# An Approximation for Job Scheduling on Cloud with Synchronization and Slowdown Constraints

Dejun Kong[1], Zhongrui Zhang[1], Yangguang Shi[2], Xiaofeng Gao[1,*]

[1]*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China*
[2]*School of Computer Science and Technology, Shandong University, Qingdao, China*
{kdjkdjkdj99,2020zzr}@sjtu.edu.cn, shiyangguang@sdu.edu.cn, gao-xf@cs.sjtu.edu.cn

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10229078

***Abstract*—Cloud computing develops rapidly in recent years and provides service to many applications, in which job scheduling becomes more and more important to improve the quality of service. Parallel processing on cloud requires different machines starting simultaneously on the same job and brings processing slowdown due to communications overhead, defined as synchronization constraint and parallel slowdown. This paper investigates a new job scheduling problem of makespan minimization on uniform machines and identical machines with synchronization constraint and parallel slowdown. We first conduct complexity analysis proving that the problem is difficult in the face of adversarial job allocation. Then we propose a novel job scheduling algorithm, United Wrapping Scheduling (UWS), and prove that UWS admits an $O(\log m)$-approximation for makespan minimization over $m$ uniform machines. For the special case of identical machines, UWS is simplified to Sequential Allocation, Refilling and Immigration algorithm (SARI), proved to have a constant approximation ratio of $8$ (tight up to a factor of $4$). Performance evaluation implies that UWS and SARI have better makespan and realistic approximation ratio of $2$ compared to baseline methods United-LPT and FIFO, and lower bounds.**

***Index Terms*—job scheduling, cloud computing, synchronization constraint, parallel slowdown**

## I. Introduction

Along with the rapid development of data centers and cloud service, cloud computing is extensively used in various kinds of computer services nowadays [1]. Clients upload pending jobs to the cloud service and get the processing results back from it, no need for considering the process of computing. The integration of data storage and computing resources provides the clients with better network resource management and lower cost. However, larger scales of network resources and service demands become more and more challenging for the service management [2]. In this problem, job scheduling, which mainly concentrates on the allocation of different jobs to parallel machines or threads, is a critical part because a good scheduling can improve both the resource utilization and processing efficiency [3], studied by many researchers.

Motivated by applications in modern cloud computing, this paper studies job scheduling under the settings where each job contains multiple indivisible parts that can be processed simultaneously on parallel machines. As a typical job processed on cloud in recent years, the training of machine learning methods is widely employed in a variety of applications like image recognition [4], etc. The training process can be either time-consuming or very fast, depending on the data scale and model complexity, which contributes to a large variance of the processing time of different jobs. To accelerate the training process, distributed neural networks training is developed and studied [5], where the training instance can be split into several. As the training of a complex model usually contains several processes [6], one process has to be accomplished on one machine, which cannot be split any more. Such a part of one job is defined as subjob. Then job scheduling is to allocate these indivisible subjobs to different parallel machines. Similar scenarios of *job scheduling problem on parallel machines* to decide how to allocate subjobs to parallel machines with minimum makespan are discussed in [7].

In the procedure of job processing on cloud, the clients are usually required to affirm the amount of the parallel computing resource [8]. A practical reason is that clients need an accurate and acceptable cost on resource allocation in advance. Dynamic computing resource allocation improves the computing efficiency but cannot give a reliable cost prediction in complex cloud computing environment. Therefore, fixed parallel computing resource allocation is still employed in many cloud computing scenarios, namely, allocating all computing resources at the very beginning and processing on them until work is done. This means that the parallel computing resources are allocated to jobs simultaneously following the requests and hence subjobs assigned to different parallel resources always start at the same time (the time resources are allocated). We denote such synchronous starting as synchronization constraint. Similarly, MPI jobs scheduled on supercomputers may also be subject to synchronization constraint as MPI jobs are typically tightly coupled and synchronize often by barriers or point-to-point communication [9]. If one delay, others would soon stop and wait for it.

When parallel processing of a job is in progress, subjobs have to exchange data with each other in real time, which brings network transmission delay and results in slowdown [5], [10]. A significant cost is the communication cost (e.g. data-shuffle [11]) of transmitting the intermediate data of a job among different machines [12]. It is indicated in [13] that it spends more time processing jobs with multiple threads on Spark due to frequent switching among threads. To capture this

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on September 08,2023 at 08:16:33 UTC from IEEE Xplore. Restrictions apply.

feature in parallel processing, here we introduce a slowdown function $g : \mathbb{N}_{>0} \mapsto \mathbb{R}_{>0}$ and model the speed of a machine $j$ in processing a job $i$ as $s_j \cdot g(e_i)$, where $s_j$ is the full speed of machine $j$, and $e_i$ is the number of machines used by subjobs of $i$. Data driving approaches are used to obtain a proper description of $g(\cdot)$. In particular, we conduct experiments shown in Figure 1 on five benchmarks on the platform of PARSEC 3.0 [14] to observe slowdown versus thread number, deriving a slowdown function $g(e_i) = 1 - \delta(e_i - 1)$, where $e_i$ is thread number, $\delta > 0$ is an input *slowdown coefficient*.

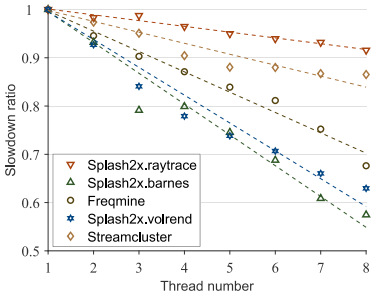



Fig. 1. Parallel slowdown experiment: Five benchmarks on data mining (Freqmine), Streamcluster and high-performance computing (Barnes, Raytrace, Volrend) are tested with $\delta = 0.0421, 0.0638, 0.0227, 0.0120, 0.0577$.

As a whole, in this paper we consider a new variation of job scheduling problem on parallel machines induced from cloud computing. Jobs with indivisible subjobs are submitted to cloud for processing. Subjobs of the same job on the same machine should be processed continuously in a non-preemptive manner. The start time of a job $i$ on a machine $j$ refers to the start time of the first subjob of $i$ on $j$, and it is required that the start time of a job should be the same on all the machines that it uses, namely, satisfying the synchronization constraint. During processing, the speeds of machines are given by the above formula involving the slowdown function. The parallel machines can be identical or uniform, because heterogeneous computing resources, which usually exist in large-scale data centers after years of updates, and different initialization of virtual machines, can lead to different executing speeds for parallel processing. Our goal is to find a scheduling strategy assigning subjobs to appropriate machines so that the overall finishing time is minimized. An introductory example is illustrated in Figure 2. A feasible solution shown in Figure 3 illuminates a solution.

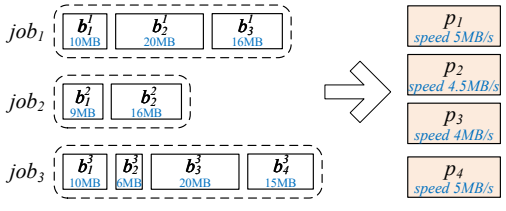



Fig. 2. A toy example: Three jobs with various sizes of subjobs require to be scheduled over four heterogeneous machines with different computing speeds. Due to the synchronization, communication and file management issues, the parallel computing speed decreases by a slowdown coefficient $\delta = 0.03$.

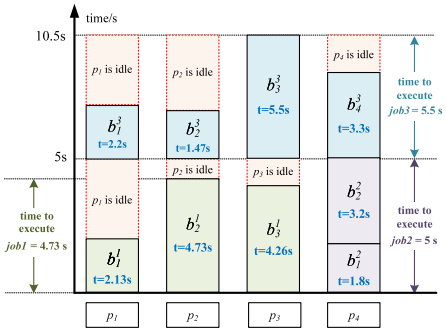



Fig. 3. A feasible solution for Figure 2. Here $job1$ has been assigned to three machines 1, 2, 3; $job2$ to one single machine 4; and $job3$ to all four machines. Idle slots are sacrificed due to the synchronization constraint.

Our contributions are illustrated as follow. First, we formulize the model of the problem and investigate the hardness of this problem, proving that the scheduling problem with the synchronization constraint admits no $O\big(\frac{n}{2^{(\log n)^{3/4+\epsilon}}}\big)$-approximation under a conventional computational hardness assumption. Then we consider parallel processing and design an algorithm with three phases: *Sequential Allocation*, *Refilling*, and *Immigration* (*SARI*) for identical machines, eventually achieving a feasible solution with a constant approximation ratio of 8. In all, SARI balances the gain in the parallel speedup and the drawback of synchronization and slowdown. After that, we develop another subroutine called *Wrapping* to utilize SARI on uniform machines. Together with SARI, the entire framework is named as *United Wrapping Scheduling* or *United We Stand* (*UWS*), trying to unite subjobs of one job together, which is proved to have an $O(\log m)$-approximation ratio. Finally, we conduct extensive numerical evaluations to illuminate the performance of SARI and UWS. Sensitivity analysis upon different parameters and the performance are explored with synthetic datasets. Results imply the superiority of the proposed algorithms and a much better real approximation ratio. sTo the best of our knowledge, we are the first to provide polylog-approximation for uniform machine scheduling problem with synchronization and slowdown.

## II. Related Work

Job scheduling is a classical problem studied for tens of years. [10] introduces a bulk-synchronous parallel model as a bridge between hardware and software in computer, which enables task speedup by parallel processing, making parallel job scheduling in computing have practical significance. [15] first studies $P||C_{\max}$ and proposes a greedy-based algorithm with a constant approximation ratio of 2, which allocates job $i$ to machine $j$ with the least load so far. Later, [16] extends the above algorithm to $Q||C_{\max}$ and proves to have an $O(\log m)$-approximation ratio. Meanwhile, they also design an 8-approximate algorithm for $Q||C_{\max}$, called ASSIGN-R. ASSIGN-R works with an estimation $\Lambda \geq \texttt{OPT}$ of the optimal makespan $\texttt{OPT}$ and allocates job $i$ to the slowest machine. According to the classification of [9], the scheduling

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on September 08,2023 at 08:16:33 UTC from IEEE Xplore. Restrictions apply.

problem discussed in this paper belongs to moldable job scheduling. Without considering indivisible subjob setting, the approximation ratio of related algorithms is continuously improved from 2 [17] to $\frac{3}{2}+\epsilon$ [18], [19]. As for scheduling on uniform machines, [20] shows that there is a PTAS scheduling plan for an unconstrained problem, while [21] proposes that the scheduling can even be improved to EPTAS in recent progress. [22] proposes Hibernation-Aware Dynamic Scheduler to schedule an integration of independent tasks with deadline constraints.

As for synchronization and slowdown constraint, some similar studies have been conducted in recent years more or less. [23] considers similar synchronization constraint, where parallel jobs are completed at the same time on all processors. In [10], synchronization among different parallel threads is considered for information sharing. Parallel slowdown on identical machines is studied in [24], while they consider the variation of total speed in sublinear trend. Scheduling with the constraint of setup time is a well researched topic in recent research. [25] extends the makespan minimization problem of one job processed by one machine to that by multiple machines with an arbitrary setup time. The lower bound of the approximation ratio is proved to be around 1.582.

## III. Model and Preliminaries

### A. Problem Statement

The input of the job scheduling problem consists of a set $\mathcal{J}$ of $n$ jobs $\{1,2,\cdots,i,\cdots,n\}$ and a set $\mathcal{M}$ of $m$ parallel machines $\{1,2,\cdots,j\cdots,m\}$. Job $i$ has a positive integer *size* $l^i \in \mathbb{N}_+$. In this paper, we pay attention to *identical machines* and *uniform machines* defined in Definition 1 progressively.

**Definition 1** (Uniform Machines and Identical Machines)**.** For a job scheduling problem defined over $n$ jobs and $m$ machines, the machines are uniform if each machine $j$ is associated with a specific processing speed $s_j \in \mathbb{N}_+$, and therefore the processing time of each job $i$ on machine $j$ is given by $l^i/s_j$. In particular, a set of uniform machines is identical if for some number $s \in \mathbb{N}_+$, $s_j = s$ holds for each machine $j$.

Each job $i$ consists of a set $\mathcal{B}^i = \{b_1^i, b_2^i, \cdots, b_k^i, \cdots, b_{n_i}^i\}$ of $n_i \in \mathbb{N}_+$ *subjobs* with sizes $\{l_k^i\}_{k\in[n_i]}$. It is assumed that the sizes of subjobs and the related job satisfy $\sum_{k=1}^{n_i} l_k^i = l^i$. Different subjobs of a job are allowed to be scheduled on different machines, while each single subjob can be only assigned to one machine exactly. For each machine $j$, subjobs allocated to machine $j$ are processed in a *non-preemptive* way. The objective is to find a schedule to minimize the total makespan, namely the time to complete all the jobs. Our study on job scheduling for makespan minimization includes two specific settings, *synchronization constraint* and *parallel slowdown*, formulized as follow.

*a) Synchronization constraint:* Consider the case where the subjobs of job $i$ are allocated to parallel machines. Let $\mathcal{B}_j^i$ be the set of subjobs of job $i$ that are allocated to machine $j$. When $\mathcal{B}_j^i \neq \emptyset$, all the subjobs in $\mathcal{B}_j^i$ are required to be processed continuously. For each non-empty $\mathcal{B}_j^i$, the *starting time* of $\mathcal{B}_j^i$ is denoted by $ts_j^i$, namely, the earliest time when the subjobs of job $i$ start to be processed on machine $j$. The scheduling algorithm is required to *synchronize* $ts_j^i$ for all $\mathcal{B}_j^i$ of each job $i$, which means to ensure that $ts_j^i = t^i$ for $t^i$ only depends on job $i$ over all the machines $j$ with $\mathcal{B}_j^i \neq \emptyset$.

*b) Parallel slowdown:* When the subjobs of a same job $i$ are distributed over $e_i \in \mathbb{N}_+$ machines, the actual speed of each machine $j$ processing the subjob set $\mathcal{B}_j^i$ is lower than the inherent speed $s_j$ of machine $j$ by a factor of $g(e_i)$, where $g : \mathbb{N}_+ \mapsto (0,1]$ is referred to as the *slowdown function*. The slowdown function is shown in Equation (1):

$$g(e_i) = 1 - \delta \cdot (e_i - 1), e_i \in [1, \frac{\delta+1}{\delta}), \tag{1}$$

where $\delta$ is the *slowdown factor* with $\delta \in (0,1)$.

Taking both the synchronization constraint and parallel slowdown into consideration, the processing time $tp_j^i$ and the finishing time $C_j^i$ of job $i$ on machine $j$ can be expressed as

$$tp_j^i = \frac{\sum_{b_k^i \in \mathcal{B}_j^i} l_k^i}{s_j^i \cdot g(e_i)}, \quad C_j^i = t^i + tp_j^i. \tag{2}$$

Then the makespan can be formulated as Equation (3):

$$C_{\max} = \max_{i\in[n]} \max_{j\in[m]} C_j^i. \tag{3}$$

Therefore, the objective of our study is to minimize Equation (3). The notations are summarized in Table I for reference.

TABLE I
Definitions and Notations

| Symbols | Definitions |
|---|---|
| $n$, $m$ | The number of jobs, the number of machines. |
| $b_k^i$, $n_i$ | The $k$-th subjob of job $i$, the number of subjobs of job $i$. |
| $s_j$ | The processing speed of machine $j$. |
| $l^i$, $l_k^i$ | The size of job $i$, the size of subjob $b_k^i$. |
| $e_i$ | The number of machines executing job $i$. |
| $\mathcal{J}$, $\mathcal{M}$ | The set of jobs, the set of machines. |
| $\mathcal{B}^i$ | The set of subjobs $\{b_1^i, b_2^i, \cdots, b_{n_i}^i\}$ of job $i$. |
| $\mathcal{M}^i$ | The set of machines $\{i_1, i_2, \cdots, i_{e_i}\}$ for job $i$. |
| $\mathcal{B}_j^i$ | The set of subjobs of job $i$ allocated to machine $j$. |
| $g(\cdot)$, $\delta$ | The slowdown function, the slowdown factor. |
| $ts_j^i$, $tp_j^i$ | The time to start processing $\mathcal{B}_j^i$, the time to process $\mathcal{B}_j^i$. |
| $t^i, tp^i$ | The time to start processing job $i$, the time to process job $i$. |
| $C_j^i$ | The finishing time of machine $j$ on job $i$. |
| $C_j$ | The finishing time of machine $j$. |

### B. Triplet Notations and Preliminaries

To clearly represent different job scheduling problems, we adopt the three-field notations [26], [27]in the form of $\alpha|\beta|\gamma$, where $\alpha$ represents the *machine environment*, $\beta$ specifies the *processing characteristics* and *constraints*, and $\gamma$ denotes the *objective(s)*. We express the identical machines by $\alpha = P$ and uniform machines by $\alpha = Q$. For processing characteristics and constraints, we use $\beta = \{\text{sync, slow}\}$ to show synchronization constraint and parallel slowdown, and $\beta = \circ$ for the

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on September 08,2023 at 08:16:33 UTC from IEEE Xplore. Restrictions apply.

scenario where each job should be entirely allocated to a single machine.[1] The objective considered in our paper is makespan minimization, which is denoted by $\gamma = C_{\max}$.

To solve $Q||C_{\max}$, a well-known technique is the 2-approximation algorithm *Largest Processing-Time-first* (LPT), presented in [28]. LPT considers the jobs in a non-increasing order of the sizes and allocates each job to the machine with the earliest completion time. Similar to the existing literature in job scheduling (e.g. [29]–[32]), in this paper we only consider the machines with geometrically separable speeds, which is illustrated in Definition 2 and Lemma 1.

**Definition 2** (Geometrically Separable Speeds [29])**.** Speeds of uniform machines are said to be geometrically separable if $\max_{j\in[m]} s_j \leq \alpha \cdot m \cdot \min_{j\in[m]} s_j$, where $\alpha$ is constant.

**Lemma 1** ( [29])**.** *When the speeds of the machines are geometrically separable, one can partition the machines into at most $\lceil \log m \rceil$ groups so that in each group, the fastest machine is at most two times faster than the slowest one.*

### C. Complexity Analysis

The synchronization constraint distinguishes our work from the existing works which allow subjobs to be processed at any time (e.g. [33]). In particular, consider the case that the allocation of the jobs to the machines is fixed by an adversary. Then the induced scheduling problem, which requires determining $ts_j^i$, is proved to be hard to approximate in Lemma 2.

**Lemma 2.** *The scheduling problem induced by fixed job allocation has no $O\big(\frac{n}{2^{(\log n)^{3/4+\epsilon}}}\big)$-approximation for any $\epsilon > 0$ under the assumption that $NP \neq BPTIME\big(2^{(\log n)^{O(1)}}\big)$.*

*Proof.* We make reduction from the graph coloring problem. Given an instance of the coloring problem on a graph $G = (V, E)$ with $n$ nodes and $m$ edges, we construct an instance $\mathcal{I}$ of the induced scheduling problem as follows. For each node $i$ (resp. edge $j$), we add a job (resp. machine) to $\mathcal{I}$. The speeds of all the machines are set to 1. For a pair of nodes $\{i, i'\} \in V \times V$, if they are connected by an edge $j$ in $E$, we allocate two subjobs of jobs $i$ and $i'$ to machine $j$, respectively. The size of each subjob of job $i$ is set to $\mathrm{g}(\deg(i))$, where $\deg(i)$ represents the degree of node $i$ in graph $G$. Such a setting is consistent as $\deg(i) \leq m$ holds for each node $i$, and it ensures that the processing time of each subjob on machine is 1. Such an instance $\mathcal{I}$ can be built in polynomial time.

Given a feasible solution of the instance $\mathcal{I}$ with makespan LB, we build a solution of the original graph coloring problem as follow. For each job $i$, if its subjobs start at time $t^i$, we color the corresponding node $i$ with $(t^i + 1)$-th color. Such a coloring is feasible since for two nodes $i, i'$ connected by edge $j$, the corresponding jobs cannot start simultaneously for they share the machine $j$. Furthermore, our setting on the sizes of the subjobs ensures that each color ranging from 1 to LB is allocated to at least one node. Thus, we obtain a feasible coloring of $G$ with LB colors. By the inapproximability of graph coloring [34], this proposition holds. □

Lemma 2 indicates that one obstacle in scheduling the subjobs of jobs under the synchronization constraint is the difficulty in deciding the processing order of the jobs.

## IV. Scheduling Jobs Sequentially

We first consider a simple technique scheduling the job integrally, namely, allocating each job to a single machine. In such a case, one can adopt the existing algorithms for job scheduling to choose the machine for each job. The approximation ratio analysis is presented in Theorem 1.

**Theorem 1.** *$Q|$sync, slow$|C_{\max}$ has a $(\beta \cdot \max_i n_i)$-approximation algorithm, if $Q||C_{\max}$ admits a $\beta \geq 1$ approximation.*

*Proof.* For a given instance $\mathcal{I}$ of $Q|$sync, slow$|C_{\max}$, let $\mathcal{I}'$ be the problem instance of $Q||C_{\max}$ over the same sets of jobs and machines, and $\mathcal{I}^*$ be the relaxed version of $\mathcal{I}$ where subjobs can be processed at any time and $g(x) \equiv 1$. Denote the optimal makespan of $\mathcal{I}$, $\mathcal{I}'$ and $\mathcal{I}^*$ by OPT, OPT′, and OPT*, respectively. It is easy to see that OPT $\leq$ OPT*. We prove this proposition by showing that OPT′ $\leq \max_i n_i \cdot$ OPT*. In particular, we construct a feasible solution $\widehat{\mathcal{S}}$ of $\mathcal{I}'$ based on the optimal solution $\mathcal{S}^*$ of $\mathcal{I}^*$ in the following way. For each job $i$, the solution $\widehat{\mathcal{S}}$ allocates it to an arbitrary machine $j$ satisfying $\sum_{k, b_k^i \in \mathcal{B}_j^i} l_k^i \geq \frac{l^i}{n_i}$ in $\mathcal{S}^*$. By the pigeonhole principle, such a machine $j$ always exists. As a result, the load of each machine in $\widehat{\mathcal{S}}$ is at most $\max_i n_i$-times larger than the load of the same machine in $\mathcal{S}^*$. Therefore, this proposition holds. □

With the 8-approximate algorithm Assign-R and Theorem 1, Corollary 1 can be directly inferred as follow.

**Corollary 1.** *One can obtain an $\big(2 \cdot \max_i n_i\big)$-approximation result for $Q|$sync, slow$|C_{\max}$ in polynomial time.*

The algorithm guaranteed in Corollary 1, United-LPT, is described in Algorithm 1. Here the Boolean variable $x_j^i \in \{0, 1\}$ represents whether a job $i$ is allocated to machine $j$.

**Algorithm 1:** United-LPT

**Input:** Jobs $\mathcal{J}$, machines $\mathcal{M}$
**Output:** $\mathbf{x} = \{x_j^i\}_{i\in[n], j\in[m]}$

```
Initialize x ← 0⃗;
Sort the jobs in a non-increasing order of lengths;
foreach job i ∈ J do
    j* ← argmin_j (Σ_{i'<i} l^{i'} · x_j^{i'}) / s_j ; x_{j*}^i ← 1;
        //assign job to machine with earliest makespan
```

## V. Scheduling on Identical Machines

United-LPT proposed in Section IV does not exploit the benefits of parallel processing and performs bad with more subjobs. In this section, we introduce SARI algorithm for parallel scheduling on identical machines with three steps:

[1]The symbol ∘ is omitted in three-field notations for scheduling problems.

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on September 08,2023 at 08:16:33 UTC from IEEE Xplore. Restrictions apply.

*Sequential Allocation* with threshold as basic scheduling, *Refilling* to satisfy synchronization barrier, and *Immigration* for parallel slowdown, achieving a constant approximation ratio.

### A. Lower Bound Analysis

With identical machine, the speed of each machine $j$ is $s$. Without parallel slowdown, the lower bound LB of makespan in any scheduling must be restricted by Equation (4):

$$\texttt{LB} = \max\left\{\frac{\sum_i l^i}{m \cdot s}, \max_i \frac{\max_k l_k^i}{s}\right\}, \tag{4}$$

where the first term represents the average processing time per machine, and the second term represents the processing time of the largest subjob. Under the best condition, LB can be reached as an optimal solution. Let OPT be an optimal solution of $P|\text{sync,slow}|C_{max}$. $\texttt{LB} \leq \texttt{OPT}$ always holds.

From Equation (1), the benefit of parallel processing may be lessened by parallel slowdown, described in Lemma 3.

**Lemma 3.** *The best number of identical machines processing one job is* $\lfloor \frac{\delta+1}{2\delta} \rfloor$ *or* $\lceil \frac{\delta+1}{2\delta} \rceil$ *if arbitrarily splittable.*

*Proof.* To minimize the processing time $tp_i$ of job $i$ over identical machines, one should balance the load over the machines chosen for job $i$. Assume that job $i$ is split evenly over $e_i$ machines, then on each of these machines $j$, the processing time of job $i$ is $f(e_i) = \frac{1}{e_i \cdot [1-\delta(e_i-1)]}$. The derivative of $f(e_i)$ indicates that $tp_i$ is minimized when $\frac{\mathrm{d}}{\mathrm{d}e_i} f(e_i) = -\frac{1+\delta-2\delta \cdot e_i}{e_i^2 \cdot [1-\delta(e_i-1)]^2} = 0$, giving the desired result. □

### B. Sequential Allocation with Threshold

We first assign subjobs to machines without taking the synchronization constraint and parallel slowdown into consideration. The algorithm *Sequential Allocation with Threshold* (SA), as shown in Algorithm 2, packs subjobs over machines in sequence [35]. Once the total processing time of the current machine exceeds LB, SA proceeds to allocate remaining subjobs to the next machine. We define $S_j, j \in [m]$ as a queue storing the sequence of subjobs allocated to the machine $j$. With Equation (1), the guarantee of SA is cast in Lemma 4.

**Algorithm 2:** Sequential Allocation with Threshold

**Input:** Jobs $\mathcal{J}$, subjobs $\mathcal{B}^i$ for each job $i$, machines $\mathcal{P}$, processing speed $s$, threshold LB

**Output:** The scheduling queue on each machine $j$ $S_1, S_2, \cdots, S_m$

```
Initialize C ← 0; j ← 0; S_j ← ∅ for each j;
foreach job i do
    foreach subjob b_k^i do
        if C > LB then        //C counts current workload
            j ← j + 1;        //go to next machine
            Reset C ← 0;
        S_j ← S_j ∪ {b_k^i} and update C ← C + l_k^i/s;
```

**Lemma 4.** *Sequential Allocation with Threshold always generates a feasible scheduling plan with* $C_{\max} \in [\texttt{LB}, 2\texttt{LB}]$.

### C. Refilling with Synchronization Constraint

Based on SA, a 2LB scheduling plan without synchronization constraint is given. Next, we concentrate on rescheduling the previous plan to satisfy synchronization constraint.

Refilling has two stages: in the first stage, we refill the machines according to *Refilling Rules* to guarantee a 3LB upper bound of makespan, while in the second stage we reduce this 3LB upper bound to 2LB by *Switching Rules*.

Refilling Rules: For each job $i$, if subjobs $\mathcal{B}^i_{i_{e_i}}$ scheduled on the last machine assigned to one job satisfy $\sum_{k, b_k^i \in \mathcal{B}^i_{i_{e_i}}} l_k^i < s\texttt{LB}$ $(e_i > 1)$, reschedule $\mathcal{B}^i_{i_{e_i}}$ to the end of the queue $S_{i_1}$ of the first machine assigned to the job. After that, reverse the sequence of $S_{i_1}$ if the job $i$ does not satisfy the synchronization constraint. For instance, in Figure 4, $b_4^2 \in \mathcal{B}_3^2$ and $b_2^3 \in \mathcal{B}_4^3$ are scheduled on the last machine. They are rescheduled to machines 1 and 3. Job 2 does not satisfy the synchronization constraint and the queue $S_1$ on machine 1, the first machine allocated to job 2, is reversed. Thus we have Lemma 5.

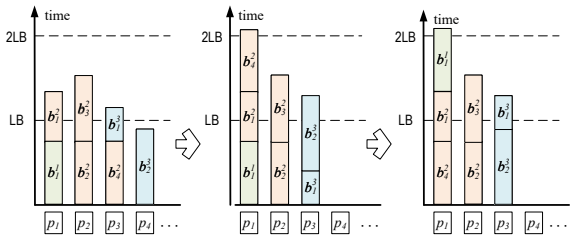



Fig. 4. The first stage of the refilling algorithm

**Lemma 5.** *Refilling after the first stage satisfies the synchronization constraint with* $C_{\max} \leq 3\texttt{LB}$.

*Proof.* With Lemma 4, $C_{\max}$ is lower than 2LB at first. After the first stage, the finishing time $C_j$ of any machine $j$ refilled with $\mathcal{B}^i_{i_{e_i}}$ is lower than 3LB as the processing time of $\mathcal{B}^i_{i_{e_i}}$ is lower than LB. As for one job against the synchronization constraint, after the first stage, all related machines $\mathcal{M}^i \backslash \{i_{e_i}\}$ start processing at time 0 except the first allocated machine $i_1$ because there may be other job(s) *completely scheduled* on machine $i_1$ before the current job $i$. The reverse operation satisfies the synchronization of the job $i$ without violating that of other job(s). Then the proposition holds. □

*Switching Rules*: First, reschedule all subjobs on any machine $j$ satisfying $C_j < \texttt{LB}$ to other machines. This is to better utilize machines. Only three cases have to deal with as follow.

- In the first case, subjobs of the same job on one machine with $C_j < \texttt{LB}$ are rescheduled. There are other subjobs of the same job on the next machine, to which the subjobs are rescheduled. In Figure 5, $b_1^3$ and $b_4^3$ on machine 2 are rescheduled to the end of $b_3^3$ on machine 3.
- In the second case, subjobs of different jobs on one machine with $C_j < \texttt{LB}$ are rescheduled. There are other

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on September 08,2023 at 08:16:33 UTC from IEEE Xplore. Restrictions apply.

subjobs on the next machine sharing the same job with the first subjob, to which all subjobs are rescheduled, enabling each job to process continuously. In Figure 6, $b_1^4$ and $b_3^3$ are rescheduled to the end of $b_3^4$ on machine 4.

- In the third case, subjobs of job(s) allocated to a single machine with $C_j < \texttt{LB}$ are rescheduled. There are no subjob of any same job on the next machine. Then the subjobs are rescheduled to machine 1. In Figure 7, $b_1^3$ is rescheduled to the end of $b_1^1$ on machine 1.

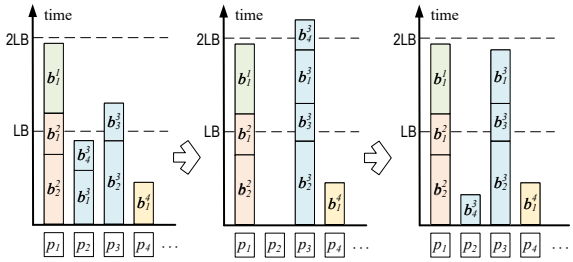



Fig. 5. Case 1 of the second stage of the refilling algorithm

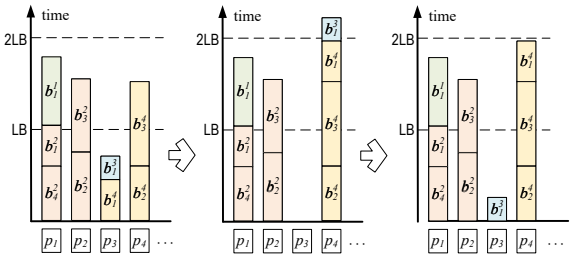



Fig. 6. Case 2 of the second stage of the refilling algorithm

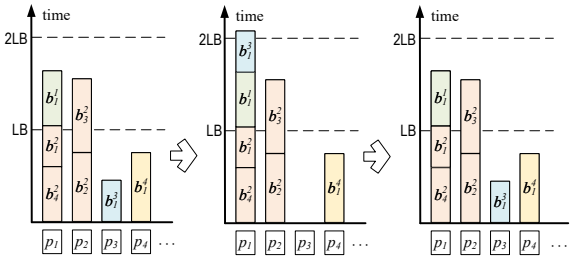



Fig. 7. Case 3 of the second stage of the refilling algorithm

After the first step of switching, $C_j$ for any machine $j$ with jobs to process is greater than LB. Until now, the operations above in refilling and switching do not involve new machines for jobs and do not violate the synchronization constraint finally. Then Lemmas 6 and 7 are given and proved as follow.

**Lemma 6.** *Once there is a machine with $C_j \geq 2\texttt{LB}$, there must be a machine idle.*

*Proof.* Assume that there are $x$ machines with $C_j \geq 2\texttt{LB}$, $y$ machines $\texttt{LB} \leq C_j < 2\texttt{LB}$, $z$ idle machines. Then we have:

$$\begin{cases} x + y + z = m, \\ 2x + y \leq m, \end{cases} \Rightarrow z \geq x, \tag{5}$$

where the first equation in Equation (5) represents the total number of the machines should be $m$, while the second one represents $\sum C_j$ of the two types of machines is no greater than $m\texttt{LB}$. Thus we have the conclusion. □

**Lemma 7.** *For machine $j$ with $C_j > 2\texttt{LB}$, the subjob queue $S_j$ can always be split into two, one with $\texttt{LB} \leq C'_j \leq 2\texttt{LB}$, and rescheduled on two machines satisfying synchronization.*

*Proof.* We first prove that a sequential processed job $i$ with $tp^i \geq \texttt{LB}$ cannot share the same machine with a parallel processed job $i'$ after SA and refilling. After SA, such a situation may exist only when machine $j$ is machine $i_{e_i}$ of some job $i$. However, refilling removes $\mathcal{B}^i_{i_{e_i}}$ away, while switching does not remove the sequential processed job away or bring other parallel processed job in with switching rules. As for other sequential processed jobs with $tp^i \geq \texttt{LB}$ merged from parallel processed jobs due to refilling and switching, they have $t^i = 0$, which means no other jobs including parallel processed ones start before them. Therefore the proposition holds. Now that the processing time of the largest subjob is at most LB, there must be at least one gap at LB to 2LB. Then the subjob queue $S_j$ can be split at the gap. Only two cases, gaps within subjobs of different jobs or the same job, exist.

- First, search for a gap satisfying the first case. The latter part of the queue $S_j$ can be directly rescheduled to an idle machine, while the former part remains unchanged. No synchronization constraint of any job is violated.
- If no gap satisfying the first case, we consider the second case. There is a job $i$ with $tp^i_j > \texttt{LB}$ covering the whole schedule of $[\texttt{LB}, 2\texttt{LB}]$. With the above proposition, either job $i$ is a parallel processed job with $t^i = 0$ or no parallel processed job starts before job $i$ on machine $j$. The latter part of the queue $S_j$ can be directly rescheduled to an idle machine, while the former part reverses the sequence on current machine $j$. Synchronization constraint is not violated in this two cases.

Therefore, Lemma 7 is proved. An instance is given in Figure 8. The gap between $b_1^2$ and $b_2^2$ is within LB and 2LB. Splitting here produces two queues with $C_j, C'_j < 2\texttt{LB}$. □

Second, continue rescheduling subjobs on machines with $C_j > 2\texttt{LB}$ until there is no $C_j > 2\texttt{LB}$ following Lemmas 6 and 7. The rescheduling on previous three cases is presented.

- In the first case, subjobs on machine 3 are split and $b_4^3$ is rescheduled to machine 2 (Figure 5).
- In the second case, subjobs on machine 4 are split and $b_1^3$ is rescheduled to machine 3 (Figure 6).
- In the third case, subjobs on machine 1 are split and $b_1^3$ is rescheduled to machine 3 (Figure 7).

**Lemma 8.** *Refilling satisfies the synchronization constraint with 2-approximation, which is proved to be a tight ratio.*

*Proof.* LB is tight as an OPT with $m$ identical jobs and $m$ machines, while 2LB is tight as an OPT with $m + 1$ identical indivisible jobs and $m$ machines ($m \to \infty$). □

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on September 08,2023 at 08:16:33 UTC from IEEE Xplore. Restrictions apply.

### D. Scheduling with Parallel Slowdown

In this subsection, the influence of parallel slowdown is discussed. According to Lemma 3, the parallel processing can be divided into two cases by $e_i$, processing with $e_i \in [1, \frac{\delta+1}{2\delta}]$ machines and processing with $e_i \in (\frac{\delta+1}{2\delta}, \frac{\delta+1}{\delta})$ machines.

In the first case, we have Lemma 9.

**Lemma 9.** *The parallel slowdown contributes to the increase of $C_{\max}$ within 2 times when $e_i \in [1, \frac{\delta+1}{2\delta}]$.*

*Proof.* When the parallel number $e_i$ increases, the slowdown coefficient $g(e_i)$ decreases. When $e_i = \frac{\delta+1}{2\delta}$, we have:

$$g(e_i) = 1 - \delta(\frac{\delta+1}{2\delta} - 1) = \frac{1+\delta}{2} > \frac{1}{2}, \tag{6}$$

which implies that no machine $j$ will slow down by more than half when processing any job $i$. Thus $C_{\max}$ will only double at most compared with before. □

In the second case, we propose Algorithm 3 to convert the case to the first one. As for a job $i$ of with $e_i \in (\frac{\delta+1}{2\delta}, \frac{\delta+1}{\delta})$, we reschedule subjobs on half of the machines with lower loads of job $i$ to those with higher loads one to one, processed after those on the machines before. Then we have Lemma 10.

**Algorithm 3:** Immigration Algorithm

**Input:** Job set $\mathcal{J}$, machine set $\mathcal{M}^i$, $e_i$ for each job $i$, subjobs $\mathcal{B}^i$ for each job $i$, the scheduling queue on each machine $j$: $S_1, S_2, \cdots, S_m$
**Output:** The new scheduling queues $S'_1, S'_2, \cdots, S'_m$

**foreach** *machine* $j$ **do**
  Initialize $S'_j \leftarrow S_j$;
**foreach** *job* $i$ **do**
  **if** $e_i > \frac{\delta+1}{2\delta}$ **then**
    Sort $\mathcal{M}^i$ in non-increasing order of $tp^i_j$;
    Split $\mathcal{M}^i$ to first half $\mathcal{M}^i$ and second half $\mathcal{M}^{i'}$;
    **foreach** *machine* $i_{j'} \in \mathcal{M}^{i'}$ **do**
      $S'_{i_j} \leftarrow S'_{i_j} \cup \mathcal{B}^i_{i'_j}$ ∀ machine $i_j \in \mathcal{M}^i$;
      $S'_{i'_j} \leftarrow S'_{i'_j} \backslash \mathcal{B}^i_{i'_j}$, $\mathcal{M}^i \leftarrow \mathcal{M}^i \backslash \{i_j\}$;

**Lemma 10.** *The parallel slowdown contributes to the increase of $C_{\max}$ within 4 times when $e_i \in (\frac{\delta+1}{2\delta}, \frac{\delta+1}{\delta})$.*

*Proof.* The converting from the second case to the first case increases $C_j$ for each retained machine $j$ by no more than 2 times because the new added loads $\mathcal{B}^i_{j'}$ are lower than the original loads $\mathcal{B}^i_j$. Therefore, the increase of $C_{\max}$ is lower than 2 times. With Lemma 9, the increase of $C_{\max}$ is lower than 4 times after considering parallel slowdown. □

Combining Lemma 8 with Lemma 10, we obtain our main result for scheduling on identical machine.

**Theorem 2.** *For the $P|sync,slow|C_{max}$ problem, SARI algorithm can guarantee a constant approximation ratio of* 8.

## VI. Scheduling on Uniform Machines

In this section, we consider scheduling on uniform machines. Our key technique, *wrapping*, is to reduce the problem on uniform machines to the one on identical machines so that our previous techniques for identical machines can be applied.

### A. Wrapping: Machine Classification

Due to the speed variance of uniform machines, wrapping classifies machines with similar speeds into $q_m$ bundles $\{B_1, \cdots, B_q, \cdots, B_{q_m}\}$ so that for each bundle $B_q$, we have:

$$2^{q-1} \leq s_j < 2^q, \quad \forall j \in B_q\,. \tag{7}$$

The size of each bundle $B_q$ is denoted by $x_q$.

**Algorithm 4:** Bundle Partitioning Algorithm

**Input:** Machine set $\mathcal{M}$, processing speed $s_j$ for each $j$
**Output:** Bundles group $B_1, B_2, \cdots, B_{q_m-1}, B_{q_m}$

**foreach** *machine* $j$ **do**
  Initialize $B_q \leftarrow \varnothing$ for $q \in N_+$;
  Find $q \in N_+$ satisfying $2^{q-1} \leq s_j < 2^q$;
  $B_q \leftarrow B_q \cup \{j\}$;

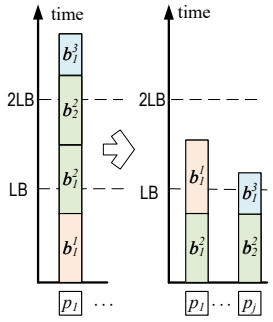



Fig. 8. Split the subjob queue

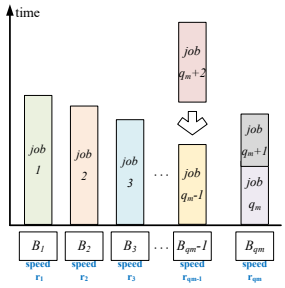



Fig. 9. Job allocation by LPT

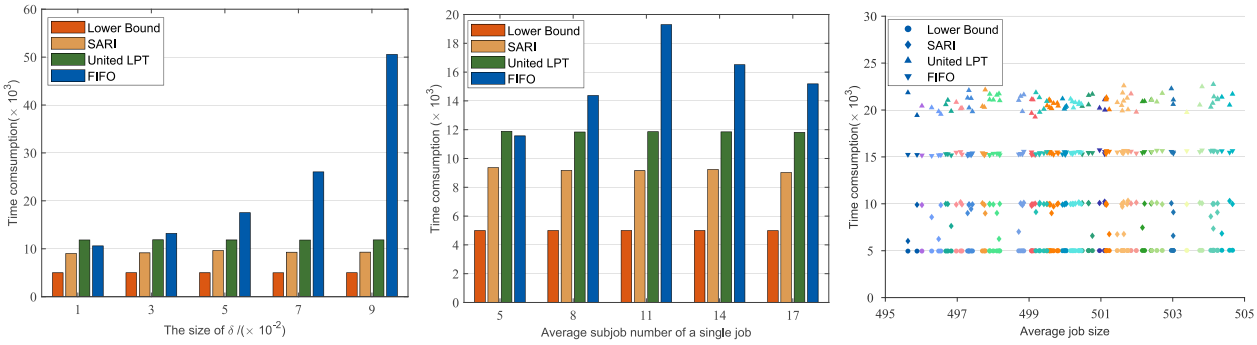
### B. Wrapping: Matching Jobs to Bundles

After partitioning the machines into $q_m$ bundles, we first match each job to a bundle within $O(\log m)$ bundles according to Lemma 1. Thus we have Lemma 11.

**Lemma 11.** *Requiring each job to be assigned to a bundle of machines loses at most $\lceil \log m \rceil$ in the approximation ratio.*

*Proof.* This proposition can be proved using the pigeonhole principle in a similar way with Theorem 1. Specifically, for each job $i$, there must exist a bundle $B_q$ such that $\sum_{j \in B_q} le^{i,*}_j \geq \frac{1}{\lceil \log m \rceil} l^i$, where $le^{i,*}_j$ represents the size of the fragment of job $i$ on machine $j$ induced by the optimal schedule. Now we move the fragments on the machines in other bundles to $B_q$ by the size of the fragment of job $i$ on each machine $j \in B_q$ by a factor of $\frac{l^i}{\sum_{j \in B_q} le^{i,*}_j}$. In this way, we obtain a feasible scheduling satisfying the constraint that each job can be allocated to the machines in a bundle, and

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on September 08,2023 at 08:16:33 UTC from IEEE Xplore. Restrictions apply.




(a) SARI: Sensitivity of $\delta$ (b) SARI: Average number of subjobs per job (c) SARI: Average job size (An instance)

Fig. 10. Experiment results for SARI on identical machines: Influence of slowdown, number of subjobs and job size

$C_{\max}$ of such a scheduling is at most $\lceil \log m \rceil$-times greater than the optimal scheduling without bundles partitioning. □

We consider machines in each bundle $B_q$ to have a unified speed $2^{q-1}$, which incurs a loss of at most 2 in the approximation ratio (formally cast in Lemma 12). Then the processing capability of a bundle can be rewritten as $r_q = 2^{q-1} \cdot x_q$.

**Lemma 12.** *Approximating the speed $s_j$ of all machines $j$ in $B_q$ to be $2^{q-1}$ increases $C_{\max}$ by a factor of at most 2.*

*Proof.* According to Equation (7), in one bundle, we have $s_j \leq 2 \cdot \min_{j,j \in B_q} s_j$, which implies that the processing time will increase by less than two times. Therefore, $C_{\max}$ of OPT increases by at most 2 times. □

To match each job to a machine bundle, UNITED-LPT is adopted and runs over machine bundles rather than singleton machines similar to Algorithm 1. Then there is Lemma 13. A scheduling instance is given in Figure 9. All the jobs are sorted first and scheduled in a non-increasing order. The first $q_m$ jobs are scheduled on $q_m$ machines, then job $q_m + 1$ and $q_m + 2$ are scheduled on the bundle with earliest $C_j$ greedily.

**Lemma 13.** *Algorithm 1 introduces 2-approximation to UWS.*

### C. Subjobs Allocation to Machines

After matching the jobs to bundles, we proceed to allocate the subjobs to machines. As $s_j$ for each machine $j$ in the bundle is unified to $2^q$, the problem of job allocation in each machine bundle is reduced to scheduling on identical machines as Section V. The $\widehat{\text{LB}}$ in each bundle $B_q$ is formulated as:

$$\widehat{\text{LB}} = \max \left\{ \frac{\sum_i l^i}{r_q}, \max_i \frac{x_q \max_k l_k^i}{r_q} \right\}, i \in B_q. \quad (8)$$

Then the scheduling problem can be solved based on the methods in Section V. Together with Theorem 2 and lemmas 11 to 13, our main result is obtained as Theorem 3.

**Theorem 3.** *For the $Q|sync,slow|C_{max}$ problem, UWS algorithm has an approximation ratio of $32\lceil \log m \rceil$.*

## VII. PERFORMANCE EVALUATION

### A. Experiments Settings

We conduct a series of numerical experiments with ablation study in the scenarios of scheduling on identical and uniform machines to analyze the performance of SARI and UWS. Each algorithm is compared with two baseline algorithms, United LPT and FIFO, and lower bound. United LPT is presented in Algorithm 1, scheduling the entire jobs greedily by $C_j$. FIFO schedules the jobs in sequence and always tries to allocate subjobs of one job to as many machines as possible with $e_i \in [1, \frac{\delta+1}{2\delta}]$. The lower bound is Equation (4) for identical machine and Equation (8) for uniform machine. Specifically for uniform machine, two lower bounds before and after bundle partitioning are computed to analyze its influence.

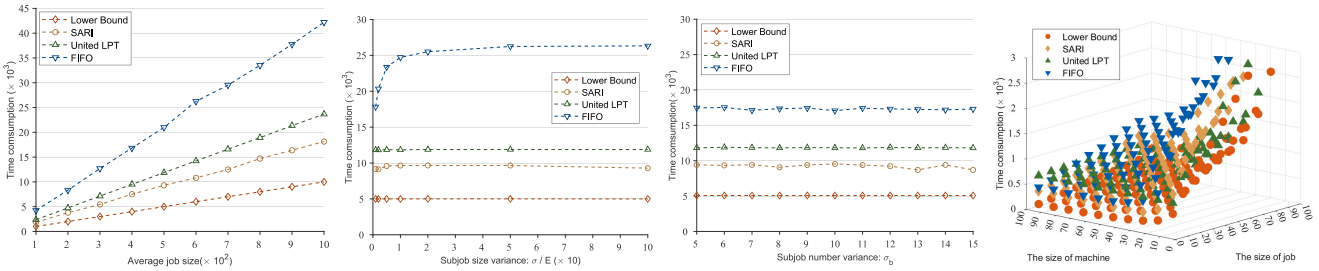
The datasets of jobs including subjobs are synthetic to better observe the performance of different methods. We generate random variables like job size, subjob size and subjob number from Gaussian distribution. Defaults of variables are set as follow if not specified: $n = 100$, $m = 10$, $\forall i$: $n_i = 10, l^i = 500$, $\delta = 0.06$; for identical machines experiments $\forall i, j : s_j^i = 1$; for uniform machines experiments $\forall i, j : E(s_j^i) = 5$. In order to avoid the influence of random factors, each group of experiments is the average outcome of 100 repeated experiments, as shown in Figure 10(c).

### B. Results and Discussions

*Results and Discussions for Identical Machines.* Figures 10 and 11 show the performance of SARI, United LPT, and FIFO on identical machines with different inputs and parameters.

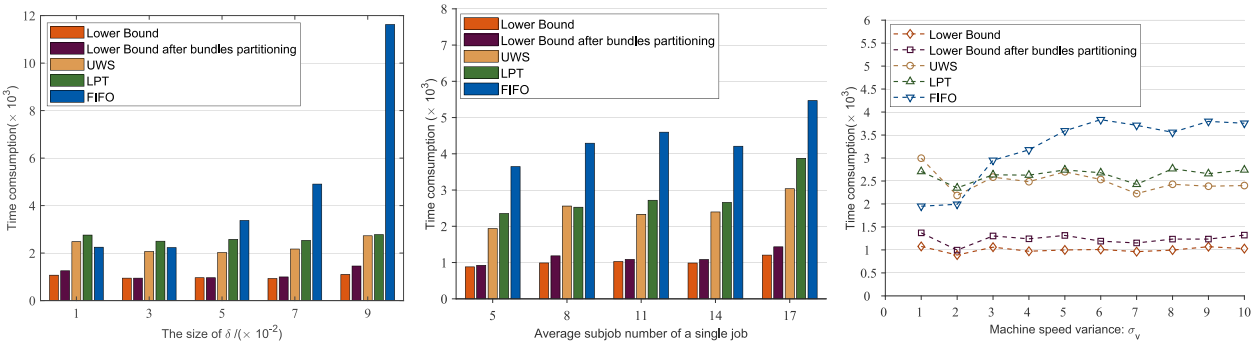
Figures 10(a) and 10(b) present the influence of the slowdown coefficient $\delta$ and the subjob number $n_i$ on makespan $C_{\max}$. Lower bound remains the same as a theoretical optimal result. United LPT also remains the same because parallel processing is not adopted. For FIFO, the increase of $\delta$ contributes to fast increase of $C_{\max}$, while the increase of $n_i$ first increases $C_{\max}$ due to the constraint of discrete subjobs, then decreases it with more small and flexible subjobs. On the contrast, SARI performs well in all situations, with small and stable $C_{\max}$ increasing caused by slowdown and decreasing with more subjobs, which implies the advantages of SARI in

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on September 08,2023 at 08:16:33 UTC from IEEE Xplore. Restrictions apply.




(a) SARI: Average job size (b) SARI: Variance of subjob size (c) SARI: Variance of subjob No. (d) SARI: Graphic Model

Fig. 11. Experiment results for SARI on identical machines: Influence of job size, subjob size and subjob number; 3-dimention relationship for the number of machines and jobs versus makespan




(a) UWS: Sensitivity of $\delta$ (b) UWS: Average number of subjobs per job (c) UWS: Variance of machine speed

Fig. 12. Experiment results for UWS on uniform machines: Influence of slowdown, subjob number and machine speed

slowdown and subjobs. Figure 10(c) gives the job size $l^i$ used in one group of experiments and related $C_{\max}$ as an instance.

Figures 11(a) to 11(c) illustrate the influence of average $l^i$, variance of $l^i$ and variance of $n_i$ on $C_{\max}$. SARI is always the best and reaches about a 2-approximation ratio compared with the lower bound. Figure 11(a) shows that both the lower bound and $C_{\max}$ of algorithms increase linearly w. r. t. the average $l^i$, which is an inevitable determining factor in this situation. Figures 11(b) and 11(c) shows that variances of $l_k^i$ and $n_i$ have almost no influence on SARI, similar to United LPT, implying that SARI can well deal with the two factors. Meanwhile, FIFO performs worse with the increase of the variance of $l_k^i$ as a large subjob can significantly affect $C_{\max}$. Figure 11(d) shows the variation of $C_{\max}$ with both $n$ and $m$ varying from 10 to 100. In general, more machines contribute to less $C_{\max}$ under the same condition and more jobs contribute to larger $C_{\max}$. United LPT performs well when $n$ is much larger than $m$, especially when the number of jobs is alike, while FIFO performs well only when $m$ is much larger than $n$ and $\delta$ is relatively small. SARI achieves good results with various inputs balancing both sequential and parallel processing.

*Results and Discussions for Uniform Machines.* Figure 12 present the performance of UWS compared to United LPT, FIFO, and lower bounds with various inputs and parameters.

LB after partitioning admits an increase less than 1.5, better than the theoretical guarantee. Figures 12(a) and 12(b) show a similar conclusion of UWS on $\delta$ and $n_i$ as SARI, which outperforms United LPT and FIFO. $C_{\max}$ of UWS increases by a small ratio along with the increase of $\delta$ and $n_i$. Figure 12(c) speculates that the performance of UWS gets better when the machine speed variance gets larger, suggesting wrapping effectively eliminates the disadvantage from the differences of $s_j$. The results of the experiment reflect that UWS has a better experimental approximation ratio around 2 than theory.

## VIII. Conclusions

In this paper, we study a new variation of uniform machine scheduling problem and develop UWS algorithm, which is a two-layer optimization design with an approximation ratio of $O(\log m)$, including *Wrapping* for the outer layer to cluster machines with similar speeds by partitioning and *Refilling* for the inner individual layer to unite subjobs of one job, combining the advantages of both sequential and parallel processing, such that the overall scheduling framework can output a sub-optimal solution. The performance is guaranteed by theoretical proofs and numerical experiments.

## Acknowledgment

This work was supported by National Key R&D Program of China [2020YFB1707900]; National Natural Science Foundation of China [62272302], Shanghai Municipal Science and Technology Major Project [2021SHZDZX0102] and Huawei Cloud [TC20220718012]. The authors would like to thank Rongpeng Chen for his contribution on this paper. Xiaofeng Gao is the corresponding author.

Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on September 08,2023 at 08:16:33 UTC from IEEE Xplore. Restrictions apply.

## REFERENCES


[1] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in *IEEE international conference on cloud computing*, 2009, pp. 626–631.

[2] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, 2015.

[3] Y. Sun, F. Lin, and H. Xu, "Multi-objective optimization of resource scheduling in fog computing using an improved nsga-ii," *Wireless Personal Communications*, vol. 102, no. 2, pp. 1369–1385, 2018.

[4] M. Pak and S. Kim, "A review of deep learning in image recognition," in *IEEE international conference on computer applications and information processing technology (CAIPT)*, 2017, pp. 1–3.

[5] S.-H. Lin, M. Paolieri, C.-F. Chou, and L. Golubchik, "A model-based approach to streamlining distributed training for asynchronous sgd," in *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2018, pp. 306–318.

[6] S. Wang, D. Li, and J. Geng, "Geryon: Accelerating distributed cnn training by network-level flow scheduling," in *IEEE Conference on Computer Communications*, 2020, pp. 1678–1687.

[7] K. Jansen, M. Maack, and A. Mäcker, "Scheduling on (un-)related machines with setup times," in *IEEE International Parallel and Distributed Processing Symposium*, 2019, pp. 145–154.

[8] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing, "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning," in *USENIX Symposium on Operating Systems Design and Implementation*, 2021.

[9] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 1997, pp. 1–34.

[10] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[11] F. N. Afrati and J. D. Ullman, "Optimizing multiway joins in a map-reduce environment," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, pp. 1282—1298, 2011.

[12] D. Fotakis, I. Milis, O. Papadigenopoulos, V. Vassalos, and G. Zois, "Scheduling mapreduce jobs on identical and unrelated processors," *Computers and Industrial Engineering*, vol. 64, pp. 754—-782, 2020.

[13] K. Aziz, D. Zaidouni, and M. Bellafkih, "Leveraging resource management for efficient performance of apache spark," *J. Big Data*, vol. 6, p. 78, 2019.

[14] X. Zhan, Y. Bao, C. Bienia, and K. Li, "Parsec3. 0: A multicore benchmark suite with network stacks and splash-2x," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 5, pp. 1–16, 2017.

[15] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal of Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.

[16] J. Aspnes, Y. Azar, A. Fiat, S. A. Plotkin, and O. Waarts, "On-line routing of virtual circuits with applications to load balancing and machine scheduling," *Journal of the ACM*, vol. 44, no. 3, pp. 486–504, 1997.

[17] K. Banerjee, "An approximate algorithm for the partitionable independent task scheduling problem," *Urbana*, vol. 51, p. 61801, 1990.

[18] K. Jansen, "A (3/2+ $\varepsilon$) approximation algorithm for scheduling moldable and non-moldable parallel tasks," in *ACM symposium on Parallelism in algorithms and architectures*, 2012, pp. 224–235.

[19] K. Jansen and F. Land, "Scheduling monotone moldable jobs in linear time," in *IEEE International Parallel and Distributed Processing Symposium*, 2018, pp. 172–181.

[20] D. S. Hochbaum and D. B. Shmoys, "A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach," *SIAM Journal on Computing*, vol. 17, no. 3, pp. 539–551, 1988.

[21] K. Jansen, "An eptas for scheduling jobs on uniform processors: using an milp relaxation with a constant number of integral variables," *SIAM Journal on Discrete Mathematics*, vol. 24, no. 2, pp. 457–485, 2010.

[22] L. Teylo, L. Arantes, P. Sens, and L. M. de A. Drummond, "A hibernation aware dynamic scheduler for cloud environments," in *International Conference on Parallel Processing*, 2019, pp. 24:1–24:10.

[23] K. Jansen and D. Trystram, "Scheduling parallel jobs on heterogeneous platforms," *Electronic Notes in Discrete Mathematics*, vol. 55, pp. 9–12, 2016.

[24] B. Berg, R. Vesilo, and M. Harchol-Balter, "hesrpt: Parallel scheduling to minimize mean slowdown," *Performance Evaluation*, vol. 144, p. 102147, 2020.

[25] J. R. Correa, A. Marchetti-Spaccamela, J. Matuschke, L. Stougie, O. Svensson, V. Verdugo, and J. Verschae, "Strong LP formulations for scheduling splittable jobs on unrelated machines," *Math. Program.*, vol. 154, no. 1-2, pp. 305–328, 2015.

[26] 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.

[27] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Annals of discrete mathematics*, 1979, vol. 5, pp. 287–326.

[28] T. F. Gonzalez, O. H. Ibarra, and S. Sahni, "Bounds for LPT schedules on uniform processors," *SIAM Journal of Computing*, vol. 6, no. 1, pp. 155–166, 1977.

[29] F. A. Chudak and D. B. Shmoys, "Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds," *Journal of Algorithms*, vol. 30, no. 2, pp. 323–343, 1999.

[30] S. Li, "Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations," *SIAM J. Comput.*, vol. 49, no. 4, 2020.

[31] S. Davies, J. Kulkarni, T. Rothvoss, J. Tarnawski, and Y. Zhang, "Scheduling with communication delays via LP hierarchies and clustering II: weighted completion times on related machines," in *ACM-SIAM Symposium on Discrete Algorithms*, 2021, pp. 2958–2977.

[32] B. Maiti, R. Rajaraman, D. Stalfa, Z. Svitkina, and A. Vijayaraghavan, "Scheduling precedence-constrained jobs on related machines with communication delay," in *IEEE Annual Symposium on Foundations of Computer Science*, 2020, pp. 834–845.

[33] M. A. Deppert and K. Jansen, "Near-linear approximation algorithms for scheduling problems with batch setup times," in *ACM Symposium on Parallelism in Algorithms and Architectures*, 2019, pp. 155–164.

[34] S. Khot and A. K. Ponnuswami, "Better inapproximability results for maxclique, chromatic number and min-3lin-deletion," in *International Colloquium on Automata, Languages and Programming*, vol. 4051, 2006, pp. 226–237.

[35] M. A. Deppert and K. Jansen, "Near-linear approximation algorithms for scheduling problems with batch setup times," in *ACM on Symposium on Parallelism in Algorithms and Architectures*, 2019, pp. 155–164.


Authorized licensed use limited to: Shanghai Jiaotong University. Downloaded on September 08,2023 at 08:16:33 UTC from IEEE Xplore. Restrictions apply.