes the entire job completion time, that is, the maximum completion time of the task on the resource. The deadline for the job J_i is represented as follows.

$$\begin{aligned} Time = &\max_R \left(m_R \left(\sum_{k=1}^{m} mEEt[i, k, R] \cdot a_{i,k,R} + Rft_R \right) \right) \\ &+ \max_R \left(r_R \left(\sum_{k'=1}^{r} rEEt[i, k', R] \cdot a_{i,k',R} + Rft_R \right) \right) \end{aligned} \tag{26}$$

The task requires corresponding cost on the public cloud resource. The sum of the cost of all Map and Reduce tasks executed on the public cloud is equal to the total cost of the public cloud of the job J_i. The total public cloud cost cannot exceed the cost constraint of the job. The cost of public cloud is below.

$$\begin{aligned} Cost = &\sum\nolimits_{R=1}^{N_R} \sum\nolimits_{k=1}^{m} mCost[i, k, R] \times a_{i,k,R} \\ &+ \sum\nolimits_{R=1}^{N_R} \sum\nolimits_{k'=1} rCost[i, k', R] \times a_{i,k',R} \end{aligned} \tag{27}$$

In general, if the private cloud can meet the user's needs, there is no need to pay an additional fee to use the public cloud. Therefore, the scheduling problem can be defined as a target multi-dimensional knapsack selection problem (DO-MMKP). Thus, the task scheduling problem can be described as follows.

$$\min\,(Cost, Time) \tag{28}$$

$$\begin{aligned} \text{s.t.}\quad & Time \le D_i, Cost \le C_i \\ & a_{i,k,R} \in \{0, 1\},\ \sum_{R=1}^{N_R} a_{i,k,R} = 1;\ a_{i,k',R} \in \{0, 1\}, \\ & \sum_{R=1}^{N_R} a_{i,k',R} = 1 \\ & m_R \in \{0, 1\},\ m_R = \vee_{k=1,2,\ldots,m} a_{i,k,R};\ r_R \in \{0, 1\}, \\ & \quad r_R = \vee_{k'=1,2,\ldots,r} a_{i,k',R} \end{aligned} \tag{29}$$

where $Time \le D_i$ constrains the completion time of the tasks to be less than the deadline. $Cost \le C_i$ indicates that the public cloud cost to complete the task must be less than the cost constraint. $a_{i,k,R}$ and $a_{i,k',R}$ denote the binary picking variables. They represent whether a map task or reduce task is assigned to the container R or not. m_R and r_R represent whether a container R is used or not.

This programming problem is a combined optimization NP-complete problem, which can be seen as a variation of the knapsack problem. The time complexity of solving it by enumeration method is $O(m + r)^{N_R}$. Especially when the problem of hybrid cloud scheduling is relatively complicated, the optimal solution cannot be obtained in a limited time. Therefore, a heuristic algorithm that uses the Max-Min strategy is proposed to reduce the time complexity for solving the hybrid cloud resource allocation problem and solving the approximate optimal solution in a finite time. First,

Algorithm 2 Task scheduling algorithm in private cloud.

```
Input: The submitted job $J_i$, available resources of private cloud $ARP$
Output: Tuple<$PriMi$, $TRP_i$>
1:  $PriMi \leftarrow \varnothing, TRP_i \leftarrow \varnothing$
2:  sort(Task priority)
3:  for each task $\in J_i$ do
4:      $resourceR$←the first resource; $ftR \leftarrow \infty$; $tmp \leftarrow Rft_R$
6:      for each $R \in ARP$ do
7:          if $task$ is a map task then
8:              $T_{map} \leftarrow BP(X_{map})$
9:              if $mEET[i, k, R] + Rft_R < ftR$ then
10:                 $ftR \leftarrow mEET[i, k, R] + Rft_R$
11:                 $resourceR \leftarrow R$
12:             end if
13:         else $task$ is a reduce task
14:             $T_{reduce} \leftarrow BP(X_{reduce})$
15:             if $rEET[i, k', R] + Rft_R < ftR$ then
16:                 $ftR \leftarrow rEET[i, k', R] + Rft_R + maxFt(map\ task)$
17:                 $resourceR \leftarrow R$
18:             end if
19:         end if
20:     end for
21:     $Rft_R \leftarrow ftR$
22:     if $Rft_R > D_i$ then
23:         $TRP \leftarrow task_k$; $Rft_R \leftarrow tmp$
24:     else
25:         Dispatch $task_k$ to $resourceR$; Update($PriMi$)
26:     end if
27: end for
28: return < $PriMi, TRP_i$ >
```

the priority of tasks would be introduced. The tasks which are failed recently should be set to high priority. Hence, let the priority of failed map tasks be 5, let the priority of failed reduce tasks and reduce tasks be 10 and let the priority of map tasks be 20. That is, if both failed map task and failed reduce task apply resources simultaneously, the resource will be first assigned to the failed map task. The heuristic approach is to assign tasks with high priority to resources and use minimal completion time in the private cloud. If the private cloud cannot meet the user's needs, the public cloud with the lowest monetary cost is applied to meet the deadline for the tasks. That is to say, the public cloud could be regarded as the extension of private cloud in hybrid clouds if necessary.

3.2.2 Task Scheduling Algorithm in Hybrid Cloud

The task scheduling algorithm in hybrid cloud mainly includes the task scheduling algorithm in the private cloud and the task scheduling algorithm in the public cloud.

(1) Task scheduling algorithm in private cloud

Algorithm 2 describes the pseudocode for task scheduling in private cloud. First, we initialize $PriM_i$ and TRP_i. $PriM_i$ records the task mapping of job i in private cloud. TRP_i represents a task set that requires public cloud resources (Algorithm 2 Line 1). Then, we traverse all tasks in descending order of priority and all resource pools. (Algorithm 2 Line 3-6). The execution

Algorithm 3 Task scheduling algorithm in public cloud.

```
Input: $TRP_i$ represents a task set that requires public cloud resources, $PubR$ represents the public resource set
       $T_{init}$ denotes initialization time of public resource containers
Output: Triples $\langle PubRE_i, PubM_i, RT_i \rangle$
 1: $PubRE_i \leftarrow \varnothing$; $PubM_i \leftarrow \varnothing$; $SumCost_i \leftarrow \varnothing$
 2: for each $task \in TRP_i$ do
 3:     for each $R \in PubR$
 4:         if $task$ is a map task then
 5:             $SumCost_k \leftarrow SumCost_k + mCost[i, k, R]$
 6:             $T_{task} \leftarrow mEET[i, k, R]$
 7:         else $task$ is a reduce task
 8:             $SumCost_k \leftarrow SumCost_k + rCost[i, k, R]$
 9:             $T_{task} \leftarrow mEET[i, k', R] + \max Ft(map\ task)$
10:         end if
11:     end for
12:     Select the public resource $R$ with minimum monetary cost subject to $T_{task} + T_{init} < D_i$
13:     if $R$ exists then
14:         Create $R$ ; dispatch $task_k$ into $R$; Update $(PubM_i)$
15:     else
16:         return $\langle Null, Null, False \rangle$
17:     end if
18: end for
19: if $SumCost_i < C_i$ then
20:     return $\langle PubRE_i, PubM_i, True \rangle$
21: else
22:     return $\langle PubRE_i, PubM_i, False \rangle$
23: end if
```

time $EEt[i, k, j]$ of the tasks are predicted according to the BP neural network model. We find the minimum completion time and update the resource completion time Rft_j (Algorithm 2 Line 7-21). max $Ft(map\,task)$ denotes the maximum finish time of map tasks which be handled by reduce task k' (Algorithm 2 Line 17). Finally, it is judged whether the completion time of the tasks is less than the deadline of the jobs. If it is less than the deadline, we assign the task to the private cloud. Otherwise, we add the tasks to the collection PRP_i and reset the resource completion time (Algorithm 2 Line 22-26).

(2) Task scheduling algorithm in public cloud Algorithm 3 describes pseudocode for task scheduling in public cloud. First, we initialize $PubRE_i$, $PubM_i$ and RT_i.$PubRE_i$ indicates the set of containers of public resources to execute tasks. $PubM_i$ records the mapping of tasks assigned to public resources.RT_i denotes whether the task scheduling meets the deadline and cost constraint or not (Algorithm 3 Line 1). Then, we traverse the tasks in the collection PRP_i and traverse the public cloud resources (Algorithm 3 Line 2-3).The cost of executing tasks on public cloud resources and the execution time of tasks are calculated (Algorithm 3 Line 4-10). We find the best public cloud resource R with the lowest price and meeting deadline constraints (Algorithm 3 Line 12). If the public cloud resource exists, we create a public cloud instance and assign tasks to this public cloud (Algorithm 3 Line 13-17). After all the tasks that require the public cloud are allocated, we compare whether the total cost of the public cloud is less than the cost constraint. If it is less than the cost constraint, the scheduling is successful. Otherwise, the scheduling fails and message is returned to the users (Algorithm 3 Line 19-22).

Algorithm 4 The algorithm of efficient adaptive scheduling in hybrid cloud.

```
Input: $J = \{J_1, J_2, ... J_I\}$
Output: $isSucceed[1...n]$
1:  $isSucceed[1,...,n] \leftarrow false,\ jobPriority[1,...,n] \leftarrow 0$
2:  for each $J_i \in J$ do
3:      Obtain HashMap ⟨Job[$i$], Queue⌈$j$⌉⟩
4:  end for
5:  sort (job Priority)
6:  for each $J_i \in J$ by Job Priority desc
7:      Schedule tasks in Private Cloud according to Algorithm 2
8:      if $TRP_i = \varnothing$ then
9:          $isSucceed[i] \leftarrow true$
10:     else
11:         Schedule tasks in Public Cloud according to Algorithm 3
12:         if $RT_i == ture$ then
13:             $isSucceed[i] \leftarrow true$
14:         else
15:             $isSucceed[i] \leftarrow fale$
16:         end if
17:     end if
18: end for
19: return $isSucceed[i]$
```

3.3 Efficient Adaptive Scheduling Algorithm for Heterogeneous Workloads in Hybrid Cloud

3.3.1 Efficient Adaptive Scheduling Algorithm in Hybrid Clouds

The pseudo-code of the efficient adaptive scheduling algorithm in hybrid cloud is shown in Algorithm 4. First, the jobs are dispatched to the private cloud according to Algorithm 1(Algorithm 4 Line 2-4). Then, some of the tasks of these jobs are executed in private and others that cannot meet the deadline in private cloud would be dispatched to public cloud according to Algorithm 2. Finally, the optimal public cloud resources are selected according to Algorithm 3.

3.3.2 The Algorithm Time Complexity

In this section, we analyze the time complexity and approximation ratio of our proposed algorithm. The algorithm consists of two parts, job scheduling in private cloud and task scheduling in hybrid cloud.

The time complexity of the adaptive job scheduling algorithm in private cloud is mainly divided into three parts: (1) Job classification. Logistic regression is used to divide the jobs into IO-intensive jobs and CPU-intensive jobs. The maximum number of iterations is N_L. The time complexity of job classification is $O(N_L)$. (2) Resource classification. We assume that the number of cluster nodes is N_{node}. The time complexity of resource classification is $O(N_{node})$. (3) Genetic algorithm. The genetic algorithm updates the genetic factors according to the objective function. The number of jobs is $N_{job}.gens$ denotes the number of generations. The size of the population is $G.N_{Job}$ denotes the total number of jobs. The time complexity of genetic algorithm is $O(gens \cdot G \cdot N_{Job})$. Therefore, the total time complexity of adaptive job scheduling algorithm based on queuing theory in private cloud is $O(gens \cdot G \cdot N_{Job})$.

In optimal task scheduling algorithm in hybrid cloud, N_h denotes the number of hidden layers in BP neural network. N_{task} presents the number of tasks submitted by users. After training BP neural network, the time complexity of the task execution time prediction is $O(N_{task} \cdot N_h)$. N'_{task} denotes the number of tasks assigned to the public cloud. The time complexity of scheduling in private cloud is

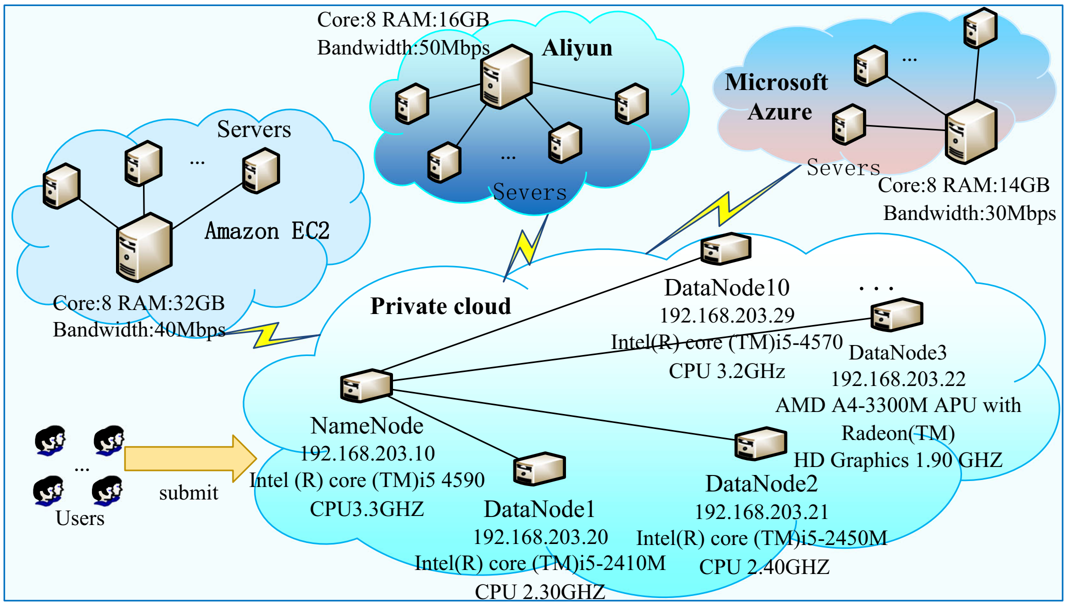


Fig. 5 The experimental environment

$O((N_{task} - N'_{task}) \cdot R \cdot N_h)$ where R indicates the number of resources in private cloud. The time cost of the scheduling in public cloud is $O\left(\left(N'_{task}\right) \cdot r \cdot N_h\right)$ where r indicates the number of resources in public cloud. Therefore, the total time complexity of task scheduling algorithm in hybrid cloud is $O\left(\left(N_{task} - N'_{task}\right) \cdot R \cdot N_h + \left(N'_{task}\right) \cdot r \cdot N_h\right)$.

In summary, the total time complexity of efficient adaptive scheduling strategy for heterogeneous workloads in hybrid cloud is $O\left(gens \cdot G \cdot N_{Job} + \left(N_{task} - N'_{task}\right) \cdot R \cdot N_h + \left(N'_{task}\right) \cdot r \cdot N_h\right)$.

4 Performance Evaluation

4.1 Experimental Environment

(1) Experimental Setup

In the experiment, we adopt Ubuntu 14.04.1 LTS operating system, JDK1.7.0_79, Strongswan5.5.1, OpenVPN2.3.2, Flex GateWay, Hadoop with stable version Hadoop 2.7.1, and development environment for Linux Eclipse 4.5.0 to form the entire experimental environment. The hybrid cloud environment includes private cloud and public cloud. The private cloud is a Hadoop cluster consisting of a NameNode node and 10 differently configured DataNode nodes. The resources provided by Alibaba Cloud, Microsoft Cloud, and Amazon Cloud are the public cloud resources. When the public cloud is required, the public cloud accesses the private cloud cluster. The hybrid cloud experimental environment architecture is shown in Fig. 5.

In the experiment, the public cloud resources of Microsoft Cloud, Amazon Cloud, and Alibaba Cloud are applied. The instance types, CPU types, and instance costs of some cloud servers provided by public cloud service providers are shown in Tables 1, 2, 3. The instance cost is

Table 1 Configurations of Microsoft Cloud

Facilitators	Instance types	CPU	Cost ($10^{-6}\$/s$)
Microsoft	Core:2 RAM:3.5GB Bandwidth:10Mbps	Intel Xeon(R)	28
Azure	Core:4 RAM:7GB Bandwidth:20Mbps	E5-2650 v2	73
	Core:8 RAM:14GB Bandwidth:30Mbps	2.60GHZ	228

Table 2 Configurations of Amazon Cloud

Facilitators	Instance types	CPU	Cost ($10^{-6}\$/s$)
Amazon	Core:2 RAM: 8GB Bandwidth:10Mbps	Intel Xeon(R)	27
EC2	Core:4 RAM:16GB Bandwidth:30Mbps	E5-2676 v3	65
	Core:8 RAM:32GB Bandwidth:40Mbps	2.4 GHZ	248

quantified as dollars per second ($/s) based on the instance price of the cloud server provided by the cloud service provider.

(2) Test Case and Experiment test parameters

In the experiments, the experimental data is from the dataset of the Stanford Network Analysis Project (SNAP) (http://snap.stanford.edu/). Different recommendation algorithms are selected, based on the project's k-nearest neighbor (KNN) recommender, the general user's recommender and the singular value decomposition (SVD) recommender. The data set contains purchase information and reviews for Amazon's approximately 1 million items. The size of the data set is approximately 64GB in total. The data size for each split is 128MB. That is to say, if the range of the size of a data set is $[2G, 64G]$, the range of the amount of map tasks is $[16, 500]$. The experiment test parameters are illustrated as follows. λ denotes the average job arrival rate ranging from 0.5 to 3. The population size G of genetic algorithm is set to 300. The range of the deadline D_i for job i is $[40, 310]$. We set the number of main IO resource pools and main CPU resource pools to 2 and 3 respectively. The number of map tasks and reduce tasks is $[16, 500]$ and [16,500] respectively. The number of jobs is ranging from 25 to 150.

(3) Evaluation metrics

The experiments are divided into two parts in this paper: experiments of job scheduling algorithm in heterogeneous private cloud and experiments of task scheduling algorithm in hybrid clouds. In the experiment of job scheduling in private cloud, the evaluation metrics of the algorithm performance are the average job response time and the throughput of cluster.

① The average job response time visually reflects the efficiency of the job execution in the cluster. The average job response time $T^r_{average}$ can be calculated as follows.

$$T^r_{average} = \sum_{i=1}^{N_{J_i}} (TC_i - TS_i) \Big/ N_{J_i} \quad (30)$$

where TC_i is the job completion time. TS_i denotes the job submission time, and N_{j_i} is the number of jobs.

② The job throughput of the cluster is the number of jobs processed by the cluster in a unit of time. In this paper, we define the throughput of the cluster as the number of jobs completed per minute. The calculation formula of throughput ($Throughput$) is shown as follows.

$$Throughput = jobN/t_{end} - t_{start} \quad (31)$$

In the experiment of task scheduling in hybrid cloud, the evaluation metrics of the algorithm performance are the average task waiting time, average task execution time, average task response time, QoS satisfaction rate and total monetary cost.

③ The average task waiting time means the time interval from the submission to the running of the task. The average task execution

Table 3 Configurations of Alibaba Cloud

Facilitators	Instance types	CPU	Cost ($10^{-6}\$/s$)
Aliyun	Core:1 RAM:1GB Bandwidth:10Mbps	Intel Xeon(R)	29
	Core:4 RAM:8GB Bandwidth:20Mbps	E5-2650 v2	89
	Core:8 RAM:16GB Bandwidth:50Mbps	2.60GHZ	235

time means the time interval from the start of the task to the end. The average task response time is obtained by the sum of the task waiting time and the task execution time.

④ QoS satisfaction rate denotes the percentage of tasks that can be completed in the private cloud within the deadline. This evaluation metric can reflect the utilization of private cloud resources. The QoS satisfaction rate can be obtained as follows.

$$QR = \frac{taskN_1}{taskN} \times 100\% \tag{32}$$

where $taskN_1$ indicates the number of tasks that can be completed in the private cloud during the deadline, and $taskN$ denotes the total number of tasks submitted.

⑤ The cost of public cloud spent on job execution is the sum of the cost of jobs using public cloud resources, when the private cloud resources cannot meet the deadline of jobs. $SumCost$ indicates the total cost of public cloud resources, and can be calculated as follows:

$$SumCost = \sum mCostF[i,k,j] \times a_{i,k,j} + \sum rCostF[i,k,j] \times a_{i,k,j} \tag{33}$$

where $mCostF[i,k,j]$ denotes the public cloud resource cost of the Map task. $mCostF[i,k,j]$ indicates the public cloud resource cost of the Reduce task. $a_{i,k,j}$ indicates whether the task uses the public cloud resource j. $a_{i,k,j} = 1$ denotes that the task uses the public cloud resource j, otherwise $a_{i,k,j} = 0$ denotes that the task does not use the public cloud resource j.

(4) Experimental benchmarking program

The benchmark test procedure determines the correctness and reliability of the proposed algorithm performance evaluation. We select the recommended jobs based on the real big data application scenario as the benchmark test assembly of this experiment. Due to the increasing variety of recommended algorithms in recent years, the complexity of the algorithms is very different and the recommendation systems of many different scenarios are applied everywhere. A brief introduction to the benchmark test procedures and functions is shown in Table 4.

In this paper, we use the recommendation system example of Mahout and the recommended jobs in the Mahout API build table. We take the user-based jobs as an example to show the recommended type jobs build process. The specific steps are as follows:

(1). We download and install Mahout. Then, we upload the data set to HDFS.

(2). The data files are loaded, and the interface DataModel is called to process the interaction data.

```
DataModel model = new FileDataModel
((new File("/path/to/dataset.csv")));
```

(3). The correlation coefficients between users are calculated.

```
UserSimilarity similarity = new
PearsonCorrelationSimilarity(model);
```

(4). We define similar users to make recommendations.

```
UserNeighborhood neighborhood = new
TheresholdUserNeighborhood(0.1,similarity,model);
```

(5). The recommended application is created.

```
UserBasedRecommender recommender = new
GenericUserBasedRecommender(nodel,neighborhooh,
similarity);
```

(6). The recommended application is compiled and packaged into the Jar file, and then it is named as the corresponding job number.

4.2 Experimental Results

4.2.1 Verification Experiments

Verification of job classification In this experiment, it is necessary to calculate the computational complexity and the amount of data of the jobs, and then obtain the job classification based on the logistics regression. The amount of data for the jobs can be determined when the user submits the jobs. The free software program *SourceMonitor* (http://www.campwoodsw.com/) is a commonly used code metric tool, which is used to look at the source code and identify the relative complexity of the module. We

Table 4 The introduction of Benchmark procedures

Benchmark procedures	Job number	brief introduction
GenericUserBasedRecommender	Job1	Recommendation algorithm based on user
GenericItemBasedRecommender	Job2	Recommendation algorithm based on Item
SlopeOneRecommender	Job3	Recommendation algorithm based on slope-one
SVDRecommender	Job4	Recommendation algorithm based on SVD
KnnItemBasedRecommender	Job5	Recommendation algorithm based on KNN
TreeClusteringRecommender	Job6	Recommendation algorithm based on Cluster

use the open source software *SourceMonitor* to calculate the complexity of static code as the computational complexity, and verifies its correctness through experiments. We run different types of jobs with each type of job repeated 100 times, calculating the jobs computational complexity, the data volume and the job type as the training set, iteratively calculating the weight w_0 and w_1. Let the number of iterations be 500 and the step size be 0.01. The computational complexity of the Hadoop job using the software SourceMonitor analysis is shown in Fig. 6.

As shown in Fig. 6, the calculation complexity of the jobs requiring a large amount of calculation is relatively high. The jobs for sorting require a large amount of calculations, so the calculation complexity of the jobs is higher. However, the jobs of the Random Writer require the IO operation to be more frequent, so the calculation complexity of the jobs is lower. In general, the software SourceMonitor can be used to calculate the computational complexity of the jobs.

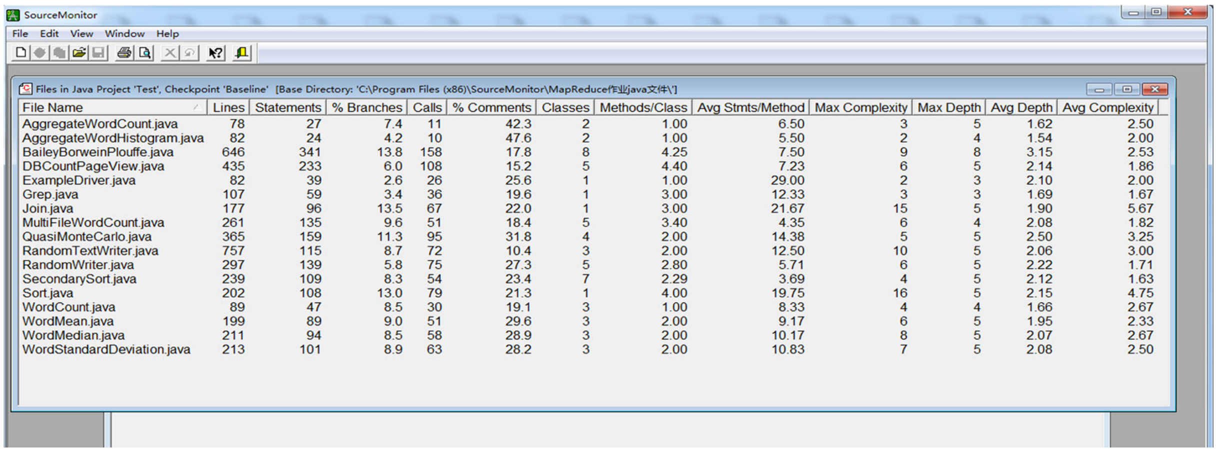
In order to judge the accuracy of the classification algorithm, we will calculate the average error rate e of the job classification this experiment. $errNum$ denotes the number of wrongly classified jobs, and $testNum$ denotes the total number of jobs. The average error rate can be described as follows.

$$e = \frac{errNum}{testNum} \times 100\% \tag{34}$$

In this experiment, we test 200 sets of data, the classification result of the jobs and the partial data of the real job type are shown in Table 5.

From the experimental results in Table 5, it can be counted that there are 7 sets of prediction errors in the 200 sets of data, so the average error rate e is 3.5%. Therefore, in this section we verify the feasibility and reliability of job classification based on Logistic regression.

Verification of resource classification In this paper, the FIO tool is used to test the number of read/write (I/O) operations per second (IOPS) of a node, and to verify the I/O performance of the resource node. The



Files in Java Project 'Test', Checkpoint 'Baseline' [Base Directory: 'C:\Program Files (x86)\SourceMonitor\MapReduce作业java文件\']

File Name	Lines	Statements	% Branches	Calls	% Comments	Classes	Methods/Class	Avg Stmts/Method	Max Complexity	Max Depth	Avg Depth	Avg Complexity
AggregateWordCount.java	78	27	7.4	11	42.3	2	1.00	6.50	3	5	1.62	2.50
AggregateWordHistogram.java	82	24	4.2	10	47.6	2	1.00	5.50	2	4	1.54	2.00
BaileyBorweinPlouffe.java	646	341	13.8	158	17.8	8	4.25	7.50	9	8	3.15	2.53
DBCountPageView.java	435	233	6.0	108	15.2	5	4.40	7.23	6	5	2.14	1.86
ExampleDriver.java	82	39	2.6	26	25.6	1	1.00	29.00	2	3	2.10	2.00
Grep.java	107	59	3.4	36	19.6	1	3.00	12.33	3	3	1.69	1.67
Join.java	177	96	13.5	67	22.0	1	3.00	21.67	15	5	1.90	5.67
MultiFileWordCount.java	261	135	9.6	51	18.4	5	3.40	4.35	6	4	2.08	1.82
QuasiMonteCarlo.java	365	159	11.3	95	31.8	4	2.00	14.38	5	5	2.50	3.25
RandomTextWriter.java	757	115	8.7	72	10.4	3	2.00	12.50	10	5	2.06	3.00
RandomWriter.java	297	139	5.8	75	27.3	5	2.80	5.71	6	5	2.22	1.71
SecondarySort.java	239	109	8.3	54	23.4	7	2.29	3.69	4	5	2.12	1.63
Sort.java	202	108	13.0	79	21.3	1	4.00	19.75	16	5	2.15	4.75
WordCount.java	89	47	8.5	30	19.1	3	1.00	8.33	4	4	1.66	2.67
WordMean.java	199	89	9.0	51	29.6	3	2.00	9.17	6	5	1.95	2.33
WordMedian.java	211	94	8.5	58	28.9	3	2.00	10.17	8	5	2.07	2.67
WordStandardDeviation.java	213	101	8.9	63	28.2	3	2.00	10.83	7	5	2.08	2.50

Fig. 6 The computational complexity of the Hadoop job

Table 5 The part data of job classification test

Host name	Read IOPS	Write IOPS	MIPS
s1	52	21	575
s2	39	15	600
s3	2362	1002	475
s4	491	203	650
s5	108	46	825
s6	69	29	600
s7	41	17	665
s8	198	80	825
s9	27	10	575
s10	503	264	800

commands to test disk read and write performance are as follows.

```
fio-filename=/dev/sda2-direct=1      -thread-rw=randrw
-iodepth 1 -rqmixread=70 -ioengine=psync -numjobs=32
-runtime=180 -bs=4k -size =10G -group_reporting
-name=mytest -ioscheduler=noop
```

where, bs = 4k means that the size of a single I/O is 4k, size = 10G means that the test file size is 10G, numjobs = 32 means that the number of test threads is 32, and runtime = 180 means that the test time is 180 seconds. We test each node with the same parameters. The disk read and write performance of all nodes is counted, and then the CPU performance of each node is evaluated. The test results are shown in Table 6.

According to the I/O performance and CPU performance of each node in Table 6, we run specific benchmark program on each node, and calculate the CPU time and IO time of each node. The resource classification results are shown in Table 7.

Table 7 node resource type classification results

Table 6 IO performance and CPU performance of nodes

JobID	Prediction job type	Actual job type
Job_1	CPU-intensive job	CPU-intensive job
Job_2	CPU-intensive job	CPU-intensive job
Job_3	I/O-intensive job	I/O-intensive job
Job_4	CPU-intensive job	CPU-intensive job
Job_5	CPU-intensive job	CPU-intensive job
Job_6	I/O-intensive job	I/O-intensive job
...	...	...

Table 7 Node resource type classification results

Host name	classification	Host name	classification
s1	CPU resources node	s6	CPU resources node
s2	CPU resources node	s7	CPU resources node
s3	I/O resources node	s8	I/O resources node
s4	I/O resources node	s9	CPU resources node
s5	CPU resources node	s10	I/O resources node

Verification of job queue selection based on improved adaptive genetic algorithm In this section, the experiment classifies the jobs submitted at negative exponential time intervals and selects the corresponding resource pool, and then the job selects the corresponding job queue. The job arrival rate is calculated based on the improved adaptive genetic algorithm, and then the probability of the job selecting a job queue is calculated. After the genetic algorithm converges, the subsequent arriving jobs directly select the job queue according to the probability, reduce the job response time and improve the job throughput of the cluster.

The algorithm parameters are set as follows. The total job arrival rate is 2, The number of job queues in the strong IO resource pool and strong CPU resource pool is 2 and 3 respectively. The population size is 300, and the number of iterations is 100. The adjustment formulas of mutation probability and crossover probability are shown in formula (13) and (14), respectively. $G_1 = 2$, $G_2 = 0.1$, $G_3 = 0$, $G_4 = 7$. In order to ensure the reliability and stability of the experiment, the experiment is repeated independently for 50 times to obtain an average value, and the objective function accuracy is 10^{-2}.

The basic genetic algorithm and adaptive genetic algorithm are used to solve the job arrival rate, and

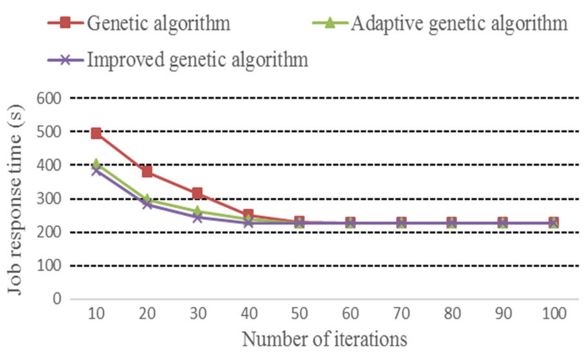


Fig. 7 Performance comparison of improved genetic algorithm

compared with the improved adaptive genetic algorithm proposed in this paper. We use the strong CPU resource pool total job response time to test the convergence of the algorithm. The experimental results are shown in Fig. 7. As shown in Fig. 7, the performance of the proposed improved adaptive genetic algorithm is better than others. The convergence speed is significantly improved compared with the traditional genetic algorithm and adaptive genetic algorithm. This is because the improved adaptive genetic algorithm loads the initial individuals according to the job queue, and at the same time improves the crossover probability and the mutation probability adjustment formula.

Verification of Task execution time prediction based on BP neural network The average absolute percentage error (MAPE) of the task execution time prediction is of the form.

$$MAPE = \left(\sum_{i=1}^{N} \left| \frac{Y - Y'}{Y} \right| \times 100\% \right) \Big/ N \qquad (35)$$

where Y indicates the actual execution time of the tasks. Y' indicates the predicted execution time of the task, and N indicates the number of samples.

According to the BP neural network model, the benchmark program and different data volumes are run in the hybrid cloud environment. The predicted time and actual execution time of the tasks are collected through the log file. The absolute percentage error is calculated to evaluate the task execution time prediction based on BP neural network. The experimental results are shown in Fig. 8. Figure 8 shows the estimated value, actual value, and error accuracy of the time prediction, which are obtained by BP neural network for different types of tasks in a hybrid cloud environment. It can be seen that the error rate of task execution time prediction is mostly less than 20%, and a few points reached 30% or 40%. The MAPE value is 12.88% according to the formula (33). Therefore, the BP neural network model predicts the task execution time, and the prediction accuracy can improve the performance of the MapReduce task scheduling algorithm.

4.2.2 The Comparison of Efficient Job Scheduling Algorithm with Other Scheduling Algorithms

In order to evaluate the average job response time and the throughput of our proposed efficient job scheduling algorithm (ASQT) based on queuing theory in private cloud, we compare the proposed algorithm with FIFO scheduling algorithm, Fair scheduling algorithm and COSHH algorithm [34]. In the experiment, we consider the impact of the average arrival rate of

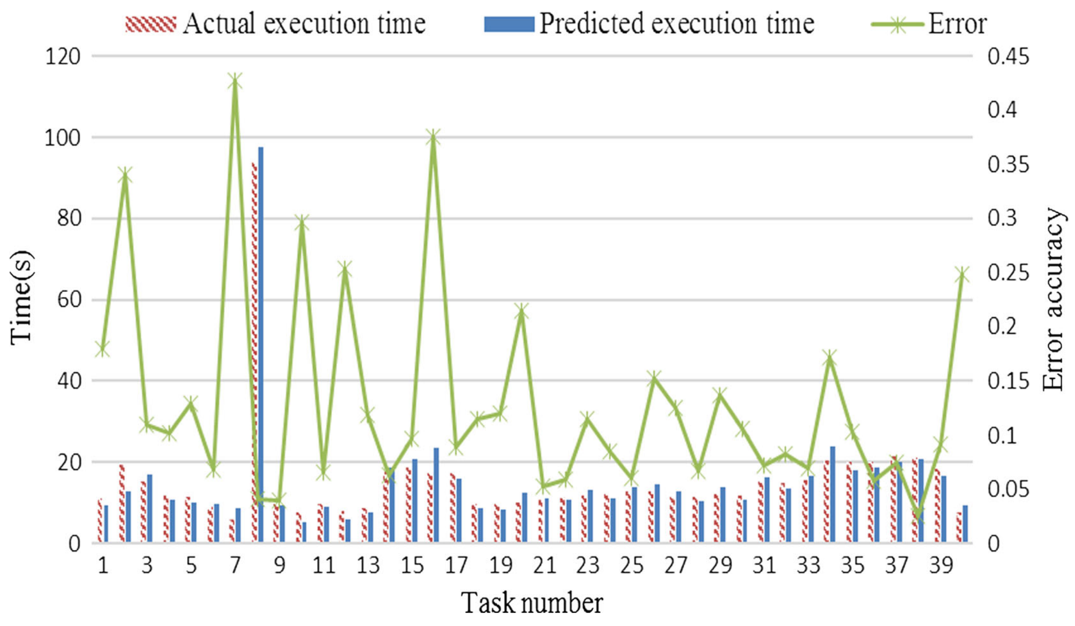
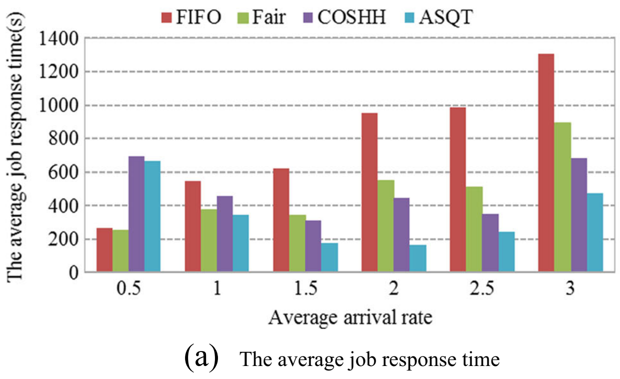


Fig. 8 Task execution time prediction and error



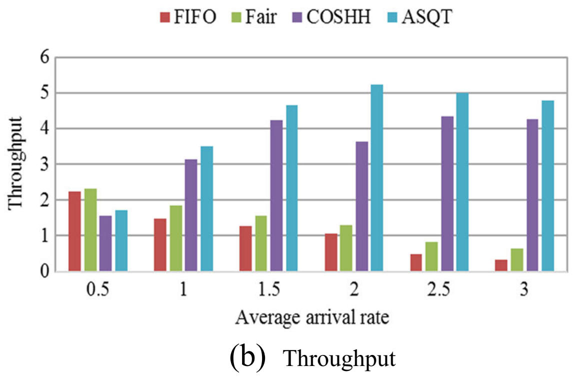


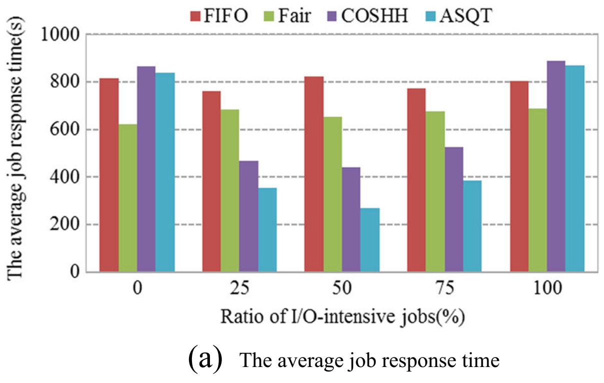
Fig. 9 The effect of the average arrival rate of jobs

jobs and the ratio of job types on the performance of the algorithm. Finally, the performance comparison results of the algorithms after the queuing model is stabilized are presented.

In the experiment, we set different job arrival rates, which changed from 0.5 to 3. The effects of the job arrival rate on the average job response time and throughput are shown in Fig. 9a and b. As the arrival rate of the jobs increases, the performance of FIFO algorithm and Fair algorithm becomes lower. The performance of COSHH algorithm and the proposed ASQT algorithm increase first and then decrease. When the job arrival rate is 2, the performance of the proposed ASQT algorithm is optimized. When the job arrival rate is 2, the average job response time of the proposed ASQT algorithm is 82.37%, 69.56% and 62.5% lower than FIFO, Fair and COSHH algorithm respectively as shown in Fig. 9a. The throughput of the proposed ASQT algorithm is 79.54%, 75.33% and 30.59% higher than FIFO, Fair and COSHH algorithm as shown in Fig. 9b.

The reason for the above experimental results is illustrated as follows. As the job arrival rate of increases, the type of jobs and the type of demands are evenly distributed. However, the resource requirements of the jobs cannot meet the round scheduling of Fair and FIFO algorithm. In the COSHH scheduling algorithm, the resource execution efficiency in the heterogeneous environment is quite different so that the search space is larger and the matching of jobs and resources is inefficient. The proposed ASQT algorithm is based on job type classification and resource type classification. The efficiency is high in matching the job and resource. For multiple consecutively arrived jobs, the job queue selection is based on the queuing model to improve the resource utilization of the job queue. Therefore, the performance of the proposed ASQT algorithm is better than Fair, FIFO and COSHH algorithms.

The ratio of job types is defined as the percentage of the number of IO-intensive jobs to the total number of jobs. The effect of the ratio of job types on the



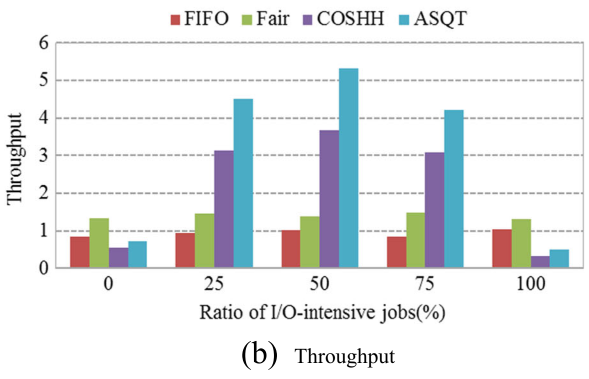


Fig. 10 The effect of the ratio of job types

average job response time and throughput is shown in Fig. 10a and b. The ratio of job types is different, and the performance of FIFO and Fair algorithm is stable. As the ratio of job types increases, the performance of FIFO and Fair algorithm is lower. The performance of the proposed ASQT algorithm and the COSHH algorithm is increased first and then decreased. When the ratio of job type is 50, the performance of the proposed ASQT algorithm is optimal. The response time of the proposed ASQT algorithm is 67.47%, 58.89% and38.81% lower than FIFO, Fair and COSHH algorithm respectively as shown in Fig. 10a. The throughput is 80.82%, 73.87% and 30.82% higher than FIFO, Fair and COSHH algorithm respectively as shown in Fig. 10b.

The reasons for the above experimental results are explained below. The FIFO and Fair algorithms do not consider the scheduling of job types, so the ratio of job types have no significant impact on their performance. The COSHH algorithm considers the job classification to match the resource. If the job type is single, the job increases the complexity of assigning resources, so the ratio of job types does not uniformly improve the performance of the COSHH algorithm. Before the scheduling of the ASQT algorithm, the jobs are divided into CPU-intensive and IO-intensive jobs according to the job characteristics. The resource pools are also divided into main IO and main CPU resource pool according to the resource utility ratio. The time complexity of matching the corresponding resources in the proposed ASQT algorithm is low. Therefore, the performance of the proposed ASQT algorithm is better than Fair, FIFO and COSHH algorithms.

The system performance comparison results of the average job response time and throughput in different algorithms are shown in Fig. 11a and b. The performance of the proposed ASQT algorithm is obviously better than the other three scheduling algorithms. The average job response time of the proposed ASQT algorithm is 76.79%,56.68% and 45.38% lower than FIFO, Fair and COSHH algorithms as shown in Fig. 11a. The throughput is 80.18%, 65.87% and 31.19% higher than FIFO, Fair and COSHH algorithms as shown in Fig. 11b.

The reasons for the above experimental results are explained below. The FIFO scheduling algorithm is mainly for single-user single-type jobs, so the response time will be longer when scheduling multi-user and multi-type jobs. The Fair algorithm is mainly for multi-user and multi-type jobs. The COSHH algorithm is optimized for heterogeneous cluster resources. The proposed ASQT algorithm targets multi-type jobs and multi-type resources. The jobs of different type are submitted to the corresponding resource pools. The queuing theory method is used to select the appropriate job queue, so it can reduce the average response time and improve the throughput.

4.2.3 The Comparison of Task Scheduling Algorithm with Other Scheduling Algorithms

In order to evaluate the performance of the task scheduling algorithm (*OTSTP*) based on BP neural network in hybrid cloud, we compare the proposed *OTSTP* algorithm with the FIFO algorithm and the ASQ algorithm [35]. By changing the different settings in each experimental group, we study how the

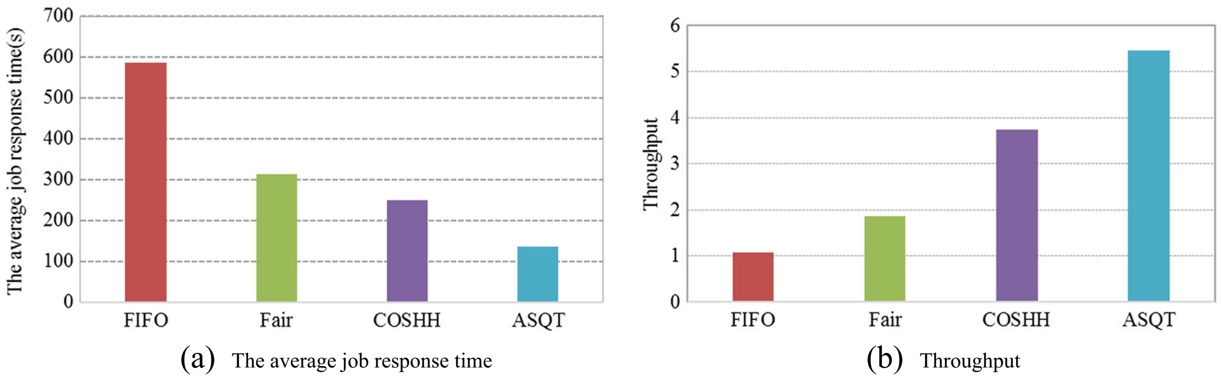


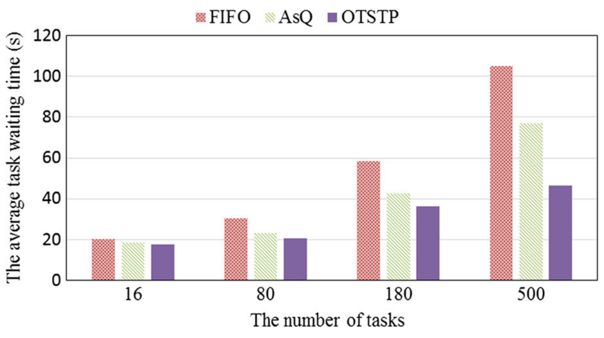
Fig. 11 The performance of the system

various parameters affect the effectiveness of the task scheduling algorithm in the hybrid cloud.

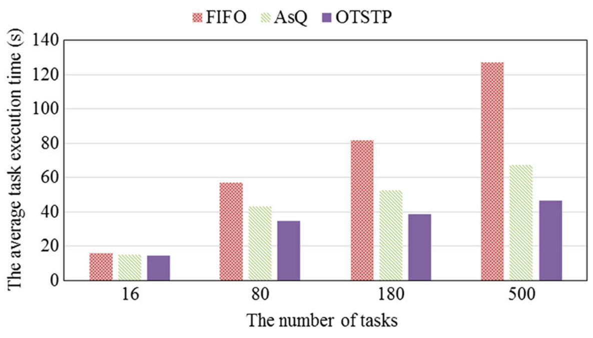
In the experiment, the number of Map tasks is set to 16, 80, 180, 500 respectively, and the corresponding data volume is 2G, 10G, 23G, 64G. The number of Reduce tasks is set to 1, and the job deadline is not limited. The effects of the number of tasks on the average waiting time, the average execution time and the average response time of the tasks are shown in Fig. 12a, b and c. When the number of tasks is 500, the average task waiting time of the proposed *OTSTP* algorithm is 56.04% and 40.41% smaller than FIFO and AsQ algorithm respectively as shown in Fig. 12a. The average task execution time of the proposed *OTSTP* algorithm is 63.50% and 30.93% smaller than FIFO and AsQ algorithm respectively as shown in Fig. 12b. The average task response time of the proposed *OTSTP* algorithm is 60.12% and 35.8% smaller than FIFO and AsQ algorithm respectively as shown in Fig. 12c. Overall, the performance of the proposed *OTSTP* algorithm is better than FIFO and AsQ algorithm.

The reasons for the above experimental results are explained below. The FIFO algorithm does not consider the matching of task load and resource performance. AsQ considers the task load, but it is not suitable for multiple job types and heterogeneous cluster resources. The proposed *OTSTP* algorithm allocates resources with appropriate performance according to the execution time of the task. Therefore, the performance of the proposed *OTSTP* algorithm is better than FIFO and AsQ algorithm.

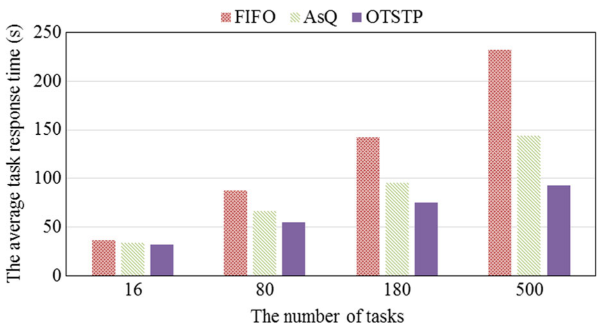
The effect of the deadline on the QoS satisfaction is shown in Fig. 13a, b and c. The deadline is set from 50 seconds to 310 seconds. In Fig. 13a, when the number of Map tasks is 48 and the deadline is 130s, the QoS satisfaction rates of the proposed *OTSTP* algorithm, FIFO and AsQ algorithm are 100% 69% and 89% respectively. In Fig. 13b, when the number of Map tasks is 96 and the deadline is 150s, the QoS satisfaction rate of the proposed *OTSTP* algorithm reaches 100%. However, when the deadline is 180s, the QoS satisfaction rates of FIFO and AsQ algorithms reach 84% and 100% respectively. In Fig. 13c, the num-



(a) The average task waiting time

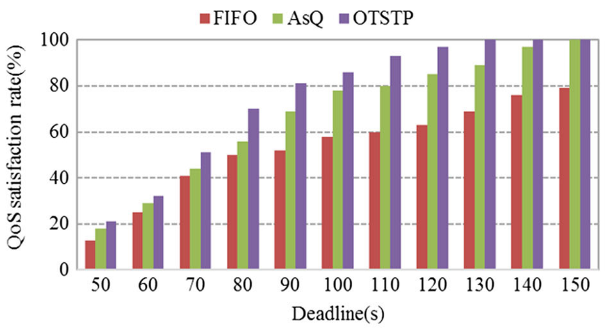


(b) The average task execution time

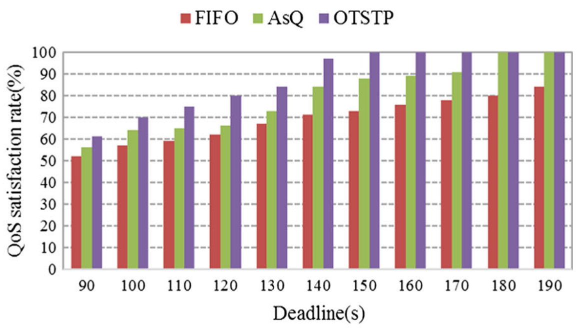


(c) The average task response time

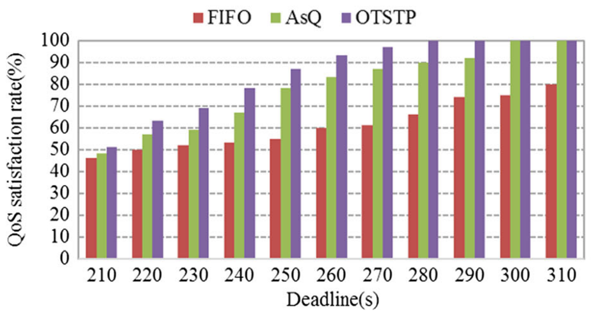
Fig. 12 The effect of the number of tasks



(a) 80 tasks



(b) 180 tasks



(c) 500 tasks

Fig. 13 The effect of deadline

ber of Map tasks is 180, the QoS satisfaction rates of the proposed *OTSTP* algorithm, FIFO and AsQ algorithms are 100%, 74% and 92% respectively. Overall, the performance of the proposed *OTSTP* algorithm is better than AsQ and FIFO algorithms in terms of QoS satisfaction rate.

The reasons for the above experimental results are explained below. The FIFO algorithm does not consider the problem of reasonable allocation of heterogeneous private cloud resources. AsQ algorithm does not consider the matching of job types and resource types. The proposed *OTSTP* algorithm makes full use of heterogeneous private cloud resources, and the task execution efficiency is high. Moreover, the error rate of prediction task execution time is low, and the resources are rationally utilized. Therefore, the proposed *OTSTP* algorithm can handle more tasks than the other two scheduling algorithms, and the percentage of tasks completed within the deadline is higher than other algorithms.

Fig. 14 Cost analysis for the different number of tasks

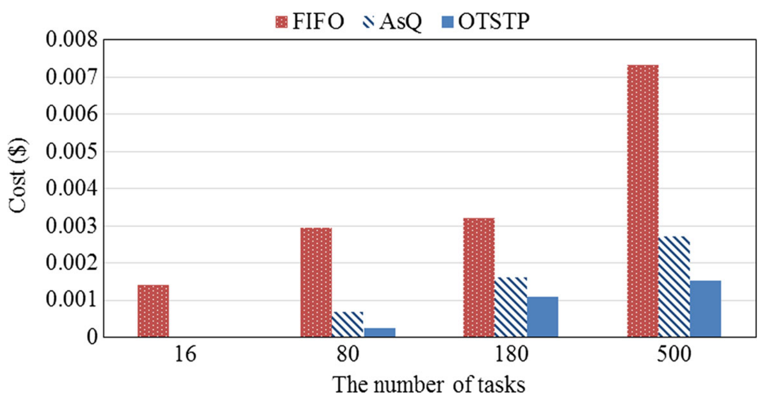
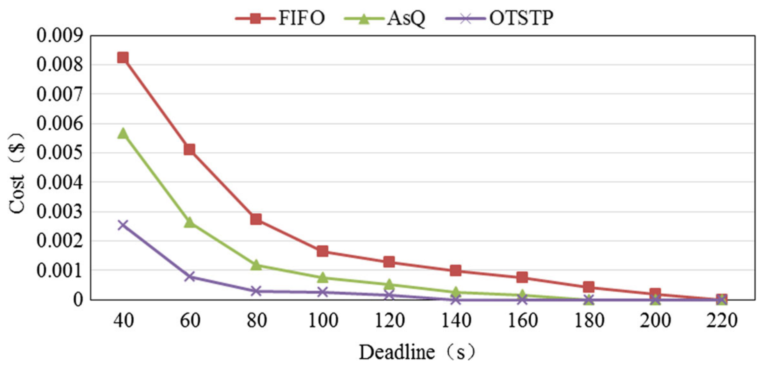


Fig. 15 Cost analysis for different deadlines



As shown in Fig. 14, when the number of job tasks is 16 and 80, the cost of the proposed *OTSTP* algorithm is 0. The reason is that the proposed algorithm can complete all tasks in the private cloud within the deadline, and does not need to apply for additional public cloud resources. When the number of tasks is 180, the proposed algorithm saves 64% and 33% cost respectively compared with FIFO and AsQ algorithm. The experimental results show that at the same job deadline, the more tasks of the same type of job, the more public cloud resources are needed, so the cost is higher. Since the tasks execution time of the proposed algorithm is smaller than FIFO and AsQ algorithm in the case of the same number of tasks, so less public cloud resources are required.

As shown in Fig. 15, when the deadline is 40, the proposed *OTSTP* algorithm saves 81% and 57% coat respectively compared with FIFO and AsQ algorithm. The jobs cannot be completed in the private cloud, so it is necessary to spend additional cost to meet the deadline constraint. As the deadline is higher, the cost of the three algorithms is reduced. When the deadline is 120s, the proposed algorithm does not need public cloud resources. When the deadline is 140s, the AsQ algorithm does not need public cloud resources. When the deadline is 220s, the FIFO algorithm does not need public cloud resources.

4.3 Experiment Summary

In this paper, the performance of the *ASQT* algorithm in private cloud is compared with FIFO scheduling algorithm, fair scheduling algorithm and *COSHH* algorithm in terms of the average job response time and throughput. Many experiments show that the *ASQT* algorithm can reduce the average job response time and improve the throughput effectively.

Compared with FIFO scheduling algorithm and *AsQ* algorithm, the performance of our proposed *OTSTP* algorithm in hybrid clouds is better in terms of the average waiting time, average execution time, average response time, QoS satisfaction rate and the monetary cost. Extensive experiments show that our proposed *OTSTP* algorithm can reduce the average task waiting time, average task execution time and average task response time significantly. It is also obvious that our algorithm can improve the QoS satisfaction rate, which guarantees the maximum utilization rate of private cloud. Moreover, the experiments illustrate that the total monetary cost of hybrid clouds is also reduced.

5 Conclusion and Future Work

In this paper, in order to improve the utilization rate of private cloud and the cost efficiency of public cloud, we have proposed efficient adaptive scheduling strategy for heterogeneous workloads in hybrid cloud. First, the queuing model is established according to the job type and heterogeneous resources, so that the job selects the optimal heterogeneous private cloud resource. Then, when the job enters the job queue of the corresponding resource pool, the task is scheduled to meet the user's constraints and apply for the optimal public cloud resource according to the requirements. Finally, a series of experiments show that our proposed efficient adaptive scheduling algorithm in hybrid clouds can improve the utilization rate and improve the throughput of private cloud. Moreover, it can also reduce the monetary cost of public cloud. In the future, we will consider the public cloud security, network load and network overhead for concurrent transfer of job data to further improve the performance of the proposed algorithm.

Acknowledgments The work was supported by the National Natural Science Foundation (NSF) under grants (No.61672397, No. 61873341), Application Foundation Frontier Project of WuHan (No. 2018010401011290), Open Foundation of Key Laboratory of Embedded System and Service Computing (Tongji University), Ministry of Education (ESSCKF 2018-2), Open Research Fund of Shaanxi Key Laboratory of Network Data Analysis and Intelligent Processing. Any opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of the above agencies.

References

1. Hwang, C.G., Yoon, C.P., Lee, D.: Exchange of data for big data in hybrid cloud environment. Int. J. Softw. Eng. Appl. **9**(4), 67–72 (2015)
2. Clementecastello, F.J., Nicolae, B., Katrinis, K., et al.: Enabling big data analytics in the hybrid cloud using iterative MapReduce. In: Proceeding of 2015 IEEE Conference on Utility and Cloud Computing. IEEE Computer Society, pp. 290–299 (2015)
3. Cisco: White paper: Cisco vni forecast and methodology (2016)
4. Guo, T., Sharma, U., Wood, T., et al.: Seagull: intelligent cloud bursting for enterprise applications. Usenix conference on technical conference. USENIX Assoc. **157**(10), 33–33 (2014)
5. Guo, T., Sharma, U., Shenoy, P., et al.: Cost-aware cloud bursting for enterprise applications. ACM Trans. Internet Technol. **13**(3), 1–24 (2014)
6. Zuo, X., Zhang, G., Tan, W.: Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud. Autom. Sci. Eng. IEEE Trans. **11**(2), 564–573 (2014)
7. Abrishami, H., Rezaeian, A., Tousi, G.K., et al.: Scheduling in hybrid cloud to maintain data privacy. In: Proceeding of 2015 International Conference on Innovative Computing Technology. IEEE, pp. 83–88 (2015)
8. Clemente-Castelló, F.J., Mayo, R., Fernández, J.C.: Cost model and analysis of iterative MapReduce applications for hybrid cloud bursting. In: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Madrid, pp. 858–864 (2017)
9. Li, C., Li, L.Y.: Hybrid cloud scheduling method for cloud bursting. Fund. Inform. **138**(4), 435–455 (2015)
10. Xue, N., Haugerud, H., Yazidi, A.: On automated cloud bursting and hybrid cloud setups using Apache Mesos. In: 2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech), Rabat, pp. 1–8 (2017)
11. Cao, Y., Lu, L., Yu, J., et al.: Online Cost-Aware service requests scheduling in hybrid clouds for cloud bursting. Web Inf. Syst. Eng. **10569**, 259–274 (2017)
12. Clemente-Castelló, F.J., Nicolae, B., Mayo, R., Fernández, J.C.: Performance Model of MapReduce Iterative Applications for Hybrid Cloud Bursting. IEEE Trans. Parallel Distrib. Syst. **29**(8), 1794–1807 (2018)
13. Wei, H., Meng, F.: A novel scheduling mechanism for hybrid cloud systems. In: International Conference on Cloud Computing, pp. 734–741. IEEE (2017)
14. Arantes, L., Friedman, R., Marin, O., et al.: Probabilistic byzantine tolerance scheduling in hybrid cloud environments. In: International Conference on Distributed Computing and Networking, pp. 2–12. ACM (2017)
15. Liu, Y., Li, C., Yang, Z., et al.: Research on cost-optimal algorithm of multi-QoS constraints for task scheduling in hybrid-cloud. J. Softw. Eng. **9**(1), 33–49 (2015)
16. Balagoni, Y., Rao, R.R.: A cost-effective SLA-aware scheduling for hybrid cloud environment. In: IEEE International Conference on Computational Intelligence and Computing Research, pp. 1–7. IEEE (2017)
17. Muñoz, V.M., Ramo, A.C., Albor, V.F., et al.: Rafhyc: an architecture for constructing resilient services on federated hybrid clouds. J. Grid Comput. **11**(4), 753–770 (2013)
18. Caballer, M., Zala, S., García, Á.L., et al.: Orchestrating complex application architectures in heterogeneous clouds. J. Grid Comput. **16**(1), 3–18 (2018)
19. Moreno-Vozmediano, R., Huedo, E., Llorente, I.M.: Implementation and provisioning of federated networks in hybrid clouds. J. Grid Comput. **15**(2), 1–20 (2017)
20. Marosi, A., Kecskemeti, G., Kertesz, A., Kacsuk, P.: FCM: an architecture for integrating IaaS cloud systems. In: Villari, M., et al. (eds.) The 2nd International Conference on Cloud Computing, GRIDs, and Virtualization, pp. 7–12 (2011)
21. Calatrava, A., Romero, E., Moltó, G., et al.: Self-managed cost-efficient virtual elastic clusters on hybrid Cloud infrastructures. Futur. Gener. Comput. Syst. **61**, 13–25 (2016)
22. Singh, D., Devgan, M., Bhushan, S.: Tasks scheduling with lessen energy usage over a cloud server using hybrid adaptive multi-queue approach. In: 2016 4th International Conference on Parallel, Distributed and Grid Computing (PDGC), Waknaghat, pp. 427–432 (2016)
23. Zuo, L., Dong, S., Shu, L., Zhu, C., Han, G.: A Multiqueue Interlacing Peak Scheduling Method Based on Tasks' Classification in Cloud Computing. IEEE Syst. J. **12**(2), 1518–1530 (2018)
24. Shorgin, S., Pechinkin, A., Samouylov, K., et al.: Queuing systems with multiple queues and b6atch arrivals for cloud computing system performance analysis. Science and Technology Conference. IEEE, pp. 1–4 (2015)
25. Singh, J., Gupta, D.: Towards energy saving with smarter multi queue job scheduling algorithm in cloud computing. J. Eng. Appl. Sci. **12**(10), 8944–8948 (2017)
26. Montes, J., Sánchez, A., Pérez, M.S.: Riding out the storm: how to deal with the complexity of grid and cloud management. J. Grid Comput. **10**(3), 349–366 (2012)

27. Pop, F., Dobre, C., Cristea, V., et al.: Deadline scheduling for aperiodic tasks in inter-Cloud environments: a new approach to resource management. J. Supercomput. **71**(5), 1754–1765 (2015)
28. Yuan, H., Bi, J., Tan, W., et al.: Temporal task scheduling with constrained service delay for profit maximization in hybrid clouds. IEEE Trans. Autom. Sci. Eng. **14**(1), 337–348 (2017)
29. Zuo, L., Shu, L., Dong, S., et al.: A multi-objective hybrid cloud resource scheduling method based on deadline and cost constraints. IEEE Access, pp. 22067–22080 (2016)
30. Zuo, X., Zhang, G., Tan, W.: Self-adaptive learning PSO-based deadline constrained task scheduling for hybrid IaaS cloud. IEEE Trans. Autom. Sci. Eng. **11**(2), 564–573 (2014)
31. Wang, Y., Xue, G., Qian, S., Li, M.: An online cost-efficient scheduler for requests with deadline constraint in hybrid clouds. In: 2017 International Conference on Progress in Informatics and Computing (PIC), Nanjing, pp. 318–322 (2017)
32. Tian, C., Zhou, H., He, Y., et al.: A dynamic MapReduce scheduler for heterogeneous workloads. In: Proceeding of 2009 International Conference on Grid and Cooperative Computing, pp. 218–224. ACM (2009)
33. Spicuglia, S., Chen, L.Y.: On load balancing: a mix-aware algorithm for heterogeneous systems. In: Proceeding of 2013 International Conference on Performance Engineering, pp. 71–76. ACM (2013)
34. Rasooli, A., Down, D.G.: COSHH: a Classification and optimization based scheduler for heterogeneous Hadoop systems. Futur. Gener. Comput. Syst. **36**, 1–15 (2014)
35. Wang, W.J., Chang, Y.S., Lo, W.T., et al.: Adaptive scheduling for parallel tasks with QoS satisfaction for hybrid cloud environments. J. Super. **66**(2), 783–811 (2013)