TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 11/18 pp344–357
DOI: 10.26599/TST.2020.9010047
Volume 27, Number 2, April 2022

Improved Heuristic Job Scheduling Method to Enhance Throughput for Big Data Analytics

Zhiyao Hu and Dongsheng Li*

Abstract: Data-parallel computing platforms, such as Hadoop and Spark, are deployed in computing clusters for big data analytics. There is a general tendency that multiple users share the same computing cluster. The schedule of multiple jobs becomes a serious challenge. Over a long period in the past, the Shortest-Job-First (SJF) method has been considered as the optimal solution to minimize the average job completion time. However, the SJF method leads to a low system throughput in the case where a small number of short jobs consume a large amount of resources. This factor prolongs the average job completion time. We propose an improved heuristic job scheduling method, called the Densest-Job-Set-First (DJSF) method. The DJSF method schedules jobs by maximizing the number of completed jobs per unit time, aiming to decrease the average Job Completion Time (JCT) and improve the system throughput. We perform extensive simulations based on Google cluster data. Compared with the SJF method, the DJSF method decreases the average JCT by 23.19% and enhances the system throughput by 42.19%. Compared with Tetris, the job packing method improves the job completion efficiency by 55.4%, so that the computing platforms complete more jobs in a short time span.

Key words: big data; job scheduling; job throughput; job completion time; job completion efficiency

1 Introduction

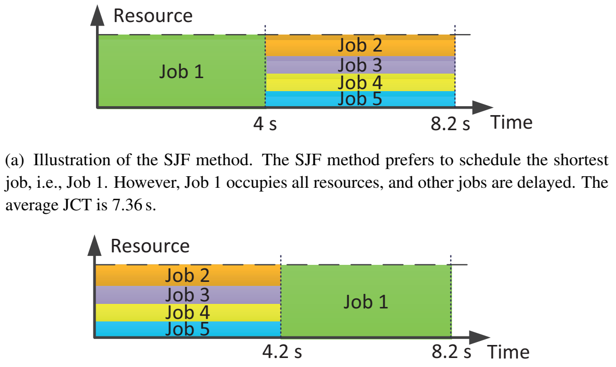
Big data analytics, such as cloud monitoring[1], is challenging for a single computer because large input data may result in an out-of-memory problem. The well-known MapReduce framework[2] aims to partition large input data, which are concurrently processed by multiple computers. Apache Spark[3] is a well-known in-memory MapReduce computing system, and is widely deployed on computing clusters. In a computing cluster, computers are interconnected via the cluster network. Users submit their big data analytics as Spark jobs. The Spark cluster has a built-in job scheduling algorithm which is used to allocate resources and determine the executing order of jobs.

Over a long period in the past, the Shortest-Job-First (SJF) method has effectively decreased the average Job Completion Time (JCT) in comparison to other scheduling techniques[4]. To decrease the JCT, the SJF method always gives high priorities to short jobs. According to experiments in recent research[5], the performance of the SJF method is the best in terms of the average JCT. However, the SJF method may lead to a suboptimal throughput of the big data computing system. The SJF method schedules jobs according to only the length of the JCT but lacks consideration of the amount of resources requested by jobs. As a result, a small number of short jobs may consume resources, incurring low job-level parallelism. This factor decreases the efficiency that jobs are completed and becomes a serious disadvantage, finally leading to a low average JCT.

Figure 1 shows an illustrative example where the SJF method is ineffective. A batch of jobs consist of five jobs:

• Zhiyao Hu and Dongsheng Li are with the College of Computer, National University of Defense Technology, Changsha 410073, China. E-mail: huzhiyao14@nudt.edu.cn; dsli@nudt.edu.cn.
∗ To whom correspondence should be addressed.

Manuscript received: 2020-09-16; revised: 2020-09-28; accepted: 2020-09-29



(a) Illustration of the SJF method. The SJF method prefers to schedule the shortest job, i.e., Job 1. However, Job 1 occupies all resources, and other jobs are delayed. The average JCT is 7.36 s.

(b) Illustration of a heuristic job scheduling method. Because Jobs 2–5 occupy less resources, they are packed together and executed concurrently. The average JCT is 5 s.

Fig. 1 Comparison of the SJF method and a heuristic job scheduling method. The heuristic job scheduling method decreases the average JCT by 32.07%.

Jobs 1–5. The JCT of Job 1, which is the shortest among all jobs, is 4 s. However, Job 1 consumes all resources. The other four jobs have longer JCT but request a small amount of resources. The JCT of the other jobs is 4.2 s. We adopt two heuristic methods, the SJF method and a heuristic job scheduling method, to schedule the batch of jobs. The job scheduling examples of the two methods are illustrated in Figs. 1a and 1b, respectively. Although the SJF method schedules the shortest job with the highest priority, the average JCT in Fig. 1a is still high. By contrast, the heuristic job scheduling method prefers to schedule other jobs. Because Jobs 2–5 request less resources, these jobs are packed together. This condition provides a desirable opportunity to increase the number of parallel jobs and decrease the average JCT. Compared with the SJF method, the heuristic job scheduling method decreases the average JCT by 32.07%.

The heuristic method performs better because of the high job completion efficiency, which we define as the number of completed jobs per unit time. The value of job completion efficiency is the rate of the number of completed jobs to the time span. In Fig. 1b, we regard Jobs 2–5 as the set of jobs. The time span of the job set is 4.2 s. The job completion efficiency of the job set is $4/4.2$, which is higher than that of Job 1. From the perspective of the throughput, the high job completion efficiency indicates that the heuristic scheduler takes a short amount of time to complete a large number of jobs. Inspired by this observation, we propose an improved heuristic job scheduling method, which schedules jobs by prioritizing the job set that has the highest job completion efficiency. Intuitively, the job completion efficiency is regarded as an analogy of a physics term density. If the set of jobs has a short time span and a large number of jobs, then the job set is dense; otherwise, the job set is sparse. Thus, the heuristic method is called the Densest-Job-Set-First (DJSF) method. Our experiments demonstrate that the DJSF method can decrease the average JCT and increase the system throughput by improving the job completion efficiency.

Before the DJSF method schedules jobs, we need to pack jobs into the set of jobs. Existing packing schedulers, such as Tetris[6], use vectors to sketch the amount of resources requested by jobs and the available amount of resources on machines. Tetris matches a job to a machine if the dot product between the resources vector of the job and that of the machine is high. However, traditional packing methods aim to maximize resources usage but do not take the completion efficiency of the job set into consideration. As a result, long and resources-expensive jobs may be packed together with short and resources-cheap jobs. This condition leads to low job completion efficiency.

In addition, the job completion efficiency of the job set determines the performance of the DJSF method. To improve the completion efficiency of the job set, we formalize the job packing problem, which involves the minimization of resources fragments and the optimization of the job completion efficiency. Nevertheless, the optimization model is computationally expensive to solve. To solve the problem, we propose a new job packing method to pack jobs in a dense manner. The job packing method enhances the job completion efficiency of packed jobs in two ways. Firstly, we employ a JCT alignment technique to align the JCT of the job set. The JCT alignment method adopts the k-means clustering algorithm to group jobs by the length of JCT. With the JCT alignment, jobs that have roughly the same JCT are placed in the same group. We analyze the JCT distribution in the Google trace[7], which shows that the majority of jobs are very close in terms of the JCT. This factor provides a desirable opportunity to align the length of the JCT. After the JCT alignment, grouped jobs are packed by the job packing method, aiming to improve the job completion efficiency. The job packing method prefers to pack jobs that request a small amount of resources into the job set. With the job packing method, the number of parallel jobs in the same job set increases. The workflow of our scheduler is shown in Fig. 2.

We design the improved heuristic job scheduling method in Fig. 2 as a pluggable module, which can be used in current resources managers, such as

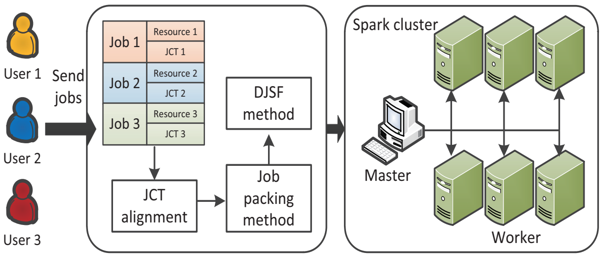


Fig. 2 Workflow of the improved heuristic job scheduling method.

Mesos[8] and YARN[9]. The heuristic module collects job information from resources managers, packs jobs in a dense manner, calculates the job completion efficiency, and determines the scheduling priority. We performed large-scale simulations based on Google cluster data. Compared with the SJF method, the DJSF method decreases the average JCT by 23.19% and enhances the system throughput by 42.19%. Compared with Tetris, the job packing method improves the job completion efficiency by 55.4%. The main contributions of this study are as follows:

- We propose an improved heuristic job scheduling method, called the DJSF method, aiming to enhance the system throughput and decrease the average JCT.
- We formalize the job packing problem, which involves the optimization of the completion efficiency of the job sets.
- We leverage the k-means clustering algorithm to make the JCT alignment and propose the job packing method to enhance the job completion efficiency.

The rest of this paper is organized as follows. In Section 2, we introduce related work about resources allocation and job scheduling methods for the Spark cluster. In Section 3, we define the concept of the job completion efficiency and propose an improved heuristic DJSF method. In Section 4, we introduce the challenge of packing jobs, formulate a job packing problem, and design the JCT clustering algorithm to make JCT alignments. In Section 5, we evaluate the DJSF method and analyze the evaluation results in Google trace-based simulations. In Sections 6 and 7, we discuss and conclude this paper, respectively.

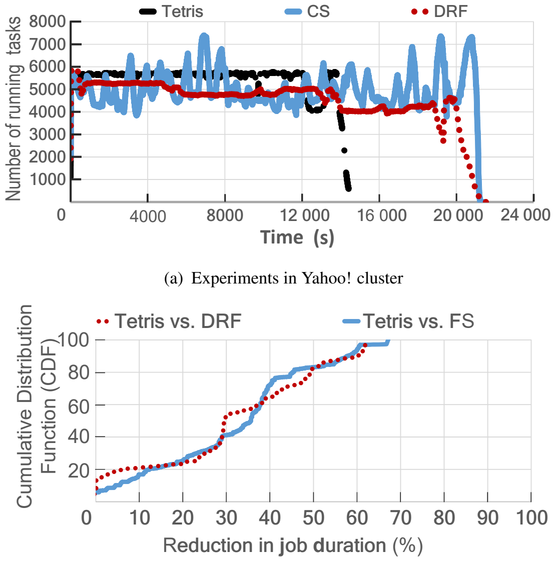
2 Background and Related Work

Resource allocation is a fundamental function in distributed systems. The JCT is heavily impacted by the amount of allocated resources. Resource managers, such as Mesos[8] and YARN[9], allocate computation resourcess. Although current scheduling policies, which are employed in Mesos and YARN, also take into account the resources usage in addition to the JCT, these scheduling policies do not achieve the optimal JCT.

Mesos employs the Dominant Resources Fairness (DRF) method by default to schedule jobs. The DRF method is used to improve the fairness[10, 11]. However, the above scheduling methods are not designed to decrease the average JCT. The Fair Scheduler (FS) and Capacity Scheduler (CS) are considered as classical scheduling methods in YARN. The FS achieves fairness between tasks by allocating averagely equal share of resources to jobs. The CS creates queues for users and the cluster capacity is shared between users. However, multiple resources demands of various jobs may incur the resources fragment problem, which degrades the performance of the FS and CS. As a result, the average JCT is prolonged.

Tetris[6] effectively addresses multiple resources packing problems and decreases the average JCT. The authors[6] compared Tetris with the FS, CS, and DRF methods, which were tested in production clusters at Facebook and Yahoo!, respectively. The results showed that Tetris performed better than the FS and CS as shown in Fig. 3. To sum up, Tetris is more effective than the scheduling methods in Mesos and YARN. For simplicity, we compare Tetris with our heuristic method in Section 5.

Tetris matches jobs to suitable machines, aiming



(a) Experiments in Yahoo! cluster

(b) Experiments in the Facebook cluster

Fig. 3 Performance of Tetris, CS, FS, and DRF methods in YARN[6].

to decrease the resources fragment. This feature is advantageous in terms of maximizing the number of simultaneously running jobs. Tetris calculates the dot product between the resources vector of jobs and the available resources vector of machines. Then Tetris prefers to schedule the job on the most suitable machine that has the highest value of the dot product.

Although Tetris decreases the JCT using a resources packing method, the average JCT is still not optimal. In a state-of-the-art research[5], the performance of the SJF method is the best in terms of the average JCT. Nevertheless, the SJF method is not optimal. The SJF method always gives high priorities to short jobs. Such a greedy scheduling policy lacks consideration of the amount of resources requested by jobs. Although jobs prioritized by the SJF method are completed in a short time, they may consume a large amount of resources and decrease the parallelism of the big data system. To address this problem, we design an improved heuristic method, called the DJSF method. The DJSF method evaluates the completion efficiency of a job set and maximizes the number of completed jobs per unit time, aiming to decrease the average JCT.

3 Improved Heuristic Job Scheduling Method Based on Job Completion Efficiency

Inspired by the observation in Fig. 1, we define the concept of the job completion efficiency and propose an improved heuristic job scheduling method, aiming to decrease the average JCT. The heuristic job scheduling method is a pluggable allocation module, which can be used in current resources managers, such as Mesos and YARN.

3.1 Concept of job completion efficiency

To decrease the average JCT, we further explore the cause why the SJF method performs worse than the heuristic method, as shown in Fig. 1. If jobs with short JCT may consume a large amount of resources, the SJF method prolongs the completion of the majority of jobs, which have a request for a small amount of resources. To sum up, the SJF method does not consider the resources capacity of the cluster. As a result, the number of parallel jobs may decrease. To address this problem, we propose an improved heuristic job scheduling method to pack as many jobs as possible into the job set and then determine the scheduling priority of the job set according to the job completion efficiency. The job completion efficiency is defined as follows.

Definition 1 (job completion efficiency). Given that a batch of jobs $j_1, j_2, \ldots, j_N$ are executed in parallel and packed into the set of jobs J. Let T denote the time span that all jobs in J are completed. The job completion efficiency is as follows:

$$\text{JCE} = \frac{N}{T} \tag{1}$$

where N is the number of jobs.

In Fig. 1, the heuristic method packs Jobs 2–5 together. We regard such packed jobs as the set of jobs. Although the heuristic method takes 4.2 s to complete jobs in the job set, the number of completed jobs is four. This achieves a high job completion efficiency (up to approximately 0.95). By contrast, the SJF method prefers to execute Job 1. This condition leads to a low job completion efficiency (i.e., 0.25). To sum up, we leverage the job completion efficiency as a priority to schedule the set of jobs. Definition 1 indicates that the value of the job completion efficiency is equivalent to the system throughput, i.e., how many jobs are completed in a given amount of time. Next, we prove that such a scheduling order, which is determined according to the job completion efficiency, achieves the smallest average JCT.

Theorem 1 Given a batch of jobs, we pack them into two sets of jobs J_1 and J_2. Note that J_1 and J_2 are executed in a successive order. If the job set with a higher job completion efficiency for scheduling, then the average JCT is smaller. Let JCE_1 and JCE_2 denote the job completion efficiency of J_1 and J_2, respectively. If $\text{JCE}_1 > \text{JCE}_2$, then the scheduling order $J_1 \rightarrow J_2$ achieves a smaller average JCT than another scheduling order $J_2 \rightarrow J_1$.

Proof We list the information about the set of jobs in Table 1. N_1 and N_2 denote the number of jobs in J_1 and J_2, respectively. T_1 and T_2 denote the time span of J_1 and J_2, respectively.

Case 1 If a scheduler prefers to schedule the job set with a higher job completion efficiency, then the scheduling preference is $J_1 \rightarrow J_2$. The completion time of J_1 is T_1. The sum of JCT of all jobs in J_1 is approximately $N_1 \times T_1$. Similarly, the completion time of J_2 is $T_1 + T_2$. The sum of JCT in J_2 is approximately

Table 1 Example for the scheduling of two job sets.

Job set	Number of jobs	Time span
J_1	N_1	T_1
J_2	N_2	T_2

$N_2 \times (T_1 + T_2)$. In this case, the sum of JCT in J_1 and J_2 is

$$N_1 \times T_1 + N_2 \times (T_1 + T_2) \tag{2}$$

Case 2 If a scheduler prefers to schedule the job set with a smaller job completion efficiency, then the scheduling preference is $J_2 \rightarrow J_1$. The completion time of J_2 is T_2. The sum of JCT of all jobs in J_2 is approximately $N_2 \times T_2$. Similarly, the completion time of J_1 is $T_1 + T_2$. The sum of JCT in J_1 is approximately $N_1 \times (T_1 + T_2)$. In this case, the sum of JCT in J_1 and J_2 is

$$N_2 \times T_2 + N_1 \times (T_1 + T_2) \tag{3}$$

Because J_1 has a larger job completion efficiency than J_2, we have $N_1/T_1 > N_2/T_2$. Hence, we have $N_1 \times T_2 > N_2 \times T_1$. We compare Formula (2) with Formula (3). The sum of JCT in Case 1 is smaller than that in Case 2. We make a conclusion that for any two job sets J_1 and J_2, the job set with a larger job completion efficiency should be preferred for scheduling to achieve a smaller average JCT. ■

We extend the above conclusion to a new case where three job sets are scheduled. To extend the conclusion, we introduce the following lemma.

Lemma 1 Given that any two job sets J_a and J_b can be executed serially, the two job sets are scheduled in a successive order. We regard J_a and J_b as a new job set, denoted by J_c. Let $\mathrm{JCE_a}$ and $\mathrm{JCE_b}$ denote the job completion efficiency of J_a and J_b, respectively. We have

$$\min\{\mathrm{JCE_a}, \mathrm{JCE_b}\} < \mathrm{JCE_c} < \max\{\mathrm{JCE_a}, \mathrm{JCE_b}\}.$$

Proof Let N_a and N_b denote the number of jobs in J_a and J_b, respectively. Let T_a and T_b denote the time span of J_a and J_b, respectively. We have $\mathrm{JCE_a} = N_\mathrm{a}/T_\mathrm{a}$ and $\mathrm{JCE_b} = N_\mathrm{b}/T_\mathrm{b}$. For the job set J_c, the number of jobs in J_c is $N_\mathrm{a} + N_\mathrm{b}$. The time span of J_c is $T_\mathrm{a} + T_\mathrm{b}$. Thus, the job completion efficiency of J_c is $\mathrm{JCE_c} = (N_\mathrm{a} + N_\mathrm{b})/(T_\mathrm{a} + T_\mathrm{b})$. Assume that the job completion efficiency of J_a is larger than that of J_b. Owing to $\mathrm{JCE_a} > \mathrm{JCE_b}$, i.e., $N_\mathrm{a}/T_\mathrm{a} > N_\mathrm{b}/T_\mathrm{b}$, we have $N_\mathrm{a} \times T_\mathrm{b} > N_\mathrm{b} \times T_\mathrm{a}$. Hence, we have $(N_\mathrm{a} + N_\mathrm{b}) \times T_\mathrm{b} > N_\mathrm{b} \times (T_\mathrm{a} + T_\mathrm{b})$. The following formula holds:

$$(N_\mathrm{a} + N_\mathrm{b})/(T_\mathrm{a} + T_\mathrm{b}) > N_\mathrm{b}/T_\mathrm{b} \tag{4}$$

Hence, $\mathrm{JCE_c} > \mathrm{JCE_b}$ is proven.

Similarly, because $N_\mathrm{a} \times T_\mathrm{b} > N_\mathrm{b} \times T_\mathrm{a}$, we have $N_\mathrm{a} \times (T_\mathrm{a} + T_\mathrm{b}) > (N_\mathrm{a} + N_\mathrm{b}) \times T_\mathrm{a}$,

$$N_\mathrm{a}/T_\mathrm{a} > (N_\mathrm{a} + N_\mathrm{b})/(T_\mathrm{a} + T_\mathrm{b}) \tag{5}$$

Hence, $\mathrm{JCE_a} > \mathrm{JCE_c}$ is proven. ■

Next, we expand Theorem 1 to the case where over three job sets, i.e., J_1, J_2, and J_3, are scheduled. We sort them by the value of the job completion efficiency. Assume that $\mathrm{JCE}_1 > \mathrm{JCE}_2 > \mathrm{JCE}_3$. According to Lemma 1, we regard J_2 and J_3 as a new set of jobs. Let $J_2{}'$ denote the new job set. JCE_2' denotes the job completion efficiency of $J_2{}'$. According to Lemma 1, we have $\mathrm{JCE}_2 > \mathrm{JCE}_2' > \mathrm{JCE}_3$. Then the scheduling problem becomes the case in Theorem 1, where two job sets, J_1 and $J_2{}'$, are scheduled. According to Theorem 1, we prefer to schedule J_1 because the job completion efficiency of J_1 is larger than those of the other job sets. Similarly, after scheduling J_1, we prefer to schedule J_2 rather than J_3. The final scheduling preference is $J_1 \rightarrow J_2 \rightarrow J_3$, which achieves the smallest average JCT.

To sum up, Theorem 1 can be recursively applied in multiple-job-set cases. We make the following general theorem.

Theorem 2 Given multiple job sets, the job set with the largest job completion efficiency should be scheduled with the highest priority to decrease the average JCT.

3.2 DJSF method

According to Theorem 2, we design an improved heuristic job scheduling method called the DJSF method. As the job completion efficiency is the rate of the number of jobs to the length of the time span, we intuitively regard the job completion efficiency as a physics term density. If the set of jobs has a short time span and a large number of jobs, then the set of jobs is dense. Otherwise, the set of jobs is sparse if it has a few jobs and a long time span. Algorithm 1 shows the core idea of the DJSF method. The input of the DJSF method is a batch of job sets. The DJSF method calculates the job completion efficiency of each job set. The DJSF method prefers to allocate resources to the job set with

Algorithm 1 DJSF method

```
Input: The set of jobs $j_1, j_2, \ldots, j_N$.
Input: The available resources capacity $C$.
for the $m$-th job set, $m = 1, \ldots, M$ do
    Calculate the job completion efficiency $\mathrm{JCE}_m$.
Sort all job sets in decreasing order of their job completion
  efficiency.
for the $m$-th job set, $m = 1, \ldots, M$ do
    if remaining resources capacity $C$ is sufficient then
        Allocate resources to $J_m$.
        Update the amount of remaining resources.
```

the highest job completion efficiency. The complexity of the DJSF method is approximately $O(M)$, where M is the number of job sets. Such a complexity is smaller than that of the SJF method, which is approximately $O(N)$, where N is the number of jobs.

4 Job Packing Method

We clarify that the DJSF method schedules a batch of job sets. Jobs should be packed into the job sets first before the DJSF method works. The completion efficiency of the job set is determined by how jobs are packed into the job set. Thus, the job packing method finally impacts the average JCT. However, existing job packing schemes focus on decreasing resources fragments but incur a low job completion efficiency. In this section, we design a job packing method to pack jobs into the set of jobs in a way that the job completion efficiency is maximized.

4.1 Challenge

Tetris is the state-of-the-art packing method[6], aiming to decrease resources fragments by evaluating the packing score. For scheduled jobs, Tetris profiles the resources request for the Central Processing Unit (CPU), Random Access Memory (RAM), and disk. A resources vector v_i is used to represent the resources request of the i-th job. To assign the job to the best suitable machine, Tetris measures the remaining amount of resources on all machines. A vector h_j is used to represent the remaining resources on the j-th machine. The dot product between v_i and h_j is the packing score, indicating whether the i-th job matches the j-th machine. Tetris calculates all packing scores, and the job with the highest score is preferred to schedule. However, Tetris does not take the JCT into consideration, and as a result, the set of jobs that are packed into the same machine may suffer from a low job completion efficiency.

In Fig. 4, there are five jobs that are scheduled on a machine. The available resources of the machine is $\langle 0.2, 0.2, 0.2\rangle$. The scores of Tetris packing five jobs to one machine are listed in Table 2. Tetris calculates the packing score using the dot product. We can see that Job 3 has the highest score. Therefore, Tetris packs the job to the machine. However, the packing scheme incurs a low job efficiency, which is as low as 0.2. As a comparison, we design a heuristic job packing method. The heuristic method prefers to pack as many jobs as possible. Jobs 1, 2, 4, and 5 are packed together. In addition, the time span of jobs that are packed together is as short as 4 s. Thus, the heuristic method achieves a high job completion efficiency. To sum up, current packing methods, such as Tetris, do not take the job completion efficiency into consideration. This incurs a low job completion efficiency and further impacts the effectiveness of the DJSF method.

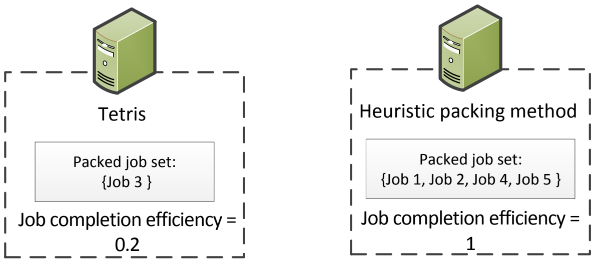

Tetris
Packed job set: {Job 3 }
Job completion efficiency = 0.2
Heuristic packing method
Packed job set: {Job 1, Job 2, Job 4, Job 5 }
Job completion efficiency = 1

Fig. 4 Illustrative example of low job completion efficiency that is incurred by Tetris. In the example, the heuristic job packing method can achieve a high job completion efficiency.

Table 2 Scores of Tetris packing five jobs to one machine.

Job number	⟨CPU, RAM, disk⟩	JCT (s)	Packing score
1	⟨0.2, 0.1, 0.1⟩	4	0.08
2	⟨0.2, 0.3, 0.1⟩	4	0.12
3	⟨0.8, 0.7, 0.6⟩	5	0.42
4	⟨0.2, 0.2, 0.1⟩	4	0.10
5	⟨0.2, 0.1, 0.1⟩	4	0.08

4.2 Problem formulation

We further formalize the job packing problem, which involves the optimization of the job completion efficiency of the job sets. Given a batch of jobs $\{j_1, \ldots, j_N\}$, there are N jobs in total. M denotes the number of the job sets. Let N_m denote the number of jobs in the m-th job set. Let β_n^m denote a binary variable, indicating whether the n-th job is packed in the m-th set of jobs. Let JCT_n and R_n denote the JCT of the n-th job and the amount of resources requested by the n-th job, respectively. The objective function is as follows:

$$\max \sum_{m=1}^{M} \mathrm{JCE}_m \tag{6}$$

where JCE_m denotes the job completion efficiency of the m-th job set. The optimization goal is subject to the following constraints:

$$\mathrm{JCE}_m = N_m / T_m \tag{7}$$

$$N_m = \sum_{n=1}^{N} \beta_n^m \tag{8}$$

$$T_m = \max_{n=1}^{N} \mathrm{JCT}_n \times \beta_n^m \tag{9}$$

$$R_m = \sum_{n=1}^{N} R_n \times \beta_n^m \tag{10}$$

$$R_m \leqslant C, \forall m \tag{11}$$

Constraint (7) defines the calculation of the job completion efficiency of the m-th job set. Constraint (8) calculates the number of jobs in the m-th job set. If the n-th job belongs to the m-th job set, then the binary variable β_n^m is 1. Otherwise, β_n^m is 0. Constraint (9) calculates the time span of the m-th job set, i.e., the longest JCT of jobs in the m-th job set. Constraint (10) calculates the sum of the amount of resources requested by all jobs in the m-th job set. Constraint (11) requires the sum of the amount of resources in the cluster. To sum up, the optimization model is computationally expensive to solve because it is an integer nonlinear programming problem.

4.3 Increasing job completion efficiency through JCT alignment

To increase the job completion efficiency of a job set, we pack jobs in a dense manner. We propose the JCT alignment, aiming to pack jobs that have roughly the same JCT into the job set. This method effectively decreases the average time span.

For example, there are six jobs, i.e., Jobs 1–6, that need to be packed. Assume that the amount of resources requested by each job is 0.5. That is, only two jobs can be packed together. Nevertheless, their JCT are different. Assume that the JCT of Jobs 1–6 are 1, 2, 5, 6, 9, and 10 s, respectively.

In the example, the JCT alignment method packs Jobs 1 and 2 together, Jobs 3 and 4 together, and Jobs 5 and 6 together. This generates three job sets. The corresponding time span of the job set is 2, 6, and 10 s, respectively. The total job completion efficiency is approximately 1.53. By contrast, the average job completion efficiency is low if the JCT in different job sets is not aligned. If we pack Jobs 1 and 6, Jobs 2 and 5, and Jobs 3 and 4 together, the total job completion efficiency is approximately 0.75. To sum up, the JCT alignment enhances the average job completion efficiency.

We leverage the cumulative distribution curve of JCT to make a JCT alignment. Figure 5 shows the illustrative example of the cumulative distribution curve of JCT. All jobs are sorted by the length of JCT. Jobs that have the same JCT are represented as a dot on the curve. To make a JCT alignment, the jobs that are at the same point on the curve should be packed together.

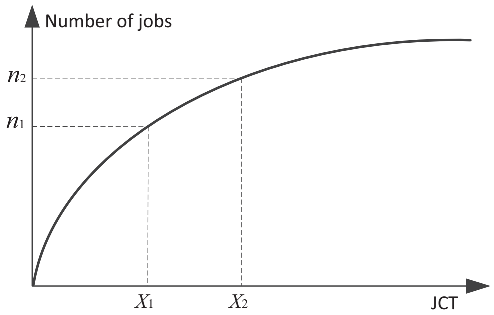


Fig. 5 Illustrative example: Cumulative distribution curve of JCT.

Owing to Constraint (11), the sum of resources requested by jobs in the job set should be smaller than the capacity of resources in the cluster. In most cases, the number of jobs that are at the same point on the curve is small. Thus, we may pack jobs that have multiple successive points on the curve. Concretely, we employ machine learning techniques, such as the clustering algorithm, to group jobs according to the length of the JCT. In a group, the JCT is approximately aligned.

We analyze the distribution of JCT in the Google trace. In the production environment, the real JCT of the majority of jobs ranges from 25 s to 625 s, as shown in Fig. 6. Figure 6 shows that the number of jobs that have roughly the same JCT is large. This feature provides a desirable opportunity to make the JCT alignment.

4.4 *k*-means clustering-based job packing method

The job packing method adopts the k-means clustering algorithm to group a batch of JCT according to the length of JCT. The k-means clustering algorithm is a well-known data mining and machine learning tool, which is used to assign data points (i.e., the JCT of multiple jobs) into groups without any prior knowledge of those relationships. The k-means clustering algorithm attempts to learn the difference between data points and

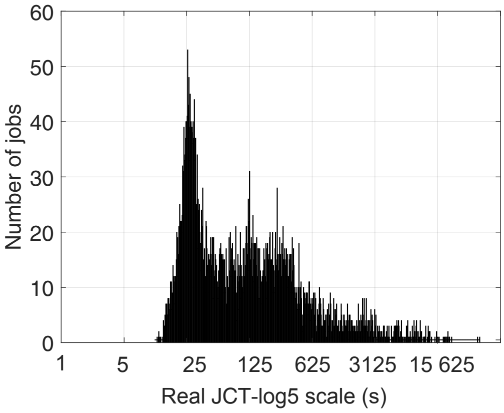


Fig. 6 Number of jobs changes with real JCT.

finds which group, a data point belongs to. The k-means clustering algorithm is one of the simplest clustering techniques, which is widely used in biometrics, medical imaging, and related fields. The k-means clustering algorithm determines the group where data points should be assigned into rather than the user instructing to build the group of data points based on heuristic rules.

We are motivated to leverage the k-means clustering algorithm[12] to group jobs with different JCT. Algorithm 2 shows the main steps of the JCT clustering algorithm. The k-means clustering based job packing method places a job to the group of jobs if the sum of the difference between the centroid of the group and the JCT of the jobs is at a minimum. A less variation in the group results in the jobs within the group to have similar JCT. The k-means based JCT clustering algorithm is known to have a time complexity of $O(N^2)$[13].

In addition to the JCT alignment, the job packing method should increase the number of jobs packed in the job set. Given a group of jobs, the job packing method prefers to pack jobs that request a small amount of resources. This method aims to enhance the number of parallel jobs in the job set. In Fig. 5, n_1 is the number of jobs, of which the JCT is smaller than X_1. Similarly, n_2 is the number of jobs, of which the JCT is smaller than X_2. Assume that we pack the jobs, of which the JCT is larger than X_1 but smaller than X_2, into a new job set. The number of jobs in the job set is $n_2 - n_1$. The time span of the job set is X_2. Thus, the job completion efficiency of the job set is $(n_2 - n_1)/X_2$. To enhance the value of the job completion efficiency, the job packing method maximizes the value of $n_2 - n_1$.

Algorithm 2 k-means-based JCT clustering method

```
Input: A batch of jobs $J = \{j_1, j_2, \ldots, j_N\}$, the resources
    $\{R_1, R_2, \ldots, R_N\}$, the JCT $\{\text{JCT}_1, \text{JCT}_2, \ldots, \text{JCT}_N\}$, and
    the available resources capacity $C$.
Estimate the number of groups $G = \sum_{n=1}^{N} R_n / C$.
Initialize the centroid for each group.
Clustering($J$, $G$).
for the $i$-th group, $i = 1, \ldots, G$ do
    Take all jobs in the $i$-th group.
    Compute the average value of their JCT $\overline{\text{JCT}}$.
    if $W_i \neq \overline{\text{JCT}}$ then
        Update the centroid, $W_i' = \overline{\text{JCT}}$.
        Clustering($J$, $G$).
    else if Centroids of all groups are not updated then
        Stop the algorithm.
return The group of jobs.

function Clustering($J$, $G$)
for the $n$-th job, $n = 1, \ldots, N$ do
    for the $i$-th group, $i = 1, \ldots, G$ do
        Get $W_i$, the centroid of the $i$-th group.
        Calculate the difference $D_i = |W_i - \text{JCT}_n|$.
    Calculate $i' = \arg\min_{i=1}^{G} D_i$.
    /*Find the group with a minimal difference */
    Assign the $n$-th job to the $i'$-th group.
```

Algorithm 3 shows the main steps of the job packing method. After the JCT clustering method outputs the group of jobs, the job packing method packs jobs into the job set. The job packing method calculates the average JCT of each group. Then the job group that has smaller average JCT is preferred to be used in Algorithm 3. Within each group, the job packing method sorts jobs by the amount of resources. Jobs that request a small amount of resources are packed into the job set with high priorities. If the remaining resources is not enough for packing a job into the job set, then the job is packed into the next job set. When the last job is packed, the job packing method stops. Algorithm 3 has two For loops. The complexity of the job packing method is $O(GN)$, where G is the number of groups.

Algorithm 3 Job packing method

```
Input: A batch of jobs $\{j_1, j_2, \ldots, j_N\}$, the
    amount of resources $\{R_1, R_2, \ldots, R_N\}$, the JCT
    $\{\text{JCT}_1, \text{JCT}_2, \ldots, \text{JCT}_N\}$, and the available resources
    capacity $C$.
Record remaining amount of resources, $C_{\text{remain}} = C$.
Record the number of job sets $k = 1$.
Calculate the job group $G$.
Call $k$-means based JCT clustering method.
Calculate the average JCT of jobs in each group.
Sort $G$ in increasing order of their average JCT.
for the $i$-th group, $i = 1, \ldots, G$ do
    Get $G_i$, i.e., the number of jobs in the $i$-th group.
    Get $\{j_1^i, \ldots, j_{G_i}^i\}$, i.e., all jobs in the $i$-th group.
    Sort jobs in increasing order of their resources.
    for the $n$-th job, $n = 1, \ldots, G_i$ do
        Get $R_n^i$, i.e., resources that are requested by $J_n^i$.
        if $R_n^i \leqslant C_{\text{remain}}$ then
            Get $k$, the number of job sets.
            Pack $J_n^i$ into the $k$-th job set.
            Update $C_{\text{remain}} = C_{\text{remain}} - R_n^i$.
        else
            Pack $J_n^i$ into the $(k+1)$-th job set.
            $k = k + 1$.
            Update $C_{\text{remain}} = C - R_n^i$.
```

5 Evaluation

We evaluate the performance of our method, Tetris, and the SJF method. We perform simulations in a computer equipped with an Intel(R) Core(TM) i7-4700MQ CPU 2.40 GHz and 8 GB RAM. Our workloads are taken from the Google trace that is widely accepted as a data-parallel benchmark[14–16].

5.1 Simulation settings

Workload: Google cluster data provide a detailed task duration, resources requirements, and machine constraints. We extract 1000 jobs from the Google trace. The 1000 jobs are submitted in one batch. Note that it is very slow to simulate the execution of 1000 jobs because the real durations in the Google trace are long. We set the simulated program to run 10 times faster than the trace. The JCT distribution of 1000 jobs is shown in Fig. 7. The average JCT is 685.39 s.

Methods comparison: We compare our methods with two scheduling methods, Tetris and the SJF method. Tetris is a multi-resources packing method to enhance resources utilization. Then we evaluate the SJF method. The SJF method prefers to schedule the job with the least JCT. According to past research[6], Tetris performed better than default scheduling algorithms in YARN and Mesos, including the FS, CS, and DRF methods, as shown in Fig. 3. The result shows that Tetris performs better than the FS, CS, and DRF methods, and thus, we compare Tetris with our heuristic method for simplicity.

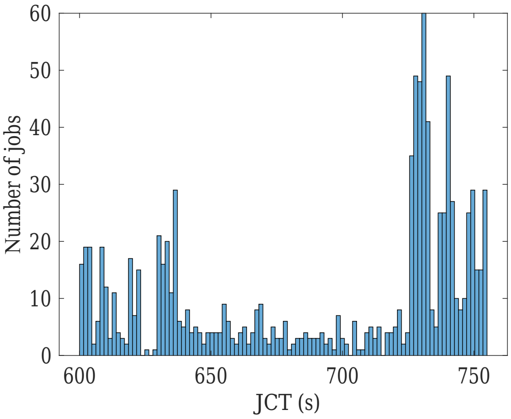


Fig. 7 JCT distribution of 1000 jobs that are used in simulation.

5.2 Simulation result

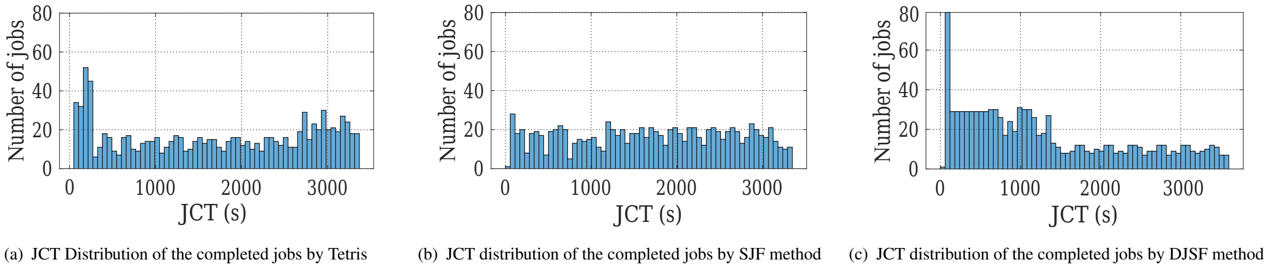
We measure the completion time of 1000 jobs and show the JCT distribution of Tetris, SJF method, and DJSF method. In addition, we record the changing cluster state, including the system throughput, CPU utilization, memory, and disk space. We evaluate the job completion efficiency of Tetris and our job packing method. Then we evaluate the algorithm overhead incurred by Tetris, SJF method, and DJSF method.

JCT distribution of completed jobs: We record the JCT of 1000 jobs, which are completed by Tetris, SJF method, and DJSF method. The JCT distribution of Tetris is shown in Fig. 8a. Tetris completes a different number of jobs when the time increases from 0 to over 3000 s. In the beginning, Tetris completes jobs at a high speed, which lasts for a short time and then slows down halfway. As a comparison, the SJF method completes jobs at a stable but low speed, as shown in Fig. 8b. The average JCT of Tetris and SJF method is 1696.3 and 1695.4 s, respectively. The DJSF method achieves the highest job completion efficiency.

In Fig. 8c, large considerable jobs are completed at the first 1000 s. We infer that the DJSF method prefers to schedule jobs in the job set that has a high job completion efficiency. After the first 1000 s, the DJSF method completes jobs at a low speed. We infer that the DJSF method schedules the job sets, which consist of long and resources-expensive jobs. These job sets have low job completion efficiency. Nevertheless, the average JCT of the DJSF method is 1302.2 s, which is smaller than that of Tetris and SJF method.

JCT reduction: We define the JCT reduction of the DJSF method against other compared methods as follows:

$$\text{IR} = \frac{\text{JCT}_{\text{Compared methods}} - \text{JCT}_{\text{DJSF}}}{\text{JCT}_{\text{Compared methods}}} \tag{12}$$



(a) JCT Distribution of the completed jobs by Tetris (b) JCT distribution of the completed jobs by SJF method (c) JCT distribution of the completed jobs by DJSF method

Fig. 8 JCT distribution of 1000 jobs that are completed by Tetris, SJF method, and DJSF method. The majority of jobs that are completed by the DJSF method have shorter JCT.

where JCT_{DJSF} and $\text{JCT}_{\text{Compared methods}}$ denote the JCT of a test job that is scheduled by the DJSF method and other compared methods, respectively. Figure 9 shows the JCT reduction of the DJSF method against Tetris and the SJF method, respectively. The minority of 1000 jobs suffer from the negative value of the JCT reduction. This finding indicates that the DJSF method takes a longer time to complete these jobs than Tetris and the SJF method. Nevertheless, the DJSF method achieves a good job reduction for the majority of jobs. The highest JCT reduction of the DJSF method against other compared methods is up to 80%.

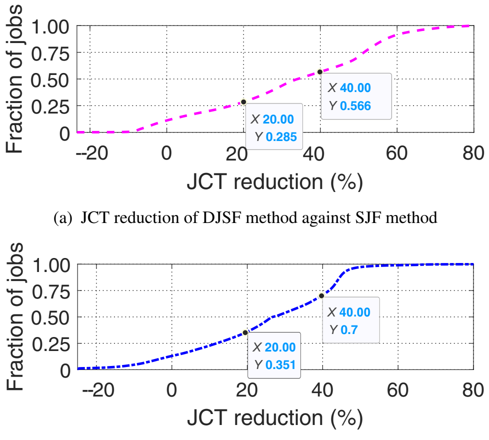
System throughput: We evaluate the system throughput of the simulated Spark cluster. The system throughput P is calculated as follows:

$$P = \frac{N^*}{T^*} \tag{13}$$

where T^* denotes the cumulative working time in the simulated Spark cluster and N^* denotes the cumulative number of jobs that are completed in a simulated Spark cluster. Of note, Eq. (13) is similar to the job completion efficiency in Eq. (1).

Figure 10 shows the system throughput when 1000 jobs are scheduled by Tetris, SJF method, and DJSF method, respectively. The system throughput of the SJF method is the least and hardly changes from the beginning to the end. Tetris achieves a higher system throughput than the SJF method at the first 1000 s. This finding indicates that Tetris can complete more jobs than the SJF method at the first 1000 s. The DJSF method achieves the highest system throughput at the first 3000 s.

Job completion efficiency of the job set: We compare the k-means job packing method with Tetris in terms of the job completion efficiency. To make the DJSF method more effective, we propose the k-means job packing method to pack jobs in a dense manner so that the job completion efficiency of the job set is improved. Figure 11 shows the job completion efficiency of all job sets.



(a) JCT reduction of DJSF method against SJF method

(b) JCT reduction of DJSF method against Tetris

Fig. 9 JCT reduction of DJSF method against SJF method and Tetris.

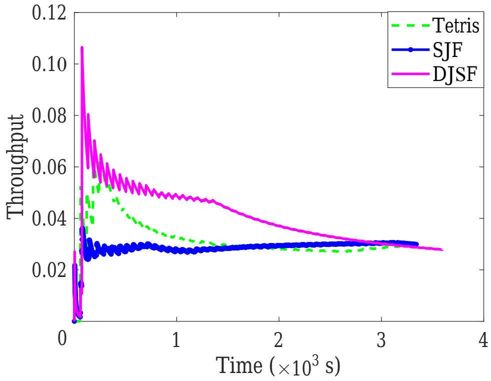


Fig. 10 System throughput when the scheduling time increases.

Tetris packs all jobs into 125 job sets, whereas our packing method packs all jobs into 127 job sets. The job completion efficiency of Tetris changes from 0.023 to 0.058. This is because Tetris does not consider the job completion efficiency of job sets. In the job set packed by Tetris, although the resources packing score is high, the job completion efficiency is low. As a comparison, the k-means job packing method maximizes the number of jobs that are packed together. Thus, the highest job completion efficiency is up to approximately 0.12. Nevertheless, the job completion efficiency of individual job set is low and approximately 0.04. Thus, long and resources-expensive jobs are packed in these job sets.

Resource utilization: We measure the resources usage when the jobs are scheduled by Tetris, SJF method, and DJSF method. The bottleneck resource is memory.

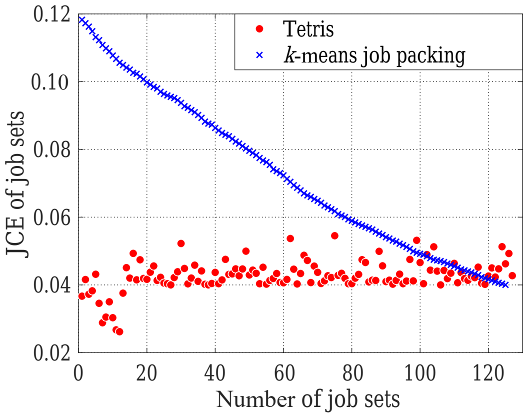
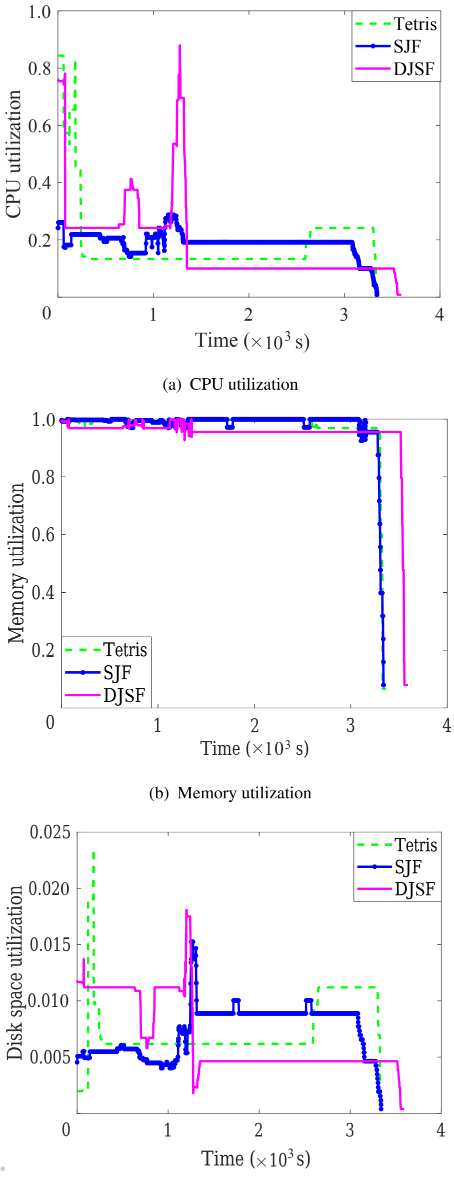


Fig. 11 Job completion efficiency of different job sets. The average job completion efficiency of Tetris and k-means job packing method is 0.0426 and 0.0662, respectively.

Owing to the limited amount of memory, no more jobs can be executed concurrently. The amount of remaining CPU and disk space resources is sufficient in Figs. 12a and 12c. However, the memory resources are fully used in Fig. 12b.

Compared with Tetris and the SJF method, the DJSF method leads to a higher CPU and disk space resource consumption at the first 1000 s. Then the CPU and disk space resource consumption decreases at the remaining time. The DJSF method completes more jobs at the first 1000 s and thus, its resource consumption is higher.

To sum up, the DJSF method makes full use of available resources to increase the number of completed jobs at an early scheduling time. We regard this as an important cause that the DJSF method achieves the least average JCT.



(a) CPU utilization

(b) Memory utilization

(c) Disk space utilization

Fig. 12 Normalized resources usage of the CPU, memory, and disk space when 1000 jobs are scheduled by Tetris, SJF method, and DJSF method, respectively.

Algorithm overhead: We evaluate the algorithm overhead incurred by Tetris, SJF method, and our method. Our scheduling method incurs two main algorithm overheads that are incurred by the job packing method and the DJSF method. Table 3 shows the algorithm overhead incurred by Tetris, SJF method, and our method. The algorithm overhead incurred by the heuristic job scheduling method is far smaller than that by the other methods.

The SJF method incurs the least algorithm overhead because the SJF method sorts the JCT only once. When free resources are available, the SJF method does not sort the JCT anymore. As a comparison, Tetris sorts the jobs by the amount of remaining resources many times because when a job is scheduled, the amount of remaining resources changes. Tetris has to recalculate packing scores and resort jobs each time a job is scheduled. In our method, the job packing method is performed only once. However, the k-means clustering algorithm is executed recursively and thus, incurs large algorithm overheads. Moreover, the algorithm overhead incurred by the DJSF method is close to that incurred by the SJF method.

6 Discussion

6.1 Impact of prediction error

The SJF method is a non-preemptive method in which a waiting job with the smallest estimated run-time-to-completion is prioritized. The SJF method is dependent on the precise knowledge of how long a job will run, and this information is not usually available. Our job packing method also relies on a priori knowledge of JCT and the amount of resources requested by jobs. To address the problem, several researchers have focused on the study of performance prediction for data-parallel jobs in big data systems, such as Spark and Hadoop[17].

Ernest ran the entire job on small datasets and tried to capture how the JCT of a job changed with the increasing size of the input dataset[18]. The model of

Table 3 Algorithm overhead.

Scheduler	Algorithm overhead (s)
Tetris	0.555 357
SJF	0.198 722
Our method	0.968 528

Ernest was tightly bound to the application type, which limited the cross-application use of Ernest models. Bei et al.[19] proposed to construct two ensembles of performance models using a random-forest approach for the map and reduce stages, respectively. However, it is difficult to model the duration of each operation. Yu et al.[20] proposed a Hierarchical Model (HM), aiming to combine a number of individual sub-models in a hierarchical manner. The HM was agnostic to the execution of various operations. Hu et al.[21] designed a deep learning based prediction model, which employed a graph convolutional network to learn the execution of Spark jobs. To sum up, the above prediction models can be seamlessly applied to predict the JCT and combined with our job scheduling method. In the context of the paper, we regard job information, such as JCT, as a priori knowledge.

We added new simulations to evaluate the performance of the DJSF method. In the experiment setting, each job has the real JCT (JCT_r) and the predicted JCT (JCT_p). The real JCT is extracted from the dataset. Given the prediction error ε, the predicted JCT is calculated as follows:

$$\text{JCT}_\text{p} = \text{JCT}_\text{r} \times (1 - \varepsilon) \tag{14}$$

The prediction accuracy of the state-of-the-art prediction model is above 0.8. Thus, we set the value of ε in a range from 0.0001 to 0.2. We evaluate the average JCT of different methods. The results are listed in Table 4.

When the prediction error increases, the performance improvement of the DJSF method over the SJF method increases. We infer that the JCT alignment method alleviates the negative influence incurred by the prediction error. Before the DJSF method schedules jobs, the JCT alignment method groups jobs by the length of JCT. If the prediction error is small, then the jobs that have roughly the same JCT are still grouped together. When the DJSF method schedules these jobs, the prediction error slightly changes the scheduling order of jobs.

As a comparison, the SJF method is more sensitive to the prediction error because the SJF method determines the job scheduling order according to only the length of the JCT. The prediction error changes the job scheduling order in a straightforward way. The results show that the prediction error degrades the performance of the SJF method. The Tetris method is a job packing method, and thus, Tetris is not impacted by the prediction error of JCTs.

Besides, several works involve the scheduling problem. Reducing the energy consumption of the storage systems disk read/write requests plays an important role in improving the overall energy efficiency of high-performance computing systems[22]. Static placement and dynamic management are two types of Virtual Machine (VM) management methods[23]. VirtCo is proposed to achieve joint coflow scheduling and virtual machine placement in cloud data centers[24]. The authors investigated the scalability of the request scheduling process in cloud computing and provided a theoretical definition of the scalability of this process[25]. Virtual machine fault-tolerant placement for data centers of cloud systems also involves the scheduling problem.

6.2 Impact of number of job groups

One of the main challenges in the k-means clustering algorithm is to specify the number of groups as an input parameter. Algorithm 2 is not capable of determining the appropriate number of groups. For the job packing case, we should make the set of jobs compact so that jobs are completed rapidly and resources are fully utilized. Based on this idea, we roughly estimate the number of groups as follows. We calculate the sum of resources requested by all jobs. Let $\sum\limits_{n=1}^{N} R_n$ denote the sum of resources. Assume that the number of groups, denoted by G, is equal to the sum of resources divided by the capacity of resources in the cluster. We have

$$G = \sum_{n=1}^{N} R_n / C \tag{15}$$

We conduct an experiment on the varying values of the number of job groups in Algorithm 2. The results are listed in Table 5. When the number of job groups increases from 1 to 10, the performance improvement of

Table 4 Impact of prediction errors.

Prediction error	JCT of Tetris (s)	JCT of SJF (s)	JCT of DJSF (s)	Improvement rate of DJSF over SJF	Improvement rate of DJSF over Tetris
0	1696.3	1695.3	1302.2	0.232	0.232
0.0001–0.001	1696.3	1698.0	1302.2	0.233	0.232
0.001–0.01	1696.3	1793.4	1301.6	0.274	0.233
0.01–0.2	1696.3	1787.2	1299.3	0.273	0.234

Table 5 Impact of number of job groups on improvement rate.

Number of groups	Improvement rate of DJSF over SJF	Improvement rate of DJSF over Tetris	Number of groups	Improvement rate of DJSF over SJF	Improvement rate of DJSF over Tetris
1	0.2530	0.232	6	0.232	0.209
2	0.2560	0.235	7	0.231	0.210
3	0.2590	0.239	8	0.238	0.217
4	0.2490	0.229	9	0.228	0.207
5	0.2390	0.218	10	0.221	0.199

the DJSF method over other methods increases at first and then decreases. We infer that the large number of job groups decreases the performance of the job packing method. In previous experiments, we set the number of job groups to 6 according to the characteristic of the test workload. Nevertheless, the optimal setting of the number of job groups is 3. We infer that if the k-means clustering method generates too many groups, then the number of jobs in each job group would decrease. This may be disadvantageous for the job packing method to pick up and pack suitable jobs from a job group. To sum up, determining the optimal number of job groups is still an open problem. We suggest to set a smaller number of job groups than that in the heuristic setting in Eq. (15).

7 Conclusion

In this paper, we propose three algorithms, the DJSF method, job packing method, and JCT clustering method, which are jointly used to schedule multiple jobs. The DJSF method aims to decrease the average JCT and overcome the disadvantage of the SJF method. We define the concept of the job completion efficiency. The job packing method aims to pack jobs into a set of jobs, which has a small amount of resources fragment and a large job completion efficiency. The JCT clustering method adopts a machine learning technique, k-means clustering algorithm, to make a JCT alignment. To sum up, the DJSF method achieves small average JCT and enhances the system throughput.

References

[1] S. Singh and Y. Liu, A cloud service architecture for analyzing big monitoring data, *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 55–70, 2016.

[2] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters, in *Proc. 6th Conf. Operating System Design & Implementation*, San Francisco, CA, USA, 2004, pp. 137–150.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, Spark: Cluster computing with working sets, in *Proc. 2nd USENIX Workshop on Hot Topics in Cloud Computing*, Boston, MA, USA, 2020, pp. 1–10.

[4] J. C. Tang, M. Xu, S. J. Fu, and K. Huang, A scheduling optimization technique based on reuse in spark to defend against apt attack, *Tsinghua Science and Technology*, vol. 23, no. 5, pp. 550–560, 2018.

[5] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan, Altruistic scheduling in multi-resource clusters, in *Proc. 12th USENIX Conf. Operating Systems Design and Implementation*, Savannah, GA, USA, 2016, pp. 65–80.

[6] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, Multi-resource packing for cluster schedulers, in *Proc. 2014 ACM Special Interest Group on Data Communication*, Chicago, IL, USA, 2014, pp. 455–466.

[7] J. Wilkes, Google cluster data, https://github.com/google/cluster-data, 2020.

[8] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, Mesos: A platform for fine-grained resource sharing in the data center, in *Proc. 8th USENIX Conf. on Networked Systems Design and Implementation*, Boston, MA, USA, 2011, pp. 429–483.

[9] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al., Apache hadoop YARN: Yet another resource negotiator, in *Proc. 4th Annual Symp. Cloud Computing*, Santa Clara, CA, USA, 2013, pp. 1–16.

[10] T. Bonald, L. Massoulié, A. Proutière, and J. Virtamo, A queueing analysis of max-min fairness, proportional fairness and balanced fairness, *Queueing Syst.*, vol. 53, nos. 1&2, pp. 65–84, 2006.

[11] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, Dominant resource fairness: Fair allocation of multiple resource types, in *Proc. 8th USENIX Conf. Networked Systems Design and Implementation*, Boston, MA, USA, 2013, pp. 323–336.

[12] J. A. Hartigan and M. A. Wong, Algorithm AS 136: A K-means clustering algorithm, *J. Roy. Stat. Soc. Ser. C* (*Appl. Stat.*), vol. 28, no. 1, pp. 100–108, 1979.

[13] M. K. Pakhira, A linear time-complexity k-means algorithm using cluster shifting, in *Proc. 2014 IEEE Int. Conf. Computational Intelligence and Communication Networks*, Bhopal, India, 2014, pp. 1047–1051.

[14] J. O. Iglesias, L. Murphy, M. De Cauwer, D. Mehta, and B. O'Sullivan, A methodology for online consolidation of tasks through more accurate resource estimations, in *Proc. 2014 IEEE ACM 7thInt. Conf. Utility and Cloud Computing*, London, UK, 2014, pp. 89–98.

[15] P. Janus and K. Rzadca, SLO-aware colocation of data

center tasks based on instantaneous processor requirements, in *Proc. 2017 Symp. Cloud Computing*, Santa Clara, CA, USA, 2017, pp. 256–268.

[16] M. Carvalho, D. A. Menascé, and F. Brasileiro, Capacity planning for IaaS cloud providers offering multiple service classes, *Future Gener. Comput. Syst.*, vol. 77, no. 4, pp. 97–111, 2017.

[17] Z. Y. Hu, D. S. Li, and D. K. Guo, Balance resource allocation for spark jobs based on prediction of the optimal resources, *Tsinghua Science and Technology*, vol. 25, no. 4, pp. 487–497, 2020.

[18] S. Venkataraman, Z. H. Yang, M. Franklin, B. Recht, and I. Stoica, Ernest: Efficient performance prediction for large-scale advanced analytics, in *Proc. 13th USENIX Conf. Networked Systems Design and Implementation*, Santa Clara, CA, USA, 2016, pp. 363–378.

[19] Z. D. Bei, Z. B. Yu, H. L. Zhang, W. Xiong, C. Z. Xu, L. Eeckhout, and S. Z. Feng, RFHOC: A random-forest approach to auto-tuning Hadoop's configuration, *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1470–1483, 2016.

[20] Z. B. Yu, Z. D. Bei, and X. H. Qian, Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing, in *Proc. 23rd ACM Int. Conf. Architectural Support for Programming Languages and Operating Systems*, Williamsburg, VA, USA, 2018, pp. 564–577.

[21] Z. Y. Hu, D. S. Li, D. X. Zhang, and Y. X. Chen, ReLoca: Optimize resource allocation for data-parallel jobs using deep learning, in *Proc. IEEE Conf. Computer Communications*, Toronto, Canada, 2020, pp. 1163–1171.

[22] Y. Dong, J. Chen, Y. Tang, J. J. Wu, H. Q. Wang, and E. Q. Zhou, Lazy scheduling based disk energy optimization method, *Tsinghua Science and Technology*, vol. 25, no. 2, pp. 203–216, 2020.

[23] W. Zhang, X. Chen, and J. H. Jiang, A multi-objective optimization method of initial virtual machine fault-tolerant placement for star topological data centers of cloud systems, *Tsinghua Science and Technology*, vol. 26, no. 1, pp. 95–111, 2021.

[24] D. Shen, J. Z. Luo, F. Dong, and J. X. Zhang, VirtCo: Joint coflow scheduling and virtual machine placement in cloud data centers, *Tsinghua Science and Technology*, vol. 24, no. 5, pp. 630–644, 2019.

[25] C. Xue, C. Lin, and J. Hu, Scalability analysis of request scheduling in cloud computing, *Tsinghua Science and Technology*, vol. 24, no. 3, pp. 249–261, 2019.

Zhiyao Hu received the BSc degree in electronic information science from Beijing Institute of Technology in 2014. He is now a PhD candidate in computer science at College of Computer, National University of Defense Technology, Changsha, China. His main research interests include distributed computing, data-parallel distributed framework, and machine learning.

Dongsheng Li is a professor at National University of Defense Technology, China. He received the BSc and PhD degrees in computer science from National University of Defense Technology, China in 1999 and 2005, respectively. His research interests include distributed computing, cloud computing, and big data. He is a member of the ACM and IEEE.