Mitigating Service Variability in MapReduce Clusters via Task Cloning: A Competitive Analysis

Huanle Xu, Wing Cheong Lau, *Senior Member, IEEE*
Zhibo Yang, *Student Member, IEEE,* Gustavo de Veciana, *Fellow, IEEE,* Hanxu Hou

Abstract—Measurement traces from real-world production environment show that the execution time of tasks within a MapReduce job varies widely due to the variability in machine service capacity. This variability issue makes efficient job scheduling over large-scale MapReduce clusters extremely challenging. To tackle this problem, we adopt the task cloning approach to mitigate the effect of machine variability and design corresponding scheduling algorithms so as to minimize the overall job flowtime in different scenarios. For offline scheduling where all jobs arrive at the same time, we design an $O(1)$-competitive algorithm, which gives priorities to jobs with small effective workload. We then extend this offline algorithm to yield the so-called Smallest Remaining Effective Workload based β-fraction Sharing plus Cloning algorithm (SREW+C(β)) for the online case. We also show that SREW+C(β) is $(1+2\beta+\epsilon)$-speed $O(\frac{1}{\beta\epsilon})$-competitive with respect to the sum of job flowtime within a cluster. We demonstrate via trace-driven simulations that SREW+C(β) can significantly reduce the overall job flowtime by cutting down the elapsed time of small jobs substantially. In particular, SREW+C(β) reduces the total job flowtime by 14%, 10% and 11% respectively when comparing to Mantri, Dolly and Grass.

Index Terms—MapReduce, job Scheduling, cloning, job flowtime, competitive performance ratio

1 Introduction

MapReduce [10] and its open-source realization via Hadoop [1] have emerged as the defacto framework to support large-scale parallel/distributed processing and data analytics. Under the MapReduce framework, the overall computation of a job is decomposed into two separate phases, namely, the Map phase and the Reduce phase. Within each phase, many relatively small tasks are executed in parallel across a large number of machines within the MapReduce cluster. The MapReduce computational model also requires that the Reduce phase of a job cannot begin until all the tasks within its Map phase have been completed. A key feature of catalyzing the widespread adoption of MapReduce framework is the ability to transparently deal with the challenges of executing these tasks in a distributed setting. One of such fundamental challenges is the machine service variability caused by partial or intermittent machine failures, localized resource bottleneck(s) or congested network connections [9], [37]. It has been pointed out in [15] that the service/failure model of most parallel and distributed computing systems follows the Gamma distribution. Such a machine variability can easily lead to stragglers, i.e., tasks that are unfortunately assigned to machines suffering from an extremely low processing capacity. Measurement traces from the real-world production environment [5] indicate that stragglers lead to a large variation in execution times among tasks in the same job phase and delay job completion substantially.

The dominant technique to mitigate variability in task execution times is via redundant execution: a strategy which preventively or reactively handles long running tasks by automatically launching redundant copies of a task on alternative machines. With redundancy, it is expected that one of the copies of the same task may complete quickly to avoid long task/job completion times. There are two main classes of redundancy approaches proposed in the literature, namely, the Cloning approach [3] and the Straggler-Detection-based one [1], [5], [9], [18], [37]. Under the Cloning approach, extra copies of a task are scheduled with the initial task in parallel and the one which finishes first is used for the subsequent computation. For the Straggler-Detection based approach, the progress of each task is monitored by the system and backup copies are launched when a straggler is detected. While existing redundant execution strategies have been proven to be efficient via practical implementations, most of them are designed based on rudimentary heuristics. There is a lack of systematic analysis or reference study on how well or suboptimal these schemes are when compared to the theoretical limits of various performance metrics such as the job flowtime, which is a common metric to be optimized in the scheduling literature, e.g., [7], [8], [22], [24], [30], [35], [38] and in production clusters, e.g., [3]–[5].

To take a more systematic approach for the design of redundancy schemes, previous works in the literature including [32]–[34] have proposed several optimization-based schemes. However, in [32] and [34], the precedence

- *Huanle Xu is with the College of Computer Science and Technology, Dongguan University of Technology. E-mail: xuhl@dgut.edu.cn.*
- *Wing Cheong Lau and Zhibo Yang are with the Department of Information Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. E-mail: {wclau,yz014}@ie.cuhk.edu.hk.*
- *Gustavo de Veciana is with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, US. E-mail: gustavo@ece.utexas.edu.*
- *Hanxu Hou is with the College of Electronic Engineering, Dongguan University of Technology. E-mail: houhx@dgut.edu.cn.*

Part of this work has been presented in IEEE ICDCS 2015.

Algorithm	minimize	combine job scheduling and task redundancy	redundancy approach	machine speed	competitive ratio
SREW(β)	$\sum_{j=1}^{N} f_j$	$\surd$	cloning	$1+2\beta+\epsilon$	$O(\frac{1}{\beta\epsilon})$
SCA [34]	$\sum_{j=1}^{N}(f_j+\gamma cost_j)$	$\surd$	cloning	Not Applicable	No Bound
ESE [34]	$\sum_{j=1}^{N}(f_j+\gamma cost_j)$	$\surd$	speculative execution		
Mantri [5]	Not Applicable	×	speculative execution		
Dolly [3]		×	cloning		
Grass [4]		×	speculative execution		
Hopper [28]		$\surd$	speculative execution		
LAPS [11]	$\sum_{j=1}^{N} f_j$	Not Applicable	no	$1+\beta+\epsilon$	$O(\frac{1}{\beta\epsilon})$
WLAPS+E [12]	$\sum_{j=1}^{N}(f_j+\gamma cost_j)$		no	$1+\epsilon$	$O(\frac{1}{\epsilon^2})$
WLAPS [14]	$\sum_{j=1}^{N} f_j^2$		no	$2+\epsilon$	$O(\frac{1}{\beta\epsilon})$
Intermediate -SRPT [16]	$\sum_{j=1}^{N} f_j$		no	1	$O(1)\cdot 4^{1/(1-\alpha)}\cdot\log P$

Fig. 1. Comparison between SREW(β) and other different algorithms where f_j denotes the job flowtime and $cost_j$ denotes the job computation cost. The last four algorithms do not adopt task redundancy and consider each job consists of only one task. Moreover, the competitive ratio for the last algorithm, i.e., Intermediate-SRPT contains two factors, $\log P$ and $4^{1/(1-\alpha)}$, where P is the ratio between the largest and the smallest job size and α is the parallelizability level.

constraints between the two phases in the MapReduce framework are ignored and the knowledge of the complete distribution of task execution time is needed a priori. [33] has overcome these limitations, but it does not directly take into consideration whether the variability in task execution time is due to the variability in task sizes or in machine processing speed. In fact, if there is no variability in machine service capacity, scheduling multiple copies of the same task may not help at all and redundancy is a waste of resource.

To tackle the aforementioned problems, in this paper, we explicitly model the precedence between the Map and Reduce phase. More importantly, we build a time-slotted stochastic framework to explore the variability in machine service capacity using task cloning. We also study the relationship between the variation in task execution time and machine service capacity. Our primary goal is to design scheduling algorithms which can adapt the number of cloned copies so as to minimize the overall job flowtime.

Our objective yields a stochastic scheduling problem which turns out to be more difficult than the NP-Hard scheduling problem presented in [38]. Therefore, we resort to the use of approximate algorithms in both the offline and online setting. Under our proposed algorithms, the scheduler only needs to know the first and second moment of the task execution time in advance. For the analysis of the online algorithm, we assume task preemption and resource augmentation [19], which are necessary to derive non-trivial lower bounds for the parallel scheduling on multiple machines. To summarize, this paper has made the following technical contributions:

- To the best of our knowledge, this paper is the first one to model the impact of service variability of computing nodes within a cluster and cast the dynamic scheduling problem into a stochastic optimization problem that aims at finding a cloning scheme to minimize the overall job flowtime (Section 2).
- This paper investigates the design of redundancy algorithms that achieve competitive performance bounds by taking an analytical approach with mathematical vigor. Under the transient scheduling setting [2], we design the Smallest Effective Workload (SEW) algorithm which gives higher priorities to jobs with the smallest effective workload. We prove in Section 3 that SEW is $O(1)$-competitive with respect to the expected overall job flowtime. To support online job scheduling, we extend the SEW algorithm to yield the SREW+C(β) algorithm in Section 4. By constructing a novel potential function, we prove that SREW+C(β) is $(1+2\beta+\epsilon)$-speed $O(\frac{1}{\beta\epsilon})$-competitive in terms of overall job flowtime.
- We conduct extensive trace-driven simulations to demonstrate that SREW+C(β) can reduce the average job flowtime by more than 10% when comparing to state-of-the-art scheduling schemes(Section 5).

Fig. 1 summarizes the contributions of SREW(β) by comparing its properties to other scheduling algorithms.

2 System Model and Problem Formulation

Consider a MapReduce Cluster which consists of M machines indexed from 1 to M. A machine could represent a processor, a CPU core or a virtual machine. Consider a time-slotted system where job j arrives at the cluster at the end of a time slot, denoted by a_j. The job arrival process, $(a_1, a_2, \cdots, a_N)$ is an arbitrary deterministic time sequence. Upon arrival, job j joins a global queue managed by a scheduler, waiting to be scheduled at the beginning of some time slot. Moreover, job j consists of m_j map tasks and r_j reduce tasks where the task sets are denoted by $\boldsymbol{M}_j = \{T_j^{m,1}, T_j^{m,2}, \cdots, T_j^{m,m_j}\}$ and $\boldsymbol{R}_j = \{T_j^{r,1}, T_j^{r,2}, \cdots, T_j^{r,r_j}\}$ respectively.

Each machine can only hold one map or reduce task at any time. We model the service process on each machine as follows: the service capacity of machines in a time slot is assumed to be identically distributed random variables. To be specific, let $X_i(t)$ denote the amount of service delivered in slot t by machine i. The service capacity of a machine has a unit mean with variance σ^2 and a peak value of Δ. Thus, for all $i \in \{1, 2, \cdots, M\}$ and $t \geq 1$, we have $X_i(t) \leq \Delta$

TABLE 1
The notations of the scheduling parameters

Notations	Corresponding meaning
$T_j^{z,i}$	the ith map/ reduce task of job j with $T_j^{m,i}$ for map task; $T_j^{r,i}$ for reduce task
a_j	arrival time of job j
Φ_j	the time when job j completes its work
Ψ_j	the flowtime of job j
$X_i(t)$	the service capacity of machine i in time slot t
Δ	the peak value of machine service capacity
p_j^z	the amount of work for the map/ reduce task in job j
E_j^z	the mean of the execution time for a task in job j
σ_j^z	the SD of the execution time for a task in job j
M	total number of machines in the cluster
$\mathcal{C}_j^{z,i}$	the time when the ith map/ reduce task in job j completes
$S_j^{z,i,k}$	the start time of the kth cloned copy for task $T_j^{z,i}$
$\Theta_j^{z,i,k}$	the execution time of the kth cloned copy for task $T_j^{z,i}$ without preemption
$P_j^{z,i}$	service time of the ith task in job j without preemption and cloning
$d_j^{z,i}$	total number of copies made for task $T_j^{z,i}$ without preemption
$d_j^{z,i}(t)$	the number of copies made for task $T_j^{z,i}$ with preemption in time slot t
$V_j^{z,i}(t)$	the set of machines which are running a copy of task $T_j^{z,i}$ in time slot t
$W_j^{z,i}(t)$	the amount of work processed in slot t for task $T_j^{z,i}$
$W_j(t)$	the total amount of work processed in slot t for job j
Υ_j	the volume of effective workload of job j
$\Upsilon_j(t)$	the remaining effective workload of job j in slot t
$\psi^s(t)$	the set of active jobs in the cluster in time slot t

almost surely and $\mathbb{E}[X_i(t)] = 1$, $\text{Var}[X_i(t)] = \sigma^2$. Table 1 summarizes all the notations in this model.

2.1 Job service process

For ease of presentation, throughout this paper, we use $z \in \{m, r\}$ to denote the map- or reduce-related statements for all the tasks: when z is used, it is set to either m or r. A task in job j requires p_j^z time slots to complete when processed on a machine at *unit* speed. For simplicity, we consider each task completes its work at the end of a time slot.

We adopt task cloning to tackle machine variability so as to reduce task/job execution time. We generalize the task cloning approach in [3] to yield a more flexible version where the cloned copies within a task can be scheduled at different times. Without loss of generality, consider the kth copy of task $T_j^{z,i}$ is run on machine l. Without preemption, the execution time of the kth copy, $\Theta_j^{z,i,k}$ satisfies:

$$\Theta_j^{z,i,k} = \min\Big\{\tau : \sum_{t=S_j^{z,i,k}}^{S_j^{z,i,k}+\tau-1} X_l(t) \geq p_j^z\Big\}, \tag{1}$$

where $S_j^{z,i,k}$ denotes the time when the kth copy is launched. Observe from (1) that, the random variable $\Theta_j^{z,i,k}$ defines a stopping time. Applying Wald's Equation [21], it follows that:

$$\mathbb{E}[\Theta_j^{z,i,k}]\mathbb{E}[X_l(t)] = p_j^z. \tag{2}$$

Let E_j^z denote the mean of the execution time for task $T_j^{z,i}$ without cloning. According to (2), for all $z \in \{m, r\}$ and $i \in \{1, 2, \cdots, z_j\}$, we have:

$$\mathbb{E}[\Theta_j^{z,i,k}] = E_j^z = p_j^z. \tag{3}$$

We explore the relationship between the variance of $\Theta_j^{z,i,k}$ and σ^2 by applying the result from [20]. Specifically, when $p_j^z \to \infty$, we have:

$$\text{Var}[\Theta_j^{z,i,k}] = (\sigma_j^z)^2 = (p_j^z + \frac{1}{2})\sigma^2 + O(1),$$

where σ_j^z denotes the standard deviation of the task execution time without cloning.

Assumption 1. *For all $j \in \{1, 2, \cdots, N\}$ and $z \in \{m, r\}$, E_j^z and σ_j^z are known to the scheduler a priori when job j arrives at the cluster.*

Denote by $\mathcal{C}_j^{z,i}$ the completion time of task $T_j^{z,i}$. Since a job completes when all its reduce tasks finish, the completion time of job j, denoted by Φ_j, is given by:

$$\Phi_j = \max_{i \in \{1,2,\cdots,r_j\}} \mathcal{C}_j^{r,i}. \tag{4}$$

The flowtime of job j is denoted by $\Psi_j = \Phi_j - a_j$.

2.2 Problem formulation without preemption

In this section, we will formulate an optimization problem for the case where preemption is not used. When there are $d_j^{z,i}$ copies launched for task $T_j^{z,i}$, the task completion time $\mathcal{C}_j^{z,i}$ satisfies:

$$\mathcal{C}_j^{z,i} = \min_{k \in \{1,2,\cdots,d_j^{z,i}\}} (S_j^{z,i,k} + \Theta_j^{z,i,k}). \tag{5}$$

Denote by $\boldsymbol{S}_j^{z,i} = (S_j^{z,i,k} | k = 1, 2, \cdots, d_j^{z,i})$ the vector of start time of the $d_j^{z,i}$ copies of the i-th (map or reduce) task of job j as decided by the scheduler. $\boldsymbol{\alpha}_j^z = \left(d_j^{z,i}, \boldsymbol{S}_j^{z,i} | i = 1, 2, \cdots, z_j\right)$ therefore characterizes the number of cloned copies and the scheduling time of each copy of all tasks in job j. Let $\boldsymbol{\alpha}^z = (\boldsymbol{\alpha}_j^z | j = 1, 2, \cdots, N)$ be the scheduling trajectory of all jobs. The problem of minimizing the overall job flowtime, i.e., $\sum_{j=1}^N \Psi_j$, by determining $(\boldsymbol{\alpha}^m, \boldsymbol{\alpha}^r)$, can be formulated as the following optimization problem (P1) :

$$\min_{(\boldsymbol{\alpha}^m, \boldsymbol{\alpha}^r)} \sum_{j=1}^N \mathbb{E}[\Phi_j - a_j] \tag{P1}$$

$$\text{s.t. } S_j^{m,i,k} \geq a_j, \ \forall j, \ 1 \leq i \leq m_j, \ 1 \leq k \leq d_j^{m,i},$$

$$S_j^{r,i,k} \geq \max_{1 \leq l \leq m_j} \mathcal{C}_j^{m,l}, \ \forall j, \ 1 \leq i \leq r_j, \ 1 \leq k \leq d_j^{r,i},$$

$$\sum_{j=1}^N \sum_{i=1}^{m_j} \sum_{k=1}^{d_j^{m,i}} \mathbb{1}(t - \Theta_j^{m,i,k} < S_j^{m,i,k} \leq t) = M(t), \quad \forall t,$$

$$\sum_{j=1}^N \sum_{i=1}^{r_j} \sum_{k=1}^{d_j^{r,i}} \mathbb{1}(t - \Theta_j^{r,i,k} < S_j^{r,i,k} \leq t) = R(t), \quad \forall t,$$

$$M(t) + R(t) \leq M, \quad \forall t,$$

where $\Theta_j^{c,i,k}$, Φ_j and $\mathcal{C}_j^{c,i,k}$ are given by (1), (4) and (5) respectively. Here, $\mathbb{1}(A)$ denotes the indicator function that takes value 1 if A is true and 0 otherwise. The first constraint corresponds to the fact that a map task as well as its cloned copies can only start after the job arrives. The second constraint states that a reduce task can start processing only after all map tasks of the same job have finished. The subsequent equations specify the cluster capacity constraint,

namely, the total number of tasks (including their copies) executed at any time should be no more than M.

Remark 1. *When task cloning is not used and there is no variation in machine processing capacity, i.e., the task execution time is a deterministic number rather than a random variable, the scheduling problem in our model reduces to the optimization problem presented in [38], which has been proven to be NP-Hard even for the special offline case where all the jobs arrive the cluster at the same time. The stochastic optimization problem P1 is therefore NP-Hard and we resort to the use of approximation algorithms to tackle it.*

Worse still, previous work (e.g., [6]) shows that it is impossible to achieve any "reasonable" approximation for the overall job flowtime in the non-preemptive case of online scheduling with single-task jobs over multiple machines. Therefore, we shall design scheduling algorithms which allow task preemption so as to achieve competitive performance ratios of small values. This also enables us to derive non-trivial bounds on the performance of the corresponding algorithms.

2.3 Task Service Process and Problem formulation with preemption

In a cluster allowing preemption, the scheduler can preempt a running task and subsequently resume it. To make use of the already completed work and take full advantage of cloning, at the end of each time slot, we also use opportunistic checkpointing [26], [29], which is a valuable tool in practical parallel systems to preempt its active jobs. Upon checkpointing, the state of the redundant copy which has made the most progress is propagated and cloned to other copies. Therefore, all the redundant copies can be in the same state and continue their execution based on the new state after checkpointing.

Let $V_j^{z,i}(t)$ be the set of machines which are running a copy of $T_j^{z,i}$ at time slot t and let $d_j^{z,i}(t)$ be the cardinality of $V_j^{z,i}(t)$, i.e., $|V_j^{z,i}(t)|$. Moreover, denote by $W_j^{z,i}(t)$ the volume of work processed at time slot t. With checkpointing, $W_j^{z,i}(t)$ is given by:

$$W_j^{z,i}(t) = \max_{k \in V_j^{z,i}(t)} X_k(t), \tag{6}$$

and the task completion time $\mathcal{C}_j^{z,i}$, satisfies:

$$\sum_{t=a_j+1}^{\mathcal{C}_j^{z,i}} W_j^{z,i}(t) = p_j^z. \tag{7}$$

Further define $\boldsymbol{V}_j^z(t) = (V_j^{z,i}(t) | i = 1, 2, \cdots, z_j)$ and $\boldsymbol{V}^z(t) = (\boldsymbol{V}_j^z(t) | j = 1, 2, \cdots, N)$ to characterize the scheduling decisions of all jobs at time t. Let $\boldsymbol{\xi}^z = (\boldsymbol{V}^z(t) | t \in \mathbb{N}^+)$ be the scheduling trajectory of all jobs over time. In this paper, we aim to minimize the overall job flowtime by determining $(\boldsymbol{\xi}^m, \boldsymbol{\xi}^r)$, which yields the optimization problem $P2$, as shown below:

$$\min_{(\boldsymbol{\xi}^m, \boldsymbol{\xi}^r)} \sum_{j=1}^{N} \mathbb{E}[\Phi_j - a_j] \tag{P2}$$

$$\text{s.t. } V_{j_1}^{z_1,i_1}(t) \bigcap V_{j_2}^{z_2,i_2}(t) = \emptyset, \ \forall (j_1, i_1, z_1) \neq (j_2, i_2, z_2),$$

$$\bigcup_{1 \leq j \leq N, 1 \leq i \leq z_j, z \in \{m,r\}} V_j^{z,i}(t) \subset \{1, 2, \cdots, M\}, \ \forall t \in \mathbb{N}^+,$$

$$V_j^{r,i}(t) = \emptyset, \forall 1 \leq j \leq N, 1 \leq i \leq r_j, t \leq \max_{1 \leq i \leq m_j} \mathcal{C}_j^{m,i},$$

where $W_j^{z,i}(t)$ is given by (6) and satisfies (7). The first constraint states that a machine can only hold one task at any time. The second constraint corresponds to the cluster capacity. The third constraint states that a reduce task can only start after all map tasks of the same job have finished.

2.4 Competitive Performance Analysis

In this paper, we will study both offline and online algorithms for $P2$. This involves the determination of the number of cloned copies for each task at any time slot. In particular, we shall use the following metric to evaluate the performance of our proposed offline algorithm.

Definition 1. *An algorithm is c-competitive if the algorithm's objective is within a factor of c of the optimal solution's objective.*

It has been shown in [19] that no online algorithm can achieve a constant competitive performance bound for the total flowtime of jobs sharing multiple machines even for the simple case of one-task-per-job. As such, previous work [17], [19] has adopted a resource augmentation analysis. Under such analysis, the performance of the optimal algorithm on M unit-speed machines is compared with that of the proposed algorithms on M δ-speed machines where $\delta > 1$. The following definition characterizes the competitive performance of an online algorithm using resource augmentation argument.

Definition 2. *[19] An online algorithm is δ-speed, c-competitive if the algorithm's objective is within a factor of c of the optimal solution's objective when the algorithm is given δ resource augmentation.*

In this paper, we also adopt the resource augmentation argument to bound the competitive performance of the proposed online algorithm. With resource augmentation, the service capacity in any time slot under our algorithms is scaled by δ. In other words, the value of $X_i(t)$ for all $i \in \{1, 2, \cdots, M\}$ and $t \in \mathbb{N}^+$ is δ times that under the optimal algorithm of the same variables.

3 Transient Scheduling [2]: All jobs arrive at the cluster at the same time

Before designing the online algorithm, in this section, we consider a special case of offline scheduling, namely, the so-called transient scheduling setting where all the jobs enter into the system at the same time. In particular, we consider that $a_j = 0$ for all $j \in \{1, 2, \cdots, N\}$. Although this setting is simple, the offline algorithm presented in the sequel provides good insights for us to design an online algorithm.

In this section, we assume without loss of generality that jobs have been ordered such that $(m_1 E_1^m + r_1 E_1^r) \leq (m_2 E_2^m + r_2 E_2^r) \leq \cdots \leq (m_N E_N^m + r_N E_N^r)$.

3.1 A lower bound for the optimal scheduling algorithm

Under the transient scheduling setting, we shall first give a lower bound for the overall job flowtime achieved by any deterministic scheduling algorithm with task cloning. To achieve this, we consider an optimal scheduling algorithm, A^{old} and a particular time slot t.

From (6), we have:

$$W_j^{z,i}(t) = \max_{k \in V_j^{z,i}(t)} X_k(t) \leq \Delta,$$

which implies that, a task can be processed at a rate of at most Δ. Therefore, job j requires at least $\frac{E_j^m + E_j^r}{\Delta}$ units of time to complete. As such, a first lower bound for the total job flowtime achieved by A^{old}, is given by:

$$\sum_{j=1}^{N} \mathbb{E}[\Phi_j] \geq \sum_{j=1}^{N} \frac{E_j^m + E_j^r}{\Delta}. \tag{8}$$

Following (6), the total amount of work processed at time slot t for task $T_j^{z,i}$ under A^{old}, is upper bounded by:

$$W_j^{z,i}(t) = \max_{k \in V_j^{z,i}(t)} X_k(t) \leq \sum_{k \in V_j^{z,i}(t)} X_k(t). \tag{9}$$

Thus, the total amount of work processed for job j at time slot t, $W_j(t)$, is upper bounded by:

$$W_j(t) \leq \sum_{i=1}^{m_j} \sum_{k \in V_j^{m,i}(t)} X_k(t) + \sum_{i=1}^{r_j} \sum_{k \in V_j^{r,i}(t)} X_k(t). \tag{10}$$

Summing up $W_j(t)$ over all j, it follows that, the total amount of work processed for all jobs is upper bounded by the system capacity, i.e.,

$$\sum_{j=1}^{N} W_j(t) \leq \sum_{i=1}^{M} X_i(t), \tag{11}$$

We then simulate a new system and a new scheduling algorithm A^{new} as follows: the system has one single machine whose processing capacity is $\sum_{i=1}^{M} X_i(t)$ at the t th time slot. Moreover, the t th time slot is further divided into several small time periods where the jth period has a length of $\frac{W_j(t)}{\sum_{j=1}^{N} W_j(t)}$. Under A^{new}, the jth period is fully dedicated to job j for processing at a rate of $\sum_{i=1}^{M} X_i(t)$. Based on (11), one can see that the volume of work processed in the simulated system for each job will not be smaller than that in the original system. Therefore, we can use the overall job flowtime achieved by A^{new} in the new system as a lower bound for the overall job flowtime achieved by A^{old}.

Theorem 1. *In the simulated system, the expected value of the overall job flowtime is lower bounded by:*

$$\sum_{j=1}^{N} \mathbb{E}[\Phi_j] \geq \frac{\sum_{j=1}^{N} (N+1-j)(m_j E_j^m + r_j E_j^r)}{M}.$$

Proof. Assume that the completion times of all jobs are ordered such that $\Phi_{l_1} \leq \Phi_{l_2} \leq \cdots \leq \Phi_{l_N}$ where $\{l_1, l_2, \cdots, l_N\}$ is a permutation of $\{1, 2, \cdots, N\}$. Denote by χ_j the total time spent by the simulated system serving job j.

Consider the first completed k jobs where $1 \leq k \leq N$. At any time between 0 and Φ_{l_1}, there are $(k-1)$ jobs waiting to be processed among those k jobs. Hence, the accumulated waiting time in this period is $(k-1)\Phi_{l_1}$. Similarly, at any time between Φ_{l_1} and Φ_{l_2}, there are $(k-2)$ jobs waiting to be processed and they contribute $(k-2)\cdot(\Phi_{l_2} - \Phi_{l_1})$ to the total waiting time. Hence, the overall waiting time of the k jobs is given by:

$$\sum_{j=1}^{k-1} (k-j) \cdot (\Phi_{l_j} - \Phi_{l_{j-1}}) = \sum_{j=1}^{k-1} \Phi_{l_j}. \tag{12}$$

Since a job flowtime consists of two parts, namely, the waiting time and the processing time, the overall job flowtime for these k jobs is characterized by:

$$\sum_{j=1}^{k} \Phi_{l_j} = \sum_{j=1}^{k-1} \Phi_{l_j} + \sum_{j=1}^{k} \chi_{l_j}. \tag{13}$$

By shifting terms in (13), we get: $\Phi_{l_k} = \sum_{j=1}^{k} \chi_{l_j}$. Summing up all job flowtime, it follows that:

$$\sum_{j=1}^{N} \Phi_{l_j} = \sum_{k=1}^{N} \sum_{j=1}^{k} \chi_{l_j} = \sum_{j=1}^{N} (N+1-j)\chi_{l_j}. \tag{14}$$

Taking expectation on both sides of (14), we have:

$$\sum_{j=1}^{N} \mathbb{E}[\Phi_{l_j}] = \sum_{j=1}^{N} (N+1-j)\mathbb{E}[\chi_{l_j}]. \tag{15}$$

Applying an argument similar to that for (2) and (3) yields:

$$\mathbb{E}[\chi_{l_j}] = \frac{m_{l_j} E_{l_j}^m + r_{l_j} E_{l_j}^r}{M}, \tag{16}$$

since job l_j has at least $(m_{l_j} E_{l_j}^m + r_{l_j} E_{l_j}^r)$ amount of work to process in the simulated system. Substitute (16) into (15) and rearrange the inequality, we have:

$$\begin{aligned} \sum_{j=1}^{N} \mathbb{E}[\Phi_{l_j}] &= \frac{1}{M} \sum_{j=1}^{N} (N+1-j)(m_{l_j} E_{l_j}^m + r_{l_j} E_{l_j}^r) \\ &\geq \frac{1}{M} \sum_{j=1}^{N} (N+1-j)(m_j E_j^m + r_j E_j^r). \end{aligned} \tag{17}$$

This completes the proof of Theorem 1. □

Theorem 1 gives another lower bound for the expected value of the overall job flowtime. Thus, one can get a tighter lower bound for $\sum_{j=1}^{N} \mathbb{E}[\Phi_j]$ by combining (8) and (17), i.e.,

$$\begin{aligned} \sum_{j=1}^{N} \mathbb{E}[\Phi_j] \geq \max \Big\{ & \frac{\sum_{j=1}^{N} (E_j^m + E_j^r)}{\Delta}, \\ & \frac{\sum_{j=1}^{N} (N+1-j)(m_j E_j^m + r_j E_j^r)}{M} \Big\}. \end{aligned} \tag{18}$$

Remark 2. *In (17), the equality can be attained by letting $l_j = j$, i.e., the jobs are completed in the order which is the same as the increasing order of $(m_j E_j^m + r_j E_j^r)$. This motivates us to design scheduling algorithms based on the total amount of workload for each job and give priorities to small ones. Such a scheduling principle is similar to SRPT (Shortest remaining processing time), which schedules jobs of a single task on one machine and gives priorities to jobs with the shortest processing time [25].*

3.2 An approximate algorithm for transient scheduling

Following up on Remark 2, we extend the SRPT scheduler to design an approximate algorithm, named SEW (Smallest Effective Workload) under the transient scheduling setting.

Similar to the SRPT principle, the main idea of SEW is to give scheduling priorities to jobs with small workloads. However, to guarantee system performance in the worst

case, we also consider the variance of task execution time. Specifically, we define a total effective workload, Υ_j, for job j, which incorporates the standard deviation of task execution time by multiplying with a factor λ. To be more specific,

$$\Upsilon_j = m_j \cdot (E_j^m + \lambda\sigma_j^m) + r_j \cdot (E_j^r + \lambda\sigma_j^r). \tag{19}$$

The rationale of including σ_j^z in Υ_j is that tasks with large variation in execution times may prolong the completion of other tasks and jobs substantially, and thus should be scheduled later. Nevertheless, it still remains a problem to choose the value of λ and we shall tackle this issue in Section 5.

After computing Υ_j, SEW then schedules jobs with smaller effective workload before those with larger ones. In addition, we divide the machines into two groups of the same size, namely, the map group and reduce group, which are responsible for running map tasks and reduce tasks respectively. At the beginning of each time slot, whenever a machine in the map group is available, say, machine i, SEW randomly chooses one unscheduled map task from the pool of not-yet-finished jobs, or the set of active jobs that have the smallest value of Υ_j and assign it to machine i for processing. If all jobs are scheduled and yet some tasks are not finished, then we randomly choose from those unfinished tasks and clone each to a free machine. Only when the map phase finishes can SEW begin to schedule reduce tasks on machines in the reduce group. The scheduling process for the reduce tasks is completely the same as that for the map tasks. Algorithm 1 shows the pseudo-code of this algorithm.

Algorithm 1: SEW algorithm for the transient scheduling

Input: The jobs associated with E_j^z and σ_j^z;
Output: Machine assignments for the unscheduled/unfinished tasks at the current time slot.

```
Divide the machines into two groups: M_1 = {1, 2, ··· , M/2} and M_2 = {M/2 + 1, M/2 + 2, ··· , M};
Initialize the job set Ξ_1 = Ξ_2 = {1, 2, ··· , N};
Sort the jobs in Ξ_1 and Ξ_2 based on the increasing order of Υ_j ;
if a machine in M_1 is available then
    Choose the first job j from Ξ_1;
    Choose one task from all the unscheduled map tasks of job j uniformly at random and assign it to this machine;
    if job j has no unscheduled map task then
        Ξ_1 = Ξ_1 − {j};
    if Ξ_1 = ∅ ∧ some tasks are not finished then
        Choose one task from the unfinished map tasks uniformly at random and clone a new copy for it on this machine;
if a machine in M_2 is available then
    Choose the first job j from Ξ_2;
    if all the map tasks in job j have finished then
        Choose one task from all the unscheduled reduce tasks of job j uniformly at random and assign it to this machine;
        if job j has no unscheduled reduce task then
            Ξ_2 = Ξ_2 − {j};
        if Ξ_2 = ∅ ∧ some tasks are not finished then
            Choose one task from the unfinished reduce tasks uniformly at random and clone a new copy for it on this machine;
```

3.3 Performance guarantee for SEW

We first define for job j, $\mathcal{A}_j$, which is the accumulated effective workload of those jobs whose effective workload is no larger than Υ_j. In other words, $\mathcal{A}_j$ is given by:

$$\mathcal{A}_j = \sum_{i:\Upsilon_i \le \Upsilon_j} \Upsilon_i. \tag{20}$$

We then give the following theorems to bound the worst case job flowtime with a certain confidence level and to characterize the competitive performance of SEW.

Theorem 2. *With a probability of at least $\frac{\lambda^8}{(1+\lambda^2)^4}$, Φ_j is no more than $(E_j^m + E_j^r + \lambda\sigma_j^m + \lambda\sigma_j^r + 2\mathcal{A}_j/M)$.*

Proof. To prove this theorem, we first show the following two lemmas:

Lemma 1. *With probability of at least $\frac{\lambda^2}{1+\lambda^2}$, the machines in the map group are busy processing the jobs whose effective workload is at most Υ_j during the time interval $(0, \Psi_j - E_j^m - \lambda\sigma_j^m]$ where $\Psi_j = \max_{i\in\{1,2,\cdots,m_j\}} C_j^{m,i}$.*

Denote by $P_j^{z,i}$ the total time spent by the cluster serving task $T_j^{z,i}$ under SEW. Without loss of generality, let $T_j^{m,1}$ be the last map task to finish in job j. Based on the scheduling principle of SEW, all the machines in the map group must be busy processing jobs whose effective workload is at most Υ_j during the time interval $(0, \Psi_j - P_j^{m,1}]$.

Applying Chebyshev inequality [23], the event of $P_j^{m,1} \le E_j^m + \lambda\sigma_j^m$ happens with probability of at least:

$$\begin{aligned}\Pr\{P_j^{m,1} \le E_j^m + \lambda\sigma_j^m\} &= 1 - \Pr\{P_j^{m,1} - E_j^m \ge \lambda\sigma_j^m\} \\ &\ge 1 - \frac{1}{1+\lambda^2} = \frac{\lambda^2}{1+\lambda^2}.\end{aligned}$$

This completes the proof of Lemma 1. □

Lemma 2. *With probability of at least $\frac{\lambda^2}{1+\lambda^2}$, the machines in the reduce group are busy processing the jobs whose effective workload is at most Υ_j during the time interval $(\Psi_j, \Phi_j - E_j^r - \lambda\sigma_j^r]$.*

The proof of Lemma 2 is the same as that of Lemma 1 and we omit it here.

Define a random variable W_j^m, which is the total amount of time spent by the cluster for serving the map tasks of jobs whose effective workload is at most Υ_j. The mean and variance of W_j^m are given by:

$$\mathbb{E}[W_j^m] = \sum_{i:\Upsilon_i \le \Upsilon_j} m_i \cdot E_i^m. \tag{21}$$

$$\text{Var}[W_j^m] = \sum_{i:\Upsilon_i \le \Upsilon_j} m_i \cdot (\sigma_i^m)^2. \quad (22)$$

Let $\mathcal{A}_j^m = \sum_{i:\Upsilon_i \le \Upsilon_j} m_j \cdot (E_j^m + \lambda\sigma_j^m)$. Using Chebyshev inequality, the probability that W_j^m is no less than $\mathcal{A}_j^m$ is upper bounded by:

$$\begin{aligned}
&\Pr\left\{W_j^m \ge \mathcal{A}_j^s\right\} \\
&= \Pr\left\{W_j^m - \mathbb{E}[W_j] \ge \lambda \sum_{i:\Upsilon_i \le \Upsilon_j} m_i\sigma_i^m\right\} \\
&\le \Pr\left\{W_j^m - \mathbb{E}[W_j] \ge \lambda \sqrt{\sum_{i:\Upsilon_i \le \Upsilon_j} m_i(\sigma_i^m)^2}\right\} \\
&\le \frac{1}{1+\lambda^2}.
\end{aligned}$$

Let P_j be the event that the machines in the map group is busy processing the work of W_j^m during the interval $(0, \Psi_j - E_j^m - \lambda\sigma_j^m]$. Denote by Q_j the event that $W_j^m \le \mathcal{A}_j^m$. On the one hand, we have:

$$\Pr\{P_j \cap Q_j\} = \Pr\{P_j\} \cdot \Pr\{Q_j\} \ge \frac{\lambda^4}{(1+\lambda^2)^2},$$

since P_j and Q_j are independent. On the other hand, when both P_j and Q_j happen, it follows that:

$$M/2 \cdot (\Psi_j - E_j^m - \lambda\sigma_j^m) \le \mathcal{A}_j^m,$$

because there are $M/2$ machines in the map group. We therefore conclude that, $\Psi_j \le E_j^m + \lambda\sigma_j^m + 2\mathcal{A}_j^m/M$ holds with probability of at least $\frac{\lambda^4}{(1+\lambda^2)^2}$.

Let $\mathcal{A}_j^r = \sum_{i:\Upsilon_i \le \Upsilon_j} r_j \cdot (E_j^r + \lambda\sigma_j^r)$. Using the same argument as above, it follows that, $\Phi_j - \Psi_j \le E_j^r + \lambda\sigma_j^r + 2\mathcal{A}_j^r/M$ holds with probability of at least $\frac{\lambda^4}{(1+\lambda^2)^2}$. Therefore, with probability of at least $\frac{\lambda^8}{(1+\lambda^2)^4}$, Φ_j is upper bounded by:

$$\begin{aligned}
\Phi_j &\le E_j^m + \lambda\sigma_j^m + E_j^r + \lambda\sigma_j^r + 2\mathcal{A}_j^m/M + 2\mathcal{A}_j^r/M \\
&= E_j^m + E_j^r + \lambda(\sigma_j^m + \sigma_j^r) + \frac{2\mathcal{A}_j}{M}. \quad (23)
\end{aligned}$$

This completes the proof of Theorem 2. □

Theorem 3. *SEW($\lambda = 0$) is $O(1)$-competitive for the expected value of the overall job flowtime.*

Proof. let $T_j^{m,1}$ be the last reduce task to finished in job j. Using the arguments similar to the proof in Theorem 2, we have:

$$\Phi_j \le P_j^{m,1} + P_j^{r,1} + \frac{2\sum_{i:\Upsilon_i \le \Upsilon_j}(\sum_{k=1}^{m_i} P_i^{m,k} + \sum_{k=1}^{r_i} P_i^{r,k})}{M}. \quad (24)$$

Taking expectation on both sides of (24), it follows that:

$$\mathbb{E}[\Phi_j] \le E_j^m + E_j^r + \frac{2\sum_{i:\Upsilon_i \le \Upsilon_j}(m_i E_i^m + r_i E_i^r)}{M}.$$

Summing up all job flowtime, we have:

$$\begin{aligned}
\sum_{j=1}^N \mathbb{E}[\Phi_j] &\le \sum_{j=1}^N (E_j^m + E_j^r) \\
&\quad + \frac{2\sum_{j=1}^N \sum_{i:\Upsilon_i \le \Upsilon_j}(m_i E_i^m + r_i E_i^r)}{M}.
\end{aligned}$$

When $\lambda = 0$, this implies that:

$$\begin{aligned}
\sum_{j=1}^N \mathbb{E}[\Phi_j] &\le \sum_{j=1}^N (E_j^m + E_j^r) \\
&\quad + \frac{2\sum_{j=1}^N (N+1-j)(m_j E_j^m + r_j E_j^r)}{M}.
\end{aligned}$$

Let OPT and TS denote the expected value of the overall job flowtime achieved by the optimal algorithm and SEW respectively. Using the lower bound results from (18), we have:

$$\begin{aligned}
\frac{TS}{OPT} &\le \frac{\sum_{j=1}^N (E_j^m + E_j^r) + \frac{2\sum_{j=1}^N (N+1-j)(m_j E_j^m + r_j E_j^r)}{M}}{\max\{\frac{\sum_{j=1}^N (E_j^m + E_j^r)}{\Delta}, \frac{\sum_{j=1}^N (N+1-j)(m_j E_j^m + r_j E_j^r)}{M}\}} \\
&\le \frac{\sum_{j=1}^N (E_j^m + E_j^r)}{\frac{\sum_{j=1}^N (E_j^m + E_j^r)}{\Delta}} + 2 \le \Delta + 2.
\end{aligned}$$

This completes the proof. □

Note that, when the variance of machine processing capacity is negligible, the peak rate Δ, is the same as the mean value, i.e., $\Delta = 1$. In this case, $\frac{TS}{OPT} \le 3$, which implies that, SEW is a 3-competitive algorithm with respect to the total job flowtime.

4 Online Scheduling with Cloning

We present an approximate algorithm in this section for the online scheduling case where different jobs arrive at the cluster over time. We shall also derive a competitive performance bound for the proposed algorithm using resource augmentation arguments.

4.1 Smallest remaining effective workload based machine sharing principle

Extending the transient scheduling setting in Section 3, we design an online scheduling algorithm to allow the number of cloned copies of tasks to vary dynamically. We call this online algorithm the *Smallest Remaining Effective Workload based β-fraction Sharing plus Cloning* (SREW+C(β)).

The algorithm depends on the system parameter $\beta \in (0, 1)$. At a high level, SREW+C(β) works as follows: At the beginning of each time slot, the scheduler computes a priority for every active job. Jobs with the highest priority share the machines equally while satisfying the following condition:

$$\frac{\text{\# of running jobs}}{\text{\# of not-yet finished jobs}} = \beta.$$

When β is set to 1, the scheduler reduces to Fair Scheduler in Hadoop [1]. When β is close to 0, the scheduler is the same as the FIFO scheduler. By tuning the value of β, we could obtain a scheduler that best fits a cluster. In addition, this β-fraction sharing principle yields a bounded competitive ratio as presented in the subsequent sections.

Let $\psi^s(t)$ be the set of active jobs at the beginning of time slot t. Denote by $m_j(t)$ and $r_j(t)$ the number of unfinished map and reduce tasks in job j respectively. The remaining effective workload of job j at time slot t is then characterized by:

$$\Upsilon_j(t) = m_j(t) \cdot (E_j^m + \lambda\sigma_j^m) + r_j(t) \cdot (E_j^r + \lambda\sigma_j^r). \quad (25)$$

The scheduler computes $\Upsilon_j(t)$ for each job in $\psi^s(t)$ and guarantees that jobs with smaller $\Upsilon_j(t)$ have higher priority to be scheduled. Let $\psi_j^s(t)$ denote the set of jobs whose remaining effective workload is greater than or equal to that of job j at time slot t. Further define $g_j(t)$ as follows:

$$g_j(t) = \begin{cases} \frac{M}{\beta|\psi^s(t)|} & |\psi_j^s(t)| - 1 \geq (1-\beta)|\psi^s(t)|, \\ 0 & |\psi_j^s(t)| < (1-\beta)|\psi^s(t)|, \\ \frac{(|\psi_j^s(t)|-(1-\beta)|\psi^s(t)|)\cdot M}{\beta|\psi^s(t)|} & \text{otherwise.} \end{cases}$$

To simplify the analysis, we assume $g_j(t)$ is an integer. As such, job j is allocated $g_j(t)$ machines at time slot t under SREW+C(β).

After the number of machines is allocated for each job, the scheduler chooses appropriate tasks of the active jobs to schedule and make cloning decisions. Moreover, the scheduler begins to launch reduce tasks only after all the map tasks have completed. Take job j for example, let $u_j^z(t)$ be the number of unscheduled tasks of job j at the beginning of time slot t. Further define $o_j(t)$ to characterize the number of machines which are still running the tasks of job j at time t. When $u_j^z(t) > 0$, the scheduler will launch $\lfloor \frac{g_j(t)-o_j(t)}{u_j^z(t)} \rfloor$ or $\lceil \frac{g_j(t)-o_j(t)}{u_j^z(t)} \rceil$ copies of each unscheduled task on available machines such that the total number of newly launched copies for job j is exactly $(g_j(t) - o_j(t))$. Here, $\lfloor x \rfloor$ and $\lceil x \rceil$ represent the largest integer which does not exceed x and the smallest integer that is above x respectively. On the contrary, if $u_j^z(t) = 0$, $(g_j(t) - o_j(t))$ machines are assigned to existing running tasks in job j to make cloned copies. Algorithm 2 shows the pseudo-code of SREW+C(β).

Algorithm 2: SREW+C(β) Algorithm Design for Online Scheduling

```
At the beginning of time slot t, update ψ^s(t), the set of jobs which still have unfinished tasks;
Compute Υ_j(t) for each j ∈ ψ^s(t) based on (25) and sort the jobs in ψ^s(t) according to the increasing order of Υ_j(t);
for each job j ∈ ψ^s(t) do
    Compute g_j(t), the number of machines allocated to job j based on the β-fraction sharing principle;
    Update u_j^z(t) and o_j(t);
    if o_j(t) > g_j(t) then
        Preempt (o_j(t) − g_j(t)) running copies of job j uniformly at random;
    else if g_j(t) == 0 then
        continue;
    else
        if u_j^z(t) > 0 then
            Launch ⌊(g_j(t)−o_j(t))/u_j^z(t)⌋ or ⌈(g_j(t)−o_j(t))/u_j^z(t)⌉ copies for each unscheduled task in the Map/Reduce phase such that the total number of newly launched copies is exactly (g_j(t) − o_j(t));
        else
            Launch ⌊(g_j(t)−o_j(t))/z_j(t)⌋ or ⌈(g_j(t)−o_j(t))/z_j(t)⌉ cloned copies for each running task on idle machines such that the total number of newly launched cloned copies is (g_j(t) − o_j(t)).
```

4.2 Performance guarantee for SREW+C(β) with resource augmentation

In this section, we use resource augmentation argument to bound the competitive performance of the SREW+C(β) algorithm with respect to the expected value of the total job flowtime. To achieve this, we first make the following assumption:

Assumption 2. *We shall model (approximate) the service capacity under cloning, $W_j^{z,i}(t)$, by its mean, i.e.,*

$$W_j^{z,i}(t) = \omega(|V_j^{z,i}(t)|) = \mathbb{E}\Big[\max_{k\in V_j^{z,i}(t)} X_k(t)\Big]. \tag{26}$$

Remark 3. *Assumption 2 essentially replaces the service capacity of the system with the mean but accounts for the average gains one might expect with task cloning.*

Lemma 3. *$\{\omega(d)\}_{d\in\mathbb{N}}$ is a concave sequence, i.e., $\omega(d) - \omega(d-1) \leq \omega(d-1) - \omega(d-2)$.*

Proof. For a fixed t, let $F_{X_i(t)}(x)$ and $F_{H_d(t)}(x)$ denote the cumulative density function of $X_i(t)$ and $H_d(t)$ respectively where $H_d(t) = \max_{k\in V_j^{z,i}(t)} X_k(t)$ and $d = |V_j^{z,i}(t)|$. Note that $F_{H_d(t)}(x) = F^d_{X_i(t)}(x)$. We then have: $\omega(d) = \mathbb{E}[H_d(t)] = \int_0^\infty (1 - F^d_{X_i(t)}(x))dx$, which implies that,

$$\begin{aligned} \omega(d) - \omega(d-1) &= \int_0^\infty F^{d-1}_{X_i(t)}(x)\cdot(1 - F_{X_i(t)}(x))dx \\ &\leq \int_0^\infty F^{d-2}_{X_i(t)}(x)\cdot(1 - F_{X_i(t)}(x))dx \\ &= \omega(d-1) - \omega(d-2). \end{aligned}$$

This completes the proof. □

Our main result that characterizes the competitive performance of SREW+C(β) is given by the following theorem:

Theorem 4. *The algorithm SREW+C(β) is $(1 + 2\beta + \epsilon)$-speed $O(\frac{1}{\beta\epsilon})$-competitive with respect to the overall job flowtime when $\lambda = 0$ and $\beta < \min\{\frac{M}{N}, \frac{1}{2}\}$.*

We adopt the use of a potential function to prove Theorem 4. The key step of this method is to construct a proper function which combines the optimal schedule and SREW+C(β). Let SR and OPT be the overall job flowtime under SREW+C(β) and the optimal scheduling policy respectively. We define a potential function, $\Lambda(t)$, whose evolution involves job arrivals as well as task processing. $\Lambda(t)$ should satisfy the following three properties:

- Boundary Conditions: $\Lambda(0) = \Lambda(\infty) = 0$.
- Job arrival: there exists a constant c_1 such that, the overall increase in $\Lambda(t)$ caused by a job arrival is upper bounded by $c_1 \cdot OPT$.
- Task processing: there exist some constant c_2 and c_3 such that, with a factor of $(1 + 2\beta + \epsilon)$ resource augmentation, the overall increase in $\Lambda(t)$ caused by the task processing is upper bounded by $(c_2 \cdot OPT - c_3\beta\epsilon \cdot SR)$.

By summing up all increases in the potential function while accounting for the boundary conditions, one can see that the existence of such a potential function guarantees that

$SR \leq \frac{c_1+c_2}{c_3} \cdot \frac{1}{\beta\epsilon} \cdot OPT$ under a $(1+2\beta+\epsilon)$-speed resource augmentation.

4.3 Proof of Theorem 4

Before going into the details of the proof, we first define a new function, $\varpi : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ to extend the domain of ω. Specifically, for any $x \in [a, a+1)$ where $a \in \mathbb{N}$, we let

$$\varpi(x) = \omega(a) + (\omega(a+1) - \omega(a))(x-a).$$

The following lemmas illustrate two important properties of ϖ.

Lemma 4. *ϖ is an increasing and concave function.*

Proof. To prove this lemma, it suffices to show that, for any $x \in [a, a+1)$ and any $y \in \mathbb{R}_{\geq 0}$, it follows that,

$$\varpi(y) \leq \varpi(x) + (\varpi(a+1) - \varpi(a))(y-x), \tag{27}$$

Consider $y \in [b, b+1)$ where $b \in \mathbb{N}$. Suppose $y \geq x$. Based on the definition of ϖ, we have:

$$\begin{aligned} \varpi(y) &= \varpi(b) + (\varpi(b+1) - \varpi(b))(y-b) \\ &\leq \varpi(b) + (\varpi(a+1) - \varpi(a))(y-b). \end{aligned} \tag{28}$$

Inferred from Lemma 3 that

$$\begin{aligned} \varpi(b) &= \sum_{k=a}^{b-1} (\varpi(k+1) - (\varpi(k)) + \varpi(a) \\ &\leq (b-a)(\varpi(a+1) - (\varpi(a)) + \varpi(a). \end{aligned} \tag{29}$$

Substitute (29) into (28) to yield:

$$\begin{aligned} \varpi(y) &\leq \varpi(a) + (\varpi(a+1) - \varpi(a))(y-a) \\ &= \varpi(x) + (\varpi(a+1) - \varpi(a))(y-x). \end{aligned}$$

For the case where $y < x$, we can repeat the similar process to show that (27) holds. This completes the proof. □

Lemma 5. $\frac{\varpi(a)}{a} \geq \frac{\varpi(b)}{b}$ *for any* $b \geq a > 0$.

Proof. Since ϖ is a continuous and concave function, it follows that, $\varpi(\xi x + (1-\xi)y) \geq \xi\varpi(x) + (1-\xi)\varpi(y)$ for any $x, y \in \mathbb{R}_{\geq 0}$ and $\xi \in [0,1]$. Specifically, consider the case when $x = 0$, $y = b$ and $\xi = 1 - \frac{a}{b}$, we have: $\varpi(a) \geq (1-\frac{a}{b})\varpi(0) + \frac{a}{b}\varpi(b) \geq \frac{a}{b}\varpi(b)$. This completes the proof. □

4.3.1 Constructing potential function $\Lambda(t)$

Let $\delta = (1+2\beta+\epsilon)$ and $\rho(x) = \delta\varpi(x)$ for all $x \in \mathbb{R}_{\geq 0}$. Denote by $\gamma_j^{z,i}(t)$ the number of machines allocated to task $T_j^{z,i}$ at time slot t under the optimal scheduling policy. Since we are using a resource augmentation of $\delta-$speed under SREW+C, for all $j \in \{1, 2, \cdots, N\}$ and $i \in \{1, 2, \cdots, z_j\}$, it follows that:

$$\sum_{\tau=a_j+1}^{\infty} \rho(d_j^{z,i}(\tau)) = \sum_{\tau=a_j+1}^{\infty} \varpi(\gamma_j^{z,i}(\tau)) = p_j^z. \tag{30}$$

Let $\psi^*(t)$ be the set of jobs that are still active at the beginning of time slot t under the optimal scheduling policy. Thus, SR and OPT are given by:

$$SR = \sum_{t=1}^{\infty} |\psi^s(t)| \quad \text{and} \quad OPT = \sum_{t=1}^{\infty} |\psi^*(t)|. \tag{31}$$

Denote by $\pi_j^{z,i}(t)$ the cumulative service difference between the two schedules for task $T_j^{z,i}$ at time slot t. We have:

$$\pi_j^{z,i}(t) = \max\Big[\sum_{\tau=a_j+1}^{t} \varpi(\gamma_j^{z,i}(\tau)) - \sum_{\tau=a_j+1}^{t} \rho(d_j^{z,i}(\tau)),\ 0\Big], \tag{32}$$

Substitute (30) into (32), we have:

$$\pi_j^{z,i}(0) = \pi_j^{z,i}(\infty) = 0. \tag{33}$$

Define a potential function $\Lambda(t)$ as follows:

$$\Lambda(t) = \sum_{j=1}^{N}\sum_{i=1}^{m_j} \frac{\pi_j^{m,i}(t)}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})} + \sum_{j=1}^{N}\sum_{i=1}^{r_j} \frac{\pi_j^{r,i}(t)}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})}. \tag{34}$$

For ease of presentation, we shall use $\sum_{i=1}^{z_j}$ to represent $(\sum_{i=1}^{m_j} + \sum_{i=1}^{r_j})$ in the sequel. Based on (33), we have:

$$\Lambda(0) = \Lambda(\infty) = 0.$$

With the boundary conditions satisfied, we shall focus on the evolution of the potential function, which is given by:

$$\begin{aligned} \Lambda(t) - \Lambda(t-1) &= \sum_{j=1}^{N}\sum_{i=1}^{z_j} \frac{\pi_j^{z,i}(t)}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})} - \sum_{j=1}^{N}\sum_{i=1}^{z_j} \frac{\pi_j^{z,i}(t-1)}{\varpi(\frac{M}{\beta|\psi_j^s(t-1)|})} \\ &= \underbrace{\sum_{j=1}^{N}\sum_{i=1}^{z_j} \frac{\pi_j^{z,i}(t)}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})} - \sum_{j=1}^{N}\sum_{i=1}^{z_j} \frac{\pi_j^{z,i}(t-1)}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})}}_{\mathcal{P}(t)} \\ &\quad + \underbrace{\sum_{j=1}^{N}\sum_{i=1}^{z_j} \frac{\pi_j^{z,i}(t-1)}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})} - \sum_{j=1}^{N}\sum_{i=1}^{z_j} \frac{\pi_j^{z,i}(t-1)}{\varpi(\frac{M}{\beta|\psi_j^s(t-1)|})}}_{\mathcal{J}(t)}. \end{aligned} \tag{35}$$

Notice from (35) that the evolution of $\Lambda(t)$ includes two parts, namely, $\mathcal{P}(t)$ and $\mathcal{J}(t)$. $\mathcal{P}(t)$ denotes the changes caused by the task processing while $\mathcal{J}(t)$ denotes the changes caused by job arrivals or departures. In the sequel, we shall derive bounds for $\mathcal{P}(t)$ and $\mathcal{J}(t)$ separately.

4.3.2 Changes in $\Lambda(t)$ caused by job arrivals and job departures

In (35), $\mathcal{J}(t)$ can be reformulated as:

$$\mathcal{J}(t) = \sum_{j=1}^{N}\sum_{i=1}^{z_j} \Big(\frac{\pi_j^{z,i}(t-1)}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})} - \frac{\pi_j^{z,i}(t-1)}{\varpi(\frac{M}{\beta|\psi_j^s(t-1)|})}\Big). \tag{36}$$

According to the SREW+C(β) scheduling with $\lambda = 0$, a job in $\psi_j^s(t)$ can not leave the system before job j. Therefore, we have, for any j and t,

$$|\psi_j^s(t)| \geq |\psi_j^s(t-1)|,$$

which implies that, $\varpi(\frac{M}{\beta|\psi_j^s(t-1)|}) \geq \varpi(\frac{M}{\beta|\psi_j^s(t)|})$ and

$$\begin{aligned} \mathcal{J}(t) &= \sum_{j=1}^{N}\sum_{i=1}^{z_j} \frac{\pi_j^{z,i}(t-1)\Big(\varpi(\frac{M}{\beta|\psi_j^s(t-1)|}) - \varpi(\frac{M}{\beta|\psi_j^s(t)|})\Big)}{\varpi(\frac{M}{\beta|\psi_j^s(t-1)|})\varpi(\frac{M}{\beta|\psi_j^s(t)|})} \\ &\overset{\text{(i)}}{\leq} \sum_{j=1}^{N}\sum_{i=1}^{z_j} \pi_j^{z,i}(t-1)\Big(\varpi(\frac{M}{\beta|\psi_j^s(t-1)|}) - \varpi(\frac{M}{\beta|\psi_j^s(t)|})\Big) \\ &\overset{\text{(ii)}}{\leq} \sum_{j=1}^{N}\sum_{i=1}^{z_j} p_j^z\Big(\varpi(\frac{M}{\beta|\psi_j^s(t-1)|}) - \varpi(\frac{M}{\beta|\psi_j^s(t)|})\Big), \end{aligned}$$

where (i) is due to the fact that $\varpi(\frac{M}{\beta|\psi_j^s(t-1)|})$ and $\varpi(\frac{M}{\beta|\psi_j^s(t)|})$ are not smaller than one since $\beta \leq \frac{M}{N}$. (ii) is because of the fact that $\pi_j^{z,i}(t-1) \leq p_j^z$. Therefore, we have:

$$\begin{aligned}\sum_{t=1}^{\infty}\mathcal{J}(t) &\leq \sum_{t=1}^{\infty}\sum_{j=1}^{N}\sum_{i=1}^{z_j} p_j^z(\varpi(\frac{M}{\beta|\psi_j^s(t-1)|}) - \varpi(\frac{M}{\beta|\psi_j^s(t)|}))\\ &= \sum_{j=1}^{N}\sum_{i=1}^{z_j} p_j^z \sum_{t=a_j+1}^{\infty}\Big(\varpi(\frac{M}{\beta|\psi_j^s(t)|}) - \varpi(\frac{M}{\beta|\psi_j^s(t+1)|})\Big)\\ &\leq \sum_{j=1}^{N}\sum_{i=1}^{z_j} p_j^z \varpi(\frac{M}{\beta|\psi_j^s(a_j+1)|}) \leq \Delta\sum_{j=1}^{N}\sum_{i=1}^{z_j} p_j^z.\end{aligned}$$

Since the flowtime of job j is lower bounded by $\sum_{i=1}^{z_j} p_j^z/M$ under the optimal policy, it follows that:

$$\sum_{t=1}^{\infty}\mathcal{J}(t) \leq M\Delta \cdot OPT. \tag{37}$$

4.3.3 *Changes in $\Lambda(t)$ caused by task processing*

Similar to (36), $\mathcal{P}(t)$ can be reformulated as:

$$\mathcal{P}(t) = \sum_{j=1}^{N}\sum_{i=1}^{z_j}\frac{\pi_j^{z,i}(t) - \pi_j^{z,i}(t-1)}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})}. \tag{38}$$

Let $\Omega^s(t)$ and $\Omega^*(t)$ be the set which contains all the tasks that have not yet finished under SREW+C and the optimal schedule respectively. It follows that:

$$\begin{aligned}&\pi_j^{z,i}(t) - \pi_j^{z,i}(t-1)\\ \leq\ & \mathbb{1}(T_j^{z,i} \in \Omega^*(t))\varpi(\gamma_j^{z,i}(t))\\ &+\mathbb{1}(T_j^{z,i} \notin \Omega^*(t))(\varpi(\gamma_j^{z,i}(t)) - \rho(d_j^{z,i}(t))) \quad (39)\\ =\ & \varpi(\gamma_j^{z,i}(t)) - \mathbb{1}(T_j^{z,i} \notin \Omega^*(t))\rho(d_j^{z,i}(t)). \quad (40)\end{aligned}$$

When $T_j^{z,i} \in \Omega^*(t)$, task $T_j^{z,i}$ has not completed under the optimal policy and the drift is bounded by the first term in (39). When $T_j^{z,i} \notin \Omega^*(t)$, it means task $T_j^{z,i}$ has completed under the optimal policy. As such, the difference term in (32) is positive and its drift is given by the the second term in (39). Substitute (40) into (38), we have:

$$\mathcal{P}(t) \leq \underbrace{\sum_{j=1}^{N}\sum_{i=1}^{z_j}\frac{\varpi(\gamma_j^{z,i}(t))}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})}}_{\mathcal{P}^1(t)} \underbrace{- \sum_{j=1}^{N}\sum_{i=1}^{z_j}\frac{\mathbb{1}(T_j^{z,i} \notin \Omega^*(t))\rho(d_j^{z,i}(t))}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})}}_{\mathcal{P}^2(t)}. \tag{41}$$

To bound the first term in $\mathcal{P}(t)$, i.e., $\mathcal{P}^1(t)$, we will consider the following two cases:

Case 1: When $\gamma_j^{z,i}(t)) \leq \frac{M}{\beta|\psi_j^s(t)|}$, it follows that:

$$\frac{\varpi(\gamma_j^{z,i}(t))}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})} \leq \mathbb{1}(\gamma_j^{z,i}(t) \geq 1).$$

Case 2: When $\gamma_j^{z,i}(t)) > \frac{M}{\beta|\psi_j^s(t)|}$, we apply Lemma 5 to yield:

$$\frac{\varpi(\gamma_j^{z,i}(t))}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})} \leq \frac{\gamma_j^{z,i}(t)}{M/(\beta|\psi_j^s(t)|)} \leq \frac{\gamma_j^{z,i}(t)\beta|\psi^s(t)|}{M}.$$

Combining Case 1 and Case 2, $\frac{\varpi(\gamma_j^{z,i}(t))}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})}$ is upper bounded by $\Big(\mathbb{1}(\gamma_j^{z,i}(t) \geq 1) + \frac{\gamma_j^{z,i}(t)\beta|\psi^s(t)|}{M}\Big)$. Therefore, $\mathcal{P}^1(t)$ is bounded by:

$$\begin{aligned}\mathcal{P}^1(t) &\leq \sum_{j=1}^{N}\sum_{i=1}^{z_j}\mathbb{1}(\gamma_j^{z,i}(t) \geq 1) + \sum_{j=1}^{N}\sum_{i=1}^{z_j}\frac{\gamma_j^{z,i}(t)\beta|\psi^s(t)|}{M}\\ &\overset{\text{(iii)}}{\leq} M + \beta|\psi^s(t)| \leq M|\psi^*(t)| + \beta|\psi^s(t)|,\end{aligned} \tag{42}$$

where (iii) is due to the fact that $\sum_{j=1}^{N}\sum_{i=1}^{z_j}\gamma_j^{z,i}(t) \leq M$.

Now, it remains to bound the second term, $\mathcal{P}^2(t)$. Note that, $d_j^{z,i}(t) \leq \frac{M}{\beta|\psi_j^s(t)|}$, applying Lemma 5, we have:

$$\frac{\rho(d_j^{z,i}(t))}{\varpi(\frac{M}{\beta|\psi_j^s(t)|})} \geq \delta \cdot \frac{d_j^{z,i}(t)}{M/(\beta|\psi_j^s(t)|)},$$

which implies:

$$\begin{aligned}\mathcal{P}^2(t) &\leq -\delta\sum_{j=1}^{N}\sum_{i=1}^{z_j}\frac{\mathbb{1}(T_j^{z,i} \notin \Omega^*(t)) \cdot d_j^{z,i}(t)}{M/(\beta|\psi_j^s(t)|)}\\ &= -\delta\beta\sum_{j=1}^{N}\sum_{i=1}^{z_j}\frac{d_j^{z,i}(t)\cdot|\psi_j^s(t)|}{M}\\ &\quad + \delta\sum_{j=1}^{N}\sum_{i=1}^{z_j}\frac{\mathbb{1}(T_j^{z,i} \in \Omega^*(t)) \cdot d_j^{z,i}(t)}{M/(\beta|\psi_j^s(t)|)}.\end{aligned} \tag{43}$$

Based on the scheduling principle of SREW+C(β), we have:

$$d_j^{z,i}(t)|\psi_j^s(t)| \geq (1-\beta)\cdot d_j^{z,i}(t)|\psi^s(t)|. \tag{44}$$

and for all $j \in \{1, 2, \cdots, N\}$,

$$\begin{aligned}\sum_{i=1}^{z_j}\mathbb{1}(T_j^{z,i} \in \Omega^*(t))d_j^{z,i}(t) &\leq \mathbb{1}(j \in \psi^*(t))M/(\beta|\psi^s(t)|)\\ &\leq \mathbb{1}(j \in \psi^*(t))M/(\beta|\psi_j^s(t)|).\end{aligned} \tag{45}$$

Substitute (44) and (45) into (43), it follows that:

$$\begin{aligned}\mathcal{P}^2(t) &\leq -\delta\beta(1-\beta)|\psi^s(t)|\sum_{j=1}^{N}\sum_{i=1}^{z_j}\frac{d_j^{z,i}(t)}{M}\\ &\quad + \delta\sum_{j=1}^{N}\mathbb{1}(j \in \psi^*(t))\\ &= -\delta\beta(1-\beta)|\psi^s(t)| + \delta|\psi^*(t)|.\end{aligned} \tag{46}$$

Combine (42) and (46), $\mathcal{P}(t)$ is bounded by:

$$\begin{aligned}\mathcal{P}(t) &\leq (M+\delta)|\psi^*(t)| - \beta(\delta(1-\beta)-1)|\psi^s(t)|\\ &\leq (M+\delta)|\psi^*(t)| - \frac{\beta\epsilon}{2}|\psi^s(t)|,\end{aligned}$$

where the last inequality is due to the fact that $\delta = (1+2\beta+\epsilon)$ and $\delta(1-\beta) \geq \frac{\epsilon}{2}$ since $\beta < \frac{1}{2}$. Summing up all $\mathcal{P}(t)$ over t, we have:

$$\sum_{t=1}^{\infty}\mathcal{P}(t) \leq (M+\delta)OPT - \frac{\beta\epsilon}{2}SR. \tag{47}$$

Finally, combine (35), (37) and (47), we have:

$$0 = \Lambda(\infty) - \Lambda(0) = \sum_{t=1}^{\infty}(\Lambda(t) - \Lambda(t-1)) \leq (M + \delta + M\Delta)OPT - \frac{\beta\epsilon}{2}SR. \quad (48)$$

(48) implies that $SR \leq \frac{2(M+\delta+M\Delta)}{\beta\epsilon}OPT$, which completes the proof of Theorem 4. □

5 Performance Evaluation

In this section, we evaluate the performance of the SREW+C(β) algorithm via extensive simulations driven by Google cluster-usage traces [27]. The traces contain the information of job submission and the completion time of Google services on a cluster with 11,000 servers. It also includes the number of tasks in each job as well as the duration of each task without preemption. From the traces, we extract the statistics of jobs during a 28-hour period as shown in Table 2. We also exclude the unfinished jobs as well as those which have specific constraints on machine attributes. All the experiments are conducted on a PC with a 2.6GHz Intel i5 Dual-core CPU.

Simulation Methodology: Based on the trace data, we estimate the first and second moment of the task service time for each job, which are used as a priori information under the SREW+C(β) algorithm. Once a cloned copy is scheduled, we set the service time of this copy to be the same as that of a task which is randomly chosen from all the tasks in the same job. We repeat the same simulation for each of the following evaluations ten times and take the average to obtain the final results.

Baseline Algorithms: We use the following four algorithms as the baselines for comparison with the SREW+C(β) algorithm:

- Microsoft Mantri's Speculative Execution Scheme: The speculative execution scheme of Mantri [5] has been demonstrated to be better than straggler mitigation schemes in Hadoop [1] , Dryad [18] , MapReduce [10] and LATE [37] (Refer to Section 6.1 in [5]). In Mantri, the system estimates the remaining time to finish, t_{rem}, for each task and predicts t_{new}, the required service time of a relaunched copy of the task. A speculative copy is launched when the probability $\Pr(t_{rem} > \frac{c+1}{c}t_{new})$ is above a certain threshold (default value = 0.25) where c is the number of copies of the task currently running.
- Dolly: Dolly is proposed by AMPLab in [3] to mitigate stragglers via cloning small jobs. Dolly adopts a simple policy of admission control to perform cloning with an allotted resource budget. When the total number of clones in the cluster is under a configurable fraction (β) of the total capacity of the cluster and a utilization threshold (τ), cloning will be applied following a simple computation based on limiting the probability that a job straggles. Under Dolly, all the tasks within a job will have the same number of clones.
- Grass: Grass is another straggler mitigation scheme proposed by AMPLab in [4] to trim stragglers for approximation jobs which require only a subset of tasks to finish based on job completion deadline or some error bound(s) of the computational results of the job. Grass combines two different strategies to launch speculative copies for stragglers, namely, the Greedy Speculative (GS) scheduling and the Resource Aware Speculative (RAS) scheduling. The point of switching from RAS to GS depends on the values of deadline/error bound, prediction accuracy and cluster utilization. To make a fair comparison between Grass and SREW+C(β), we adopt Grass's policy for handling error-bound jobs as the baseline and set the error bound to zero[1].
- Smart Cloning Algorithm (SCA): SCA is a cloning algorithm which is proposed in [32], [34]. At the beginning of each time slot, SCA first chooses which task to schedule and then performs a convex optimization to determine the number of copies assigned for each task. Under SCA, all the cloned copies are launched in parallel at the same time on available machines. SCA has been demonstrated to be able to reduce the elapsed time of small jobs substantially. However, SCA needs to estimate the distribution of the task service time and the estimation accuracy can have a high impact on the system performance.

TABLE 2
Google trace data statistics

Trace duration (seconds)	102767
Average number of tasks per job	123.8
Minimum task duration (seconds)	13.5
Maximum task duration (seconds)	22919.3
Average task duration (seconds)	1246.7

Implementation of SREW+C(β): Since the four baseline algorithms mentioned above do not support preemption and checkpointing, we modify SREW+C(β) to yield a non-preempted version for fair comparison. Specifically, we remove Line 7 in Algorithm 2 to allow a job to occupy all the machines which are still running their tasks at the beginning of each time slot. As such, under the non-preempted algorithm, the number of machines allocated to a job may exceed the quota it deserves based on the sharing principle.

Performance Metric: We compare the average job flowtime as well as the overall distribution of job flowtime among all of the aforementioned algorithms.

5.1 The impact of different parameters under SREW+C(β)

In this subsection, we run the job traces on 11,000 machines and tune the parameters β and λ to evaluate the average job flowtime under SREW+C(β).

To evaluate the impact of β on system performance, we set the variance coefficient, i.e., λ, to zero and evaluate the resultant job flowtime achieved by SREW+C(β) using different datasets. We extract three hour-long traces from the original Google trace as the testing datasets. As illustrated in Fig. 2(a), when $\beta = 0.7$, which corresponds to the scheduling of more than half of the active jobs with the smallest effective workload in each time slot, the average job flowtime attains its minimum under all these three datasets.

1. Since the details of learning the Switching Point from RAS to GS are not clearly explained in [4], we determine the Switching Point only based on the values of job sizes.

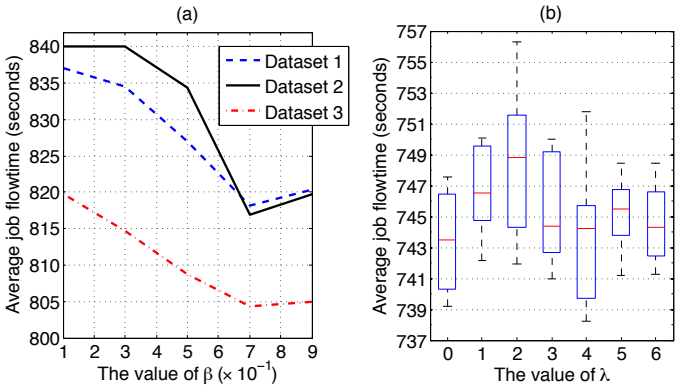


Fig. 2. The average job flowtime under the SREW+C(β) algorithm with different β and λ.

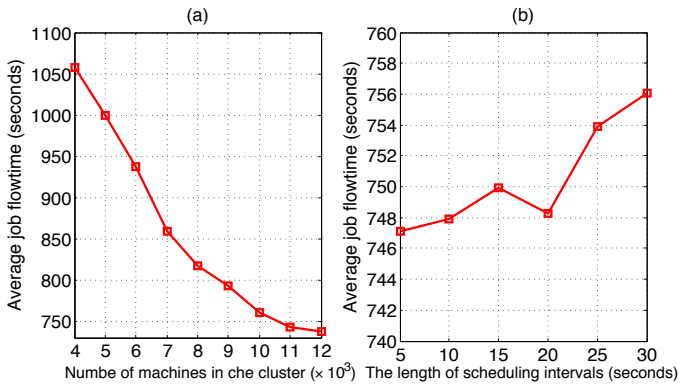


Fig. 3. The average job flowtime under the SREW+C(β) algorithm with different M and lengths of scheduling intervals.

It is worth noting that, when β is above 0.6, the average job flowtime does not vary much.

In addition, we set $\beta = 0.7$ and tune the value of λ to evaluate the impact of the variance coefficient in (25) on the average job flowtime using the whole Google traces. As implied from Section 3.3, a larger λ provides a worst case performance guarantee for the job flowtime with a higher probability at the expense of a larger average job flowtime. To better demonstrate this argument, we depict the statistics of ten simulations under a fixed λ in Fig. 2(b). It shows that, when $\lambda = 0$, the average job flowtime attains its minimum, while, the results fluctuate heavily between different simulations. When $\lambda = 1, 5$ or 6, the results are more stable across different simulations. Therefore, one can choose an appropriate λ by considering the trade-off between the average performance and the fluctuation impacts.

In the subsequent simulations, we shall choose $\beta = 0.7$ and $\lambda = 1$ for the SREW+C(β) algorithm.

5.2 The impact of M

In this part, we scale out different number of machines in the cluster to show its impact on the average job flowtime. Observe from Fig. 3(a) that the average job flowtime decreases as the number of machines in the cluster increases. When M is less than 7000, the average job flowtime reduces almost linearly as M increases. However, the average job flowtime does not decrease much when M increases beyond 10000.

5.3 The impact of the length of scheduling intervals

In this part, we evaluate the cluster performance by tuning the length of a scheduling interval, i.e., the length of each time slot. As one may expect, with short scheduling intervals, SREW+C(β) makes fast scheduling decisions and thus may reduce the job delay. Results in Fig. 3(b) show that the average job flowtime does increase with the length of scheduling intervals. Moreover, when the length of a scheduling interval is 5 seconds, our implementation of SREW+C(β) only takes 5 milliseconds to make scheduling decisions in each interval under our simulator. Given such negligible scheduling latency/overhead, we therefore choose the scheduling interval to be 5 seconds for SREW+C(β).

5.4 Comparison against baseline algorithms

With the parameters set above, we proceed to compare the performance of SREW+C(β) with the other four baseline schemes, namely, Mantri, Dolly, Grass and SCA, in a cluster with 11,000 machines. As illustrated in Table 3, the average job flowtime under SREW+C(β) is 742 seconds while that of the baseline schemes varies between 811 to 860 seconds. In other words, SREW+C(β) reduces the overall job flowtime by 9% comparing to SCA and 14% comparing to Mantri. When comparing to the state of the art schemes from AMPLab, SREW+C(β) still reduces the overall job flowtime by more than 10%.

To make a more comprehensive comparison between different algorithms, we also plot the CDF of the job flowtime in Fig. 4 and Fig. 5. As shown in Fig. 4(a), SREW+C(β) achieves the best performance for small jobs: Nearly 63% jobs complete within 300 seconds while only 56% and 53% jobs complete within 300 seconds under SCA and Mantri respectively. This means SREW+C(β) can reduce the flowtime of small jobs substantially. One can see that SREW+C(β) also achieves the best performance for large jobs. For instance, about 90% jobs can complete within 1500 seconds under SREW+C(β). By contrast, only 88% and 87% jobs can complete within such time-span under SCA and Mantri respectively.

As shown in Fig. 5(a), for small jobs which take less than 300 seconds to complete, SREW+C(β) performs similar to Dolly but much better than Grass in terms of job flowtime. Fig. 5(b) shows the results for those jobs which take more than 300 seconds to complete. Observe that SREW+C(β) is better than both Grass and Dolly for reducing the job flowtime of large jobs. This is mainly due to two advantages of SREW+C(β) over Dolly and Grass: Firstly, SREW+C(β) makes clones for jobs dynamically according to current cluster utilization and thus can help to reduce the flowtime for almost all jobs. Secondly, SREW+C(β) makes clones based on the remaining effective job workload which serves as a good metric to combine both task service time and machine service variability. As such, SREW+C(β) can jointly consider job scheduling and task cloning in order to optimize job response performance and mitigate the impact of stragglers.

Interestingly, Grass is slightly better than Dolly in reducing the flowtime of large jobs. This is because Dolly prefers to clone small jobs while Grass makes speculative copies

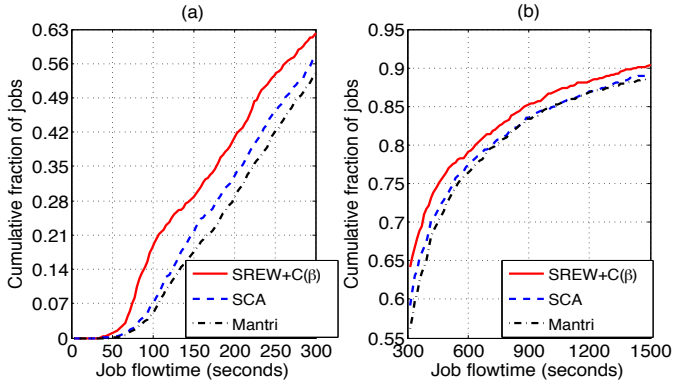


Fig. 4. The cumulative distribution of job flowtime under SREW+C(β), SCA and Mantri.

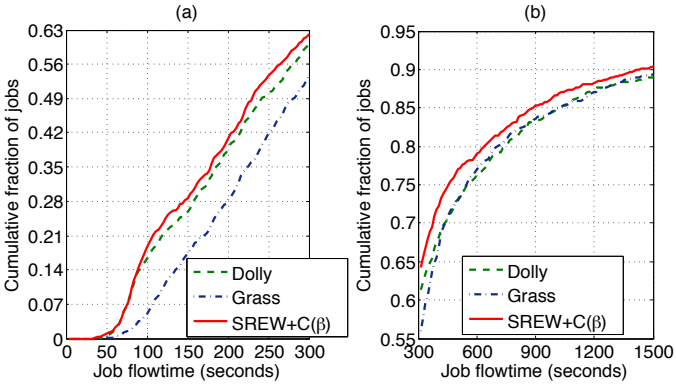


Fig. 5. The cumulative distribution of job flowtime under SREW+C(β), Grass and Dolly.

at the granularity of task (rather than job) and does not distinguish between small jobs and large jobs.

We proceed to compare the complexity of SREW+C(β) and that of the four baseline schemes. At the beginning of each scheduling interval, SCA needs to perform a convex optimization for making scheduling decisions. This may incur substantial scheduling latency and thus overhead. Mantri and Grass require the monitoring of the progress of each running task, which results in extra system instrumentation and measurement overheads. By contrast, SREW+C(β) makes scheduling decisions only based on the remaining effective workload, which can be readily computed. Dolly can also be implemented with small overhead since it makes clones for jobs following a simple computation. In summary, the implementation complexity and scheduling latency/overheads of SREW+C(β) are much lower than those of SCA, Mantri and Grass.

6 Related Work

The straggler problem was first identified in the original MapReduce paper [10]. Since then, various Straggler-Detection-based redundancy schemes [5], [9], [18], [37] have been proposed to tackle the problem. These solutions mainly focus on timely identification of stragglers and accurate prediction of the runtime of tasks. By contrast, Ananthanarayanan *et al.* designed GRASS [4] which carefully adopts the detection approach to trim stragglers for approximation

TABLE 3
Average job flowtime under different algorithms

Algorithms	Average job flowtime (seconds)
SREW+C(β)	742
SCA	811
Grass	838
Dolly	828
Mantri	860

jobs. GRASS also provides a unified solution for normal jobs. One fundamental limitation of the detection approach is that it may not be responsive enough for short tasks as one needs to wait for the collection of enough samples while monitoring the progress of tasks.

To avoid the extra delay caused by straggler detection, cloning approach was proposed in [3]. Dolly has been shown to outperform many existing straggler-detection-based approach including Mantri and LATE [37]. This also motivates us to adopt the cloning approach in this paper. Dolly relies on cloning very small job in a greedy manner to mitigate the straggler-effect and is based on simple heuristics. As a comparison, we develop a stochastic optimization framework to determine if cloning is needed for each job.

Recently, Ren *et al.* proposed Hopper [28] to jointly design task redundancy algorithm with job scheduling strategy. Hopper can immediately schedule a redundant copy once the progress rate of a task is detected to be slow. Nevertheless, Hopper does not provide any competitive performance guarantee and has several drawbacks that can degrade the cluster performance. Firstly, Hopper is non-work-conserving since its scheduler may keep a computing slot idle as a reservation for a future straggler while other tasks already queue up for computation slots. Secondly, the virtual job size is computed based on the number of tasks only without considering the task service time. By contrast, our scheduler is work-conserving and allocates machines to jobs based on the effective workload which also accounts for the effect of task service times. More importantly, our algorithms provide competitive performance bound under both offline and online settings. Readers may refer to Fig. 1 for a detailed comparison.

The design of job scheduling algorithms for MapReduce clusters has been an active research area lately. In particular, [7], [35] and [8] derived performance bounds for minimizing the total job completion time instead of job flowtime. Tan *et al.* designed the *Coupling scheduler* in [30], which mitigates the starvation problem caused by reduce tasks in large jobs. The SRPT scheduler has been studied extensively in traditional parallel scheduling literature. Moreover, SRPT has been proven to be $(1+\epsilon)$-speed $\frac{4}{\epsilon}$-competitive for total job flowtime on multiple identical machines under the single-task-per-job case [13]. It is therefore not surprising to see follow-up work, e.g. [22], [24] and [38] which extend the SRPT scheduler to minimize the total job flowtime in MapReduce-like systems. However, all of these studies ([7], [8], [22], [24], [30], [35], [38]) assume accurate knowledge of task durations and hence do not support extra copies of running tasks to be scheduled dynamically.

In our online algorithm design, we use the method of machine sharing among different jobs. The principle of

machine sharing is motivated by the work in [11], [12], [14] where machines are shared among the latest jobs arriving at the cluster (i.e., LAPS). One difference of SREW+C(β) is that it shares the machines among jobs with the smallest remaining workload. Moreover, SREW+C(β) also performs cloning for the tasks according to machine availability.

Potential function analysis is a widely adopted approach for deriving bounds for competitive performance ratio of online scheduling schemes with resource augmentation. Representative works include [11], [12] and [16]. In this paper, we also take the potential function analysis approach. However, task-cloning considerations require us to design a new potential function which is quite different from the ones used by the existing works. In particular, our new potential function incorporates the smallest-remaining-workload scheduling mechanism while considering task cloning. In contrast, the potential functions defined in [11], [12], [16] are based on the latest-arrival-first scheduling principle.

7 Conclusions and Future Directions

In this paper, we mitigate the challenge of variability in machine service capacity in a MapReduce cluster by combining task cloning and job scheduling. Our primary goal and contribution are to design both offline and online scheduling algorithms with competitive performance bounds, and to advance the modeling techniques for stochastic job scheduling with machine variability. Our proposed algorithms can be readily implemented in form of extensions of open-source frameworks like MapReduce/Hadoop YARN [31]. As future work, we plan to extend our scheduling algorithms for MapReduce jobs to other parallel computational frameworks which support more general multi-stage task dependencies, e.g., Spark [36]. We will also design approximate algorithms with competitive performance bounds for job scheduling with service variability and multi-dimensional resource requirements.

References

[1] Apache. http://hadoop.apache.org, 2013.

[2] S. Aalto, A. Penttinen, P. Lassila, and P. Osti. On the optimal trade-off between SRPT and opportunistic scheduling. In *SIGMETRICS*, 2011.

[3] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Effective straggler mitigation: Attack of the clones. In *NSDI*, April 2013.

[4] G. Ananthanarayanan, M. C.-C. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu. Grass: Trimming stragglers in approximation analytics. In *NSDI*, April 2014.

[5] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoic, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in MapReduce clusters using mantri. In *OSDI*, October 2010.

[6] N. Avrahami and Y. Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA*, 2003.

[7] H. Chang, M. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee. Scheduling in MapReduce-like systems for fast completion time. In *Proceedings of IEEE Infocom*, March 2011.

[8] F. Chen, M. Kodialam, and T. Lakshman. Joint scheduling of processing and shuffle phases in MapReduce systems. In *Proceedings of IEEE Infocom*, March 2012.

[9] Q. Chen, C. Liu, and Z. Xiao. Improving MapReduce performance using smart speculative execution strategy. *IEEE Transactions on Computers*, PP(99), January 2013.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, December 2004.

[11] J. Edmonds and K. Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *SODA*, January 2009.

[12] K. Fox, S. Im, and B. Moseley. Energy efficient scheduling of parallelizable jobs. In *SODA*, January 2013.

[13] K. Fox and B. Moseley. Online scheduling on identical machines using SRPT. In *SODA*, January 2011.

[14] A. Gupta, S. Im, R. Krishnaswamy, B. Moseley, and K. Pruhs. Scheduling jobs with varying parallelizability to reduce variance. In *SPAA*, June 2010.

[15] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello. Modeling and tolerating heterogeneous failures in large parallel system. In *SC*, 2011.

[16] S. Im, B. Moseley, and K. P. an dEric Torng. Competitively scheduling tasks with intermediate parallelizability. In *SPAA*, June 2014.

[17] S. Im, B. Moseley, K. Pruhs, and E. Torng. Competitively scheduling tasks with intermediate parallelizability. In *SPAA*, June 2014.

[18] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Eurosys*, March 2007.

[19] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *Proceedings of FOCS*, October 1995.

[20] R. Keener. A note on the variance of a stopping time. *The Annals of Statistics*, 15:1709–1712, 1987.

[21] G. F. Lawler. *Introduction to Stochastic Processes, Second Edition*. Chapman and Hall/CRC, 2006.

[22] M. Lin, L. Zhang, A. Wierman, and J. Tan. Joint optimization of overlapping phases in MapReduce. In *IFIP Performance*, 2013.

[23] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, January 2005.

[24] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlos. On scheduling in Map-Reduce and flow-shops. In *SPAA*, June 2011.

[25] M. L. Pinedo. *Theory, Algorithms, and Systems*. Springer Publishing Company, 2008.

[26] J. Pruyne and M. Livny. Managing checkpoints for parallel programs. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 140–154. Springer, 1996.

[27] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces. http://code.google.com/p/googleclusterdata, May 2011.

[28] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu. Hopper: Decentralized speculation-aware cluster scheduling at scale. In *Sigcomm*, August 2015.

[29] M. S. Squillante. On the benefits and limitations of dynamic partitioning in parallel computer systems. In *Job Scheduling Strategies for Parallel Processing*, pages 219–238. Springer-Verlag, 1995.

[30] J. Tan, X. Meng, and L. Zhang. Delay tails in MapReduce scheduling. In *SIGMETRICS*, June 2012.

[31] V. K. Vavilapalli, A. C. Murthy, C. Douglas, and M. S. Agarwal. Apache Hadoop YARN: yet another resource negotiator. In *SOCC*, October 2013.

[32] H. Xu and W. C. Lau. Optimization for speculative execution in a MapReduce-like cluster. In *Proceedings of Infocom*, April 2015.

[33] H. Xu and W. C. Lau. Task-cloning algorithms in a MapReduce cluster with competitive performance bounds. In *ICDCS*, June 2015.

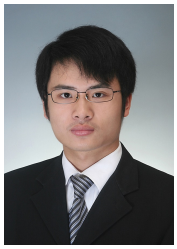
[34] H. Xu and W. C. Lau. Optimization for speculative execution in big data processing clusters. *IEEE Transactions on Parallel and Distributed Systems*, doi:10.1109/TPDS.2016.2564962(1), 2016.

[35] Y. Yuan, D. Wang, and J. Liu. Joint scheduling of MapReduce jobs with servers: Performance bounds and experiments. In *IEEE Infocom*, 2014.

[36] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, 2010.

[37] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving Mapreduce performance in heterogeneous environments. In *OSDI*, December 2008.

[38] Y. Zheng, N. Shroff, and P. Sinha. A new analytical technique for designing provably efficient MapReduce schedulers. In *Proceedings of IEEE Infocom*, April 2013.



Huanle Xu received his BSc(Eng) degree from the Department of Information Engineering, Shanghai Jiao Tong University (SJTU) in 2012 and his Ph.D. degree from the Department of Information Engineering, The Chinese University of Hong Kong (CUHK) in 2016. His primary research interests focus on Job Scheduling and Resource Allocation in Cloud Computing, Decentralized social networks, Parallel Graph Algorithms and Machine Learning. Huanle is also interested in designing wonderful algorithms for real applications and practical systems using mathematical tools.

Wing Cheong Lau is currently an Associate Professor in the Department of Information Engineering and the Director of the Mobile Technologies Center at The Chinese University of Hong Kong. Wing received his BSc(Eng) degree from The University of Hong Kong and MS and PhD degrees in Electrical and Computer Engineering from The University of Texas at Austin. From 1997 to 2004, he was a Member of the Technical Staff of the Performance Analysis Department at Bell Laboratories in Holmdel, New Jersey, where he conducted research in networking systems design and performance analysis. Wing joined Qualcomm, San Diego, California, in 2004 as a Senior Staff Member conducting research on Mobility Management Protocols for the Next Generation Wireless Packet Data Networks. He also contributed actively to the standardization of such protocols in the Internet Engineering Task Force (IETF) and 3GPP2. Wing is a Senior Member of IEEE and a member of ACM and Tau Beta Pi. He is/has been a Technical Program Committee Member of various international conferences, including ACM Sigmetrics, MobiHoc, IEEE Infocom, SECON, ICC, Globecom, WCNC, VTC and ITC. He also served as the Guest Editor for the Special Issue on High-Speed Network Security of the IEEE Journal of Selected Areas in Communications (JSAC). Wing holds 18 US patents with a few more pending. He has published more than 100 scientific papers in premier international journals and conferences. Wings recent research interests include Online Social Network Security and Privacy, Resource Allocation for Big Data Processing Systems, as well as the design and analysis of algorithms for Decentralized Online Social Networks.

Zhibo Yang is currently a Research Assistant at Department of Information Engineering, The Chinese University of Hong Kong. His research interests include computer vision, machine learning and big data processing. He is also passionate about bringing AI technologies into real products and applications. He got his MPhil degree from the same department under the supervision of Prof. W.C. Lau and Prof. C.C. Loy in fall 2016. Before that, he obtained his bachelor degree at software engineering from Harbin Institute of Technology in 2014.

Gustavo de Veciana (S'88-M'94-SM'01-F'09) received his B.S., M.S, and Ph.D. in electrical engineering from the University of California at Berkeley in 1987, 1990, and 1993 respectively, and joined the Department of Electrical and Computer Engineering where he is currently a Cullen Trust Professor of Engineering. He served as the Director and Associate Director of the Wireless Networking and Communications Group (WNCG) at the University of Texas at Austin, from 2003-2007. His research focuses on the analysis and design of communication and computing networks; data-driven decision-making in man-machine systems, and applied probability and queueing theory. Dr. de Veciana served as editor and is currently serving as editor-at-large for the IEEE/ACM Transactions on Networking. He was the recipient of a National Science Foundation CAREER Award 1996 and a co-recipient of five best paper awards including: IEEE William McCalla Best ICCAD Paper Award for 2000, Best Paper in ACM TODAES Jan 2002-2004, Best Paper in ITC 2010, Best Paper in ACM MSWIM 2010, and Best Paper IEEE INFOCOM 2014. In 2009 he was designated IEEE Fellow for his contributions to the analysis and design of communication networks. He currently serves on the board of trustees of IMDEA Networks Madrid.

Hanxu Hou received the B.Eng. degree in Information Security from Xidian University, Xian, China, in 2010, and Ph.D. degrees in the Dept. of Information Engineering from The Chinese University of Hong Kong in 2015 and School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School. He is now an Assistant Professor of Dongguan University of Technology. His research interests include erasure coding, coding for distributed storage systems and resource allocation in Cloud Computing.