Reducing Average Job Completion Time for DAG-style Jobs by Adding Idle Slots

Yubin Duan and Jie Wu
Department of Computer and Information Sciences, Temple University, Philadelphia, USA
Email: {yubin.duan, jiewu}@temple.edu

***Abstract*—Sizes of data processing jobs in cloud clusters have been growing rapidly in the big data era. It is critical to execute those jobs efficiently. The average job completion time (JCT) is a widely used metric to measure executing efficiency. JCT refers to the length of the time interval between a job's arrival to its completion. Typically, a data processing job contains multiple stages with complex precedence constraints. Carefully scheduling the processing sequence of stages within a job may significantly reduce its JCT. We investigate the scheduling problem. Our objective is to minimize the average JCT for online arrival jobs. The computation graphs of those jobs are usually directed acyclic graphs (DAGs). It makes the scheduling problem challenging. Recent works have shown that reinforcement learning (RL) agents can adaptively adjust the scheduling policies by dynamically assigning priorities for job stages. However, we notice that other factors besides stage priories may impact the JCT significantly. In particular, we observe that inserting idle slots before large jobs may reduce the waiting time of small jobs that arrive slightly later and reduce the average JCT. We analyze the benefits of inserting idle time for simple cases theoretically and show the condition in which idle slots should be inserted for two adjacent jobs. In addition, we adapt the RL-based scheduler by integrating the observation. Experiment results on both real-world and synthetic datasets show the efficiency of our scheduler. Also, a perturbation-based method is applied to demonstrate the contribution of each proposed feature.**

I. INTRODUCTION

Nowadays, the computation workload at large-scale cloud clusters has been significantly increased. As shown in [1], Alibaba's data clusters may encounter more than 70 million transactions per second. The average job completion time (JCT) could be large at those clusters. It is critical to optimize the average JCT, especially for time-sensitive applications. The average JCT is mainly determined by the cluster scheduler. In particular, the cluster scheduler may adjust the processing sequence of pending jobs and the number of executors allocated to each job. Therefore, it is an important topic for cluster operators to update their scheduling algorithms.

In this paper, our objective is to minimize the average JCT for online arrival jobs. The JCT of a job is measured by the length of duration between the job's arrival to its completion. When there are multiple jobs, we use the average JCT of those jobs to evaluate the scheduler performance. Note that a job may need to wait for available executors after its arrival. The waiting time is also a part of the JCT. For data processing engines such as Spark, their jobs usually have multiple tasks or stages. Those stages usually have complex precedence constraints. A stage cannot execute until all its predecessors are completed.

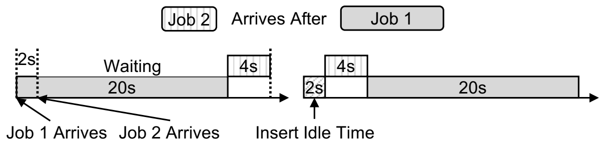


Fig. 1. Job scheduling in cloud clusters.

A directed acyclic graph (DAG) can be used to describe the job, where graph vertices represent stages and edges reveal precedence constraints. The arrival time of a job is random, and it is unknown to the scheduler. We assume those jobs are well annotated. The DAG structures including stage lengths are available to the scheduler when at jobs' arrival. The cluster scheduler would determine the start time and number of executors allocated to each stage. We assume the schedule is non-preemptive. Specifically, the executors allocated to each stage cannot be retrieved until the stage is finished. In addition, an executor can process at most one stage at a time. We propose to optimize the cluster scheduler by minimizing the average JCT for online arrival jobs represented as DAGs.

It is challenging to optimize the scheduler performance. The precedence constraints among stages and the jobs' online arrival make it difficult to generate the optimal schedule. The general DAG scheduling problem is NP-hard [2]. Existing approximation algorithms mainly discuss simplified versions of the DAG scheduling problem [3]. Considering heuristic algorithms can hardly perform well on all cases, [4] proposed to build the scheduler with reinforcement learning (RL). The RL-based scheduler can adaptively update its schedule policy for different types of DAGs. However, the RL-based scheduler lacks useful insights used by heuristic algorithms. In particular, we observe that inserting deliberate idle slots before certain jobs may decrease the average JCT. As an attempt to combine heuristic and RL approaches, this observation is integrated into an RL-based scheduler. This combination makes RL agents learn useful features faster and improve the model interpretability.

To integrate our observation of inserting deliberate idle time, we carefully design the feature and action spaces of RL agents. Specifically, we introduce DAG heights and widths to describe abstract DAG structures. The height is measured by the length of the critical path of a DAG. The width of a job is its total workload divided by the length of its critical path. RL agents can quickly capture abstract structures of different DAGs through those features. What's more, we theoretically analyze the conditions in which adding deliberate idle slots can decrease the average JCT. We also calculate the optimal lengths of idle slots for one-stage jobs. Furthermore, we evaluate our RL-based scheduler with real-world and synthetic datasets. Our experiment results on the real-world dataset present the effectiveness of the RL-based scheduler. Evaluation results on the synthetic dataset show the RL agents can determine proper idle time length which is similar to the theoretical results.

We summarize our contributions as follows:

- We investigate the online job scheduling problem for cloud clusters. We propose to insert deliberate idle slots into job execution pipelines, which can decrease the average JCT of online jobs.
- We apply RL agents to learn idle slot lengths from execution traces considering that scheduling online arrival jobs with DAG-style precedence constraints is NP-hard.
- We provide theoretical analysis for scheduling one-stage jobs. Our analysis shows in which conditions inserting idle slots can decrease the average JCT.
- We further extend the discussion for a more general case. For multiple one-stage jobs, we can sequentially investigates all pairs of adjacent jobs to determine whether idle slots should be inserted before each job.
- Experiments on real-world and synthetic datasets show that adding idle slots can decrease the average JCT. In addition, the features we selected can improve the efficiency of the RL-based scheduler.

II. RELATED WORKS

We first review the difficulties of the job scheduling problem. In general, it is NP-hard to schedule online arrival jobs. For a spatial case in which there is only one machine and jobs arrive at time 0, the problem can be optimally solved in polynomial time. The scheduling problem becomes NP-hard when jobs arrive online and have precedence constraints [5]. [6] and [7] have shown that online algorithms cannot achieve bounded competitive ratios. It is also hard to find approximate solutions for job shop scheduling problems [2].

Although the job scheduling problem is challenging, there have been many studies on this topic. From the theoretical analysis perspective, [8]–[10] investigate the scheduling problem for online arrival jobs. For example, [8] analyzes the online scheduling for unrelated machines. Different from those solutions, we consider the general case where job stages may contain parallel tasks. [11]–[15] consider more practical cases and provide heuristic solutions. In particular, Mesos [11] and Omega [12] provide scheduling platforms for large scale clusters. [16] considers a spatial DAG structure where stages only have chain structure dependency. [13] considers the case where job DAGs are heterogeneous. A heuristic algorithm that considers both resource packing and job scheduling is proposed. However, existing solutions pay little attention on adding deliberate idle slots to job execution pipelines. [17] consider inserting idle slots and [18] discusses the stage delay scheduling. [19] shows the scheduling method for heterogeneous dynamic scheduling with reinforcement learning. They did not discuss the parallelism level of each job and only considered heuristic approaches. In this paper, we consider the parallelism level and investigate RL-based schedulers.

It is intractable to optimally schedule DAG-style jobs in polynomial time [13]. [20] and [4] propose to apply RL techniques for resource management and job scheduling, respectively. In particular, [4] proposes a scheduler that trains RL agents to schedule the processing priorities of online arrival jobs. However, it ignores the fact that adding deliberate idle slots into job execution pipelines may shorten the average JCT. [21] presents an RL-based scheduler that considers adding idle slots, but it lacks theoretical analyses. We show the optimal conditions in which inserting idle slots can efficiently shorten the average JCT.

III. MODEL

Let $J = \{J_1, J_2, \dots, J_n\}$ denote the job set, where n represents the number of jobs. Each job J_i in J contains multiple stages. There are precedence relations among different stages. The precedence relations among stages within a job are modeled by a DAG. Formally, let $J_i = (S_i, E_i)$ denote the DAG for the i-th job, where S_i is the stage set and E_i is the edge set. Specifically, $S_i = \{s_{i1}, s_{i2}, \dots, s_{im}\}$ is the stage set of job J_i, where m represents the number of stages. The length of stage s_{ij} is denoted as l_{ij}. The stage length l_{ij} shows the time consumption of processing s_{ij} on a single executor. If more executors are allocated to process s_{ij}, the parallel processing time would be smaller than l_{ij}. In practice, the speedup of parallel execution is not linear and this is investigated in detail in the next section. The edge set $E_i = S_i \times S_i$ reveals the precedence constraints among stages in S_i. We use $\prec$ to denote the precedence constraint between two stages. A stage cannot be executed unless all its predecessors are processed.

We consider the online scheduling problem. Formally, we use a_i to denote the arrival time of job J_i and b_i to denote the completion time of J_i. In the online arrival setting, the arrival time a_i is stochastic. The scheduler does not know the arrival time a_i until the job J_i arrives. Each job's DAG structure and lengths of stages in the DAG are also unknown to the scheduler until the job arrives. In addition, we assume the scheduler can immediately obtain the DAG structure of a job when it arrives. Each executor processes only one job stage at a time, but non-preemptively. When job J_i arrives, our scheduler would convert the DAG-style job into an ordered list O_i that represents the processing sequence of stages in the job. When any executor becomes available, the scheduler would pick a stage from the list and assign executors to the stage and decide the parallelism level p_{ij} of every stage s_{ij}. The parallelism level p_{ij} represents the number of executors allocated to the stage s_{ij}. Furthermore, the scheduler needs to determine the idle slot's length d_{ij} for each stage s_{ij}. $d_{ij} = 0$

means there is no idle slots before s_{ij}. A schedule of a job J_i is denoted by (O_i, p_{ij}, d_{ij}).

IV. IDLE-AWARE JOB SCHEDULER

A. *Scheduling for One-stage Jobs*

We first perform theoretical analysis for simple cases and show the condition where inserting deliberate idle slots can shorten the average JCT. We mainly focus on the jobs with one stage, which eliminates the scheduling complexity introduced by the precedence constraint of stages.

Optimal conditions To derive the optimal condition for one-stage jobs, we focus on two adjacent jobs. The adjacent jobs are two jobs whose arrival times are adjacent. Let J_1 and J_2 denote those two jobs respectively. Each job has only one stage. The lengths of the two jobs are denoted as l_1 and l_2. W.l.o.g., we assume J_1 arrives earlier than J_2. The interval between the two jobs is denoted as x, i.e., $x = a_2 - a_1$. We consider the case that there is only one available machine. Otherwise, both jobs could be executed immediately when they arrive. This case usually happens when the cluster has a heavy workload. Since there is only one stage in each job, inserting idle time for stages is equivalent to inserting idle time for jobs. Let d_1 denote the length of the idle time slot inserted before job J_1. Then, the following theorem shows the condition in which inserting deliberate idle time with a proper length for J_1 could reduce the average JCT.

***Theorem** 1:* There exists an idle slot with length d_1 such that inserting it before J_1 could reduce the average JCT of J_1 and J_2 when $0 < (a_2 - a_1) \leq (l_1 - l_2)/2$ and $l_1 > l_2$.

Proof: Firstly, we calculate the average JCT without considering the deliberate idle time. W.l.o.g., we set $a_1 = 0$, since we focus on the interval $x = a_2 - a_1$ between two adjacent jobs. In this case, the job J_1 would be executed immediately at its arrival since there are no other jobs are waiting. Therefore, its termination time is $b_1 = l_1 + a_1 = l_1$. If $x < b_1$, meaning the job J_2 arrives before J_1's termination, J_2 cannot be processed until J_1 is finished. The termination time of J_2 is $b_2 = b_1 + l_2 = l_1 + l_2$. If $x \leq b_2$, then J_2 could be processed immediately when it arrives. The termination time b_2 becomes $b_2 = a_2 + l_2 = a_1 + x + l_2 = x + l_2$. Combining those two cases, the termination time of J_2 is $b_2 = \max\{a_2, b_1\} + l_2 = \max\{x, l_1\} + l_2$. Hence, the completion times of J_1 is $c_1 = b_1 - a_1 = l_1$, and that of J_2 is $c_2 = b_2 - a_2 = \max\{x, l_1\} + l_2 - x$. The average JCT is $\eta = (c_1 + c_2)/2 = (\max\{x, l_1\} + l_1 + l_2 - x)/2$.

Then, we consider the scheduling where an idle time slot with length d_2 is inserted before J_1. We note that d_1 should be larger than x for any $x > 0, l_1 > 0$, and $l_2 > 0$. Otherwise, inserting idle time would only increase the average JCT. When $d_1 < x$, the job J_2 would still be processed after J_1. In this case, inserting idle time before J_1 is equivalent to increasing the l_1 by d_1. The average JCT can still be formulated by η. Note that η is monotonically non-decreasing w.r.t. l_1. Therefore, inserting $d_1 < x$ idle time would not reduce the average JCT. However, $d_1 \leq x$ cannot guarantee to reduce the average JCT, since its formulation is no longer η.

When $d_1 \geq x$, job J_2 would be processed first since $a_1 + d_1 \geq a_2$. The termination time of J_2 is $b'_2 = a_2 + l_2 = x + l_2$. If $x \leq d_1 < x + l_2$, job J_1 would be processed right after the termination of J_2. In this case, the termination time of J_1 is $b'_1 = x + l_2 + l_1$. If $d_1 \geq x + l_2$, then J_1 needs to keep waiting even after the termination of J_2. Although it is not the optimal value of d_1, it would reduce the average JCT. In this case, the termination time of J_1 is $b'_1 = d_1 + l_1$. Combining those two cases, the termination time of J_1 is $b'_1 = \max\{x + l_2, d_1\} + l_1$ for any $d_1 \geq x$. The completion time of J_1 and J_2 become $c'_1 = b'_1 - a_1 = \max\{x + l_2, d_1\} + l_1$ and $c'_2 = b'_2 - a_2 = l_2$. The average JCT becomes $\eta' = (c'_1 + c'_2)/2 = (\max\{x + l_2, d_1\} + l_1 + l_2)/2$.

Finally, we show the condition in which delaying J_1 with optimal time length d_1 would reduce the average JCT. Note that the value of η' is determined by the choice of d_1. The lower bound of η' is $\eta'_b = (x + l_1 + 2l_2)/2$, and could be reached if $x \leq d_1 < x + l_2$. To guarantee that adding idle slots can shorten the average JCT, we let $\eta'_b < \eta$. When $x < l_1$, it is equivalent to $x + l_1 + 2l_2 < 2l_1 + l_2 - x$ or $x < (l_1 - l_2)/2$. When $x > l_1$, it is equivalent to $x + l_1 + 2l_2 < l_1 + l_2$ which never holds for $x > 0$. Above all, we have the condition that $x < (l_1 - l_2)/2$. We could examine that there exists a $d_1 \in [x, x + l_2)$ such that $\eta' < \eta$ when $x < (l_1 - l_2)/2$ and $l_1 > l_2$. □

Theorem 1 shows that adding idle slots with a proper length can improve the scheduler performance and shorten the average JCT. If the scheduler knows the value of the interval x (or $a_2 - a_1$) and l_2 in advance, i.e. before the arrival of J_2, then the optimal value of d_1 can be determined. However, in our online scheduling problem, the arrival time and the job lengths are unknown to the scheduler in advance. Therefore, it is not trivial for the scheduler to determine the length of idle time before each job when the job is ready to be executed.

***Theorem** 2:* If x and l_2 are independent random variables whose probability density functions are known, then we could formulate the expression of the optimal d_1^* which could minimize the expected value of the average JCT.

Proof: Let $f(\cdot)$ denote the probability density function of a random variable. As we have shown in the proof of Theorem 1, when an idle time slot with length d_1 is inserted before J_1, the average JCT η' depends on the relation between d_1 and x. If $0 \leq d_1 < x$, inserting an idle slot is equivalent to enlarging the length of l_1. Formally, the average JCT is $\eta' = (\max\{x, l_1 + d_1\} + l_1 + d_1 + l_2 - x)/2$. If $d_1 \geq x$, the average JCT is $\eta' = (\max\{x + l_2, d_1\} + l_1 + l_2)/2$. Combining those two cases, we have

$$\eta' = \begin{cases} (\max\{x, l_1 + d_1\} + l_1 + d_1 + l_2 - x)/2, & 0 \leq d_1 < x; \\ (\max\{x + l_2, d_1\} + l_1 + l_2)/2, & d_1 \geq x. \end{cases}$$

$$\mathbb{E}[\eta'|d_1] = \frac{1}{2}\int_0^{\infty}\left[\int_0^{d_1}(\max\{x + l_2, d_1\} + l_1 + l_2)f(x, l_2)\mathrm{d}x \right.$$
$$\left. + \int_{d_1}^{\infty}(\max\{x, l_1 + d_1\} + l_1 + d_1 + l_2 - x)f(x, l_2)\mathrm{d}x\right]\mathrm{d}l_2$$

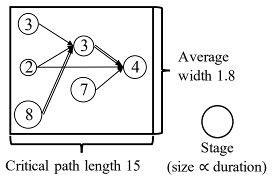


Fig. 2. Job structure and its abstraction.

The optimal value of d_1 is $d_1^* = \arg\min_{d_1} \mathbb{E}[\eta'|d_1]$. It has a closed-form expression if the probability density function f has a closed-form expression. □

Theorem 2 discusses the optimal idle slot length for two jobs and we can extend the discussion for more general cases. When there are multiple jobs, we can analyze the optimal idle slot length for every pair of adjacent jobs. Let J_i and J_{i+1} denote two adjacent jobs. For any pair of jobs $(J_i, J_{i+1}), 1 \le i < n$, we can repeatedly calculate idle slot length for J_i. If an idle slot is inserted before J_i when analyzing pair (J_{i-1}, J_i), we would skip the pair (J_i, J_{i+1}) to prevent starvation.

One-stage job abstraction Even if a job has multiple stages, we could abstract DAGs into one-stage jobs. An illustration of this is shown in Fig. 2. The number associated with each node represents the stage's processing time on one executor. The double lines indicate the critical path of the job. We can approximate a job with multiple stages into a one-stage job. Specifically, when processing the job J_i on a single executor, the processing time is the summation of stage lengths in the job. Therefore, the length of the corresponding one-stage job is $\sum_{s_{ij} \in S_i} l_{ij}$. If we treat the DAG shown in Fig. 2 as a one-stage job, the length of the stage is 27s.

B. Scheduling for General DAGs

After delivering theoretical analysis on simplified one-stage job cases, we consider a more complex but also more practical scenario: scheduling jobs with DAG structures on an arbitrary number of executors. When the stages in the DAG can have arbitrary lengths, this problem is known as NP-complete, even for offline scheduling.

Non-linear speedup Typically, the speedup of parallel execution is not linear and hard to formulate. Assigning more executors usually brings greater speedups. However, assigning too many executors to a job stage may be a waste since the speedup increases slowly after the number of executors exceeds a threshold. It is difficult to calculate the proper parallelism level for each job, especially when the computation graph of the job is a DAG. The average width might be used as a clue, as we have shown in Fig. 2. We propose to integrate the average width and critical path length into RL agents as training features. In this way, we avoid formulating the speedup ratio, but let RL agents find the proper parallelism level from execution traces.

Scheduling variables and events As we introduced in Section III, a schedule is represented by (O_i, p_{ij}, d_{ij}). Because jobs arrive online, we need to frequently update the schedule (O_i, p_{ij}, d_{ij}) to cover newly arrived jobs. Our scheduler would refresh the schedule at certain events. In particular, the (O_i, p_{ij}, d_{ij}) would be updated in the following events: i) when new jobs arrive, ii) when an occupied executor become available, and iii) when idle slots end. Compared to updating the schedule at a fixed frequency, the event-based refresh is more adaptive and efficient.

In addition to determining the scheduling events, it is also important to carefully determine the granularity of adding idle slots. In particular, we can insert idle slots before each job or before each stage. Those two scheduling granularities are denoted as job-level insertion and stage-level insertion, respectively. The job-level insertion has a smaller action space compared to the stage-level insertion since the number of potential insertion points is smaller in the job-level insertion. However, the job-level insertion may pass over the optimal solution. We use the following example to show that the job-level insertion is suboptimal. In the example, there is one available executor. Job J_1 has three stages, $s_{11} \prec s_{12} \prec s_{13}$, and each stage costs 10s to finish. Job J_2 has one stage s_{21} whose length is also 10s. If we treat job J_1 as a whole, i.e. a job with one 30s stage, then inserting idle time brings no benefit for the average JCT according to Theorem 1. However, inserting a length of $\epsilon \to 0^+$ idle slot before stage, s_{12}, could reduce the average JCT of J_1 and J_2 from 30s to 25s. Although the stage-level insertion keeps the optimal solution, the action space of the stage-level insertion is exponentially large. It is hard to go through all possible actions in polynomial time. Therefore, we propose to combine the ideas of job-level and stage-level insertions. In particular, our scheduler would investigate all available stages at each scheduling event. The available stages refer to the stages which have no unprocessed predecessors. If an idle slot is already inserted into an available stage, our scheduler would recalculate the length of the idle slot. To prevent starvation, if a stage is the next one to be processed, its idle time would not be changed.

RL-based scheduler We use RL agents to adaptively adjust the schedule (O_i, p_{ij}, d_{ij}) for online arrival jobs. The action space of the RL agent consists of the priority of each stage, the parallelism level of the next stage, and the lengths of idle slots. To generate those actions, we adapt the RL policy network proposed in [4]. In addition, the revised policy network has three types of output neurons. The first type of neurons indicates the priority of each stage. Output values of those neurons represent the probabilities of corresponding stages being selected. The second category of neurons represents the parallelism level. There are K neurons in this category, where K represents the number of executors. If the k-th neuron's output value is the largest, then the parallelism level is set to k. The third type of neuron shows the idle slot length. To reduce the action space size, we discretize the idle slot length. There are $G + 1$ neurons belonging to this type. If the g-th neuron has the largest output value. Then, the idle time length is set to $(g - 1)l_{ij}/G$, where l_{ij} is the length of the next stage s_{ij}. To prevent starvation, at most one idle slot can be inserted into each stage. In addition, we integrate the novel features we proposed to the feature space of RL agents. RL agents introduced in [4] use graph neural network (GNN) to

TABLE I
AVERAGE JCT EVALUATED IN DIFFERENT DATASETS

	(1,1)	(0,1)	(1,0)	(0,0)
Synthetic	46.3	52.7	53.5	55.0
Mixed	69.4	75.2	74.5	77.6

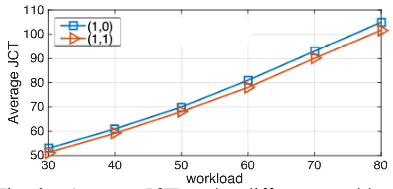


Fig. 3. Average JCT under different workload.

extract features from input DAGs. On top of the GNN, the critical path length and the average width illustrated in Fig. 2 are added into the feature space. Those features help RL agents gain a better understanding of job DAGs. The importance of those features are evaluated in Section V.

V. EXPERIMENT

A. Experiment Setup

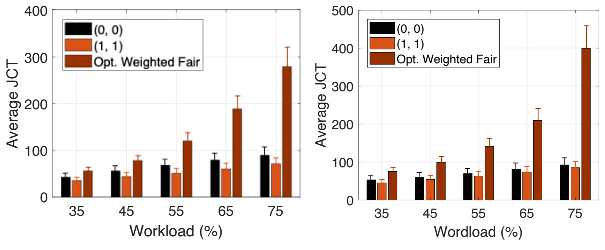
We use both real-world and synthetic datasets to evaluate our proposed scheduling method. In particular, the real-world dataset is used to show whether our scheduler could deal with online jobs with DAGs. The synthetic dataset is used to verify the efficiency of inserting idle time slots in scheduling.

There are two types of jobs in the synthetic dataset: short-term jobs and long-term jobs. Lengths of short-term and long-term jobs are fixed at 10s and 50s, respectively. The arrival of these two types of jobs is random and obeys the Poisson process. The arrival rate is controlled by the parameter λ. Specifically, the length of the interval between two adjacent events obeys the exponential distribution. The probability density function of the interval length x is

$$f(x) = \lambda \exp(-\lambda x), x \geq 0.$$

For experiments using the synthetic datasets, we set the parameter λ such that the average interval between two adjacent jobs is 10s. According to the Theorem 1, the average JCT can be reduced by inserting idle time slots under this setting. We extract TPC-H queries to build the real-world dataset. In total, the TPC-H dataset contains 22 different queries and the input data size can be chosen from 1GB to 3TB. In our real-world dataset, we randomly sample 10^3 jobs from all 22 queries with input sizes varying from 1GB to 100GB. The arrival of jobs in the real-world dataset also obeys the Poisson process. The average interval between two adjacent jobs is set to 40s, which keeps the cluster load at about 80%. In addition to the real-world and synthetic datasets, a mixed dataset is used in the experiment. Specifically, we randomly sample entries from the real-world and synthetic datasets and use the sampled entries to build the mixed dataset. We use a hyper-parameter $\alpha \in [0, 1]$ to adjust the percentage of data entries selected from the synthetic dataset. $\alpha = 0$ represents all data samples are from the real-world dataset.

The policy network is trained on a lab PC equipping a Nvidia GTX 1080 GPU. During training, the RL agent gradually improve its performance by learning from history traces. The RL agent can only generate random policies at early training iterations. Therefore, we reduce the initial cluster load by enlarging the length of job intervals. The cluster workload is determined by the parameter λ of the Poisson process. Specifically, if λ is larger, then the number of job arrivals in a fixed time interval would increase, which means the workload of the cluster is heavier. We gradually increase the workload alone with the training process. Moreover, a simulator is used to estimate the reward for the RL agent. Using the simulator can improve the training speed.



(a) synthetic dataset (b) mixed dataset

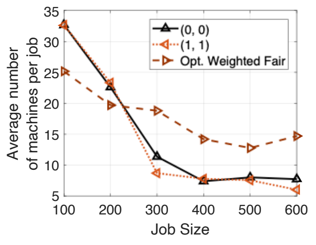
Fig. 4. Investigating different measurement metrics.

B. Experiment Results

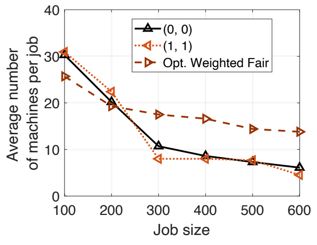
We investigate the impact of our selected features and the idle slots to the RL agent. For simplicity, we use a two-tuple to distinguish different RL agents. Specifically, the first element indicates whether the agent takes selected features. The value 0.5 represents using partial features, i.e., either critical path or average width, but not both. The second elements indicates whether the agent considers inserting idle slots.

First, we show the comparison of the average JCT in the real-world dataset. The workload is measured by the percentage of busy executors. We compare our idle aware job scheduler (labeled as (1, 1)) to the SOTA RL-based job scheduler [4] (labeled as (0, 0)) The workload is adjusted by changing the parameter λ. Fig. 3 shows the experiment result. The red line in the figure represents the average JCT achieved by our scheduler. The average JCT increases with the workload. This is because a larger workload makes scheduling more challenging. If the workload is very low such that the arrival interval between two jobs is larger than the length of the largest job, then even a FIFO policy could achieve the optimal JCT for each job. In contrast, if the workload is larger than 100%, the job would accumulate in the job queue no matter which scheduling policy is used. Therefore, in our experiment, we control the workload under 80%. The experiment results show that no matter what the level of workload is, the average JCT can be further reduce by inserting idle slots.

We also evaluate the effectiveness of the features we proposed. In particular, two types of RL agents are compared in the experiment. The first type is the RL agent we proposed, which is denoted as $(1, 1)$. This notation means that the RL agent can insert idle slots and it uses the features we proposed on top of GNN. The second category of RL agent can insert idle slots but it does not consider the additional features we



(a) synthetic dataset



(b) mixed dataset

Fig. 5. Investigating machine allocation scheme.

proposed. We use $(0, 1)$ to denote this category of RL agent. We use the mixed dataset with $\alpha = 0.5$ and the synthetic dataset to perform the comparison. We use the same device to train those two types of RL agents with the same training time length. Table 3 shows the evaluation results. In addition to those two types of RL agents, we also show the evaluation results on RL agents that cannot insert idle slots. The notation $(0, 1)$ refers to the agent who cannot insert idle slots but uses the features we proposed. $(0, 0)$ refers to the baseline, where idle slots and addition features are not considered by RL agents. From the evaluation results, we can find that using the features we proposed can efficiently improve RL agents' performance.

Fig. 4 shows the performances of different learning agents and optimal weighted fair, a heuristic algorithm. The experiment results for both the synthetic and mixed dataset show that the learning agent with additional features and idle time would have a better performance. Compared to heuristic algorithms, a learning based approach can achieve much smaller average JCT, especially when the workload is heavy.

Finally, we illustrate the number of machines allocated to each job in Fig. 5. We find our RL-based agent allocates more machines to smaller jobs. Letting smaller jobs finish first would help to reduce the average JCT, since the wait time for other queuing jobs could be reduced. This becomes more clear when comparing the RL-based schedulers with the optimal-weighted-fair scheduler. It shows that RL agents can find the proper parallelism level for each job based on the job's length.

VI. Conclusion

We propose to improve the scheduler of data processing clusters by minimizing the average JCT. The data processing jobs arrive online and have complex DAG structures. We have observed that the average JCT can be reduced by adding deliberate idle slots before certain jobs. For simplified one-stage jobs, we theoretically analyze the conditions in which deliberate idle slots should be inserted. In addition, we show the optimal lengths of deliberate idle slots for the simplified case. For general DAG-style jobs, we adapt an RL-based scheduler by combining our observation of inserting idle slots. We investigate features that can be used to integrate the observation. The average DAG width and the critical path length can be used to capture the abstract DAG structure. DAG details, such as precedence constraints, can be extracted by graph neural networks. We evaluate our RL-based scheduler on both real-world and synthetic datasets. Experiment results show the length of idle slots learned by RL agents are close to theoretical results in simplified cases. For general data processing jobs from the real-world dataset, the RL-based scheduler also outperforms a heuristic baseline.

References

[1] F. Li, "Cloud-native database systems at alibaba: Opportunities and challenges," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, 2019.

[2] M. Mastrolilli and O. Svensson, "(acyclic) job shops are hard to approximate," in *FOCS*. IEEE, 2008, pp. 583–592.

[3] S. Im, N. Kell, J. Kulkarni, and D. Panigrahi, "Tight bounds for online vector scheduling," in *FOCS*. IEEE, 2015, pp. 525–544.

[4] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of SIGCOMM*. ACM, 2019, pp. 270–288.

[5] E. L. Lawler, J. K. Lenstra, A. H. R. Kan, and D. B. Shmoys, "Sequencing and scheduling: Algorithms and complexity," *Handbooks in operations research and management science*, vol. 4, pp. 445–522, 1993.

[6] N. Garg and A. Kumar, "Minimizing average flow-time: Upper and lower bounds," in *FOCS*. IEEE, 2007, pp. 603–613.

[7] J. S. Chadha, N. Garg, A. Kumar, and V. Muralidhara, "A competitive algorithm for minimizing weighted flow time on unrelatedmachines with speed augmentation," in *STOC*. ACM, 2009, pp. 679–684.

[8] S. Im and B. Moseley, "An online scalable algorithm for minimizing k-norms of weighted flow time on unrelated machines," in *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2011, pp. 95–108.

[9] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlós, "On scheduling in map-reduce and flow-shops," in *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 289–298. [Online]. Available: https://doi.org/10.1145/1989493.1989540

[10] L. A. Goldberg, M. Paterson, A. Srinivasan, and E. Sweedyk, "Better approximation guarantees for job-shop scheduling," *SIAM Journal on Discrete Mathematics*, vol. 14, no. 1, pp. 67–92, 2001.

[11] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *NSDI*, vol. 11, no. 2011, 2011, pp. 22–22.

[12] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," 2013.

[13] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters," in *USENIX OSDI*, 2016, pp. 81–97.

[14] S. Im, N. Kell, J. Kulkarni, and D. Panigrahi, "Tight bounds for online vector scheduling," in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. IEEE, 2015, pp. 525–544.

[15] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *INFOCOM*. IEEE, 2017, pp. 1–9.

[16] E. Anderson, D. Beyer, K. Chaudhuri, T. Kelly, N. Salazar, C. Santos, R. Swaminathan, R. Tarjan, J. Wiener, and Y. Zhou, "Value-maximizing deadline scheduling and its application to animation rendering," in *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 2005, pp. 299–308.

[17] H. Zhu and H. Zhou, "Single machine predictive scheduling using inserted idle times," *Journal of Applied Mathematics*, vol. 2014, 2014.

[18] W. Shao, F. Xu, L. Chen, H. Zheng, and F. Liu, "Stage delay scheduling: Speeding up dag-style data analytics jobs with resource interleaving," in *ICPP*, 2019, pp. 1–11.

[19] N. Grinsztajn, O. Beaumont, E. Jeannot, and P. Preux, "Readys: A reinforcement learning based strategy for heterogeneous dynamic scheduling," in *IEEE Cluster 2021*, 2021.

[20] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 2016, pp. 50–56.

[21] Y. Duan and J. Wu, "Improving learning-based dag scheduling by inserting deliberate idle slots," *IEEE Network*, vol. 35, no. 6, pp. 133–139, 2021.