Online Job Scheduling with Redundancy and Opportunistic Checkpointing: A Speedup-Function-Based Analysis

Huanle Xu, *Member, IEEE,* Gustavo de Veciana, *Fellow, IEEE,*
Wing Cheong Lau, *Senior Member, IEEE,* Kunxiao Zhou

Abstract—In a large-scale computing cluster, the job completions can be substantially delayed due to two sources of variability, namely, variability in the job size and that in the machine service capacity. To tackle this issue, existing works have proposed various scheduling algorithms which exploit redundancy wherein a job runs on multiple servers until the first completes. In this paper, we explore the impact of variability in the machine service capacity and adopt a rigorous analytical approach to design scheduling algorithms using redundancy and checkpointing. We design several online scheduling algorithms which can dynamically vary the number of redundant copies for jobs. We also provide new theoretical performance bounds for these algorithms in terms of the overall job flowtime by introducing the notion of a speedup function, based on which a novel potential function can be defined to enable the corresponding competitive ratio analysis. In particular, by adopting the online primal-dual fitting approach, we prove that our SRPT+R Algorithm in a non-multitasking cluster is $(1+\epsilon)$-speed, $O(\frac{1}{\epsilon})$-competitive. We also show that our proposed Fair+R and LAPS+R(β) Algorithms for a multitasking cluster are $(4+\epsilon)$-speed, $O(\frac{1}{\epsilon})$-competitive and $(2+2\beta+2\epsilon)$-speed $O(\frac{1}{\beta\epsilon})$-competitive respectively. We demonstrate via extensive simulations that our proposed algorithms can significantly reduce job flowtime under both the non-multitasking and multitasking modes.

Index Terms—Job Scheduling, Redundancy, Optimization, Competitive Analysis, Dual-Fitting, Potential Function

1 Introduction

JOB traces from large-scale computing clusters indicate that the completion time of jobs can vary substantially [8], [9]. This variability has two sources: variability in the job processing requirements and variability in machine service capacity. The job profiles in production clusters also become increasingly diverse as small latency-sensitive jobs coexist with large batch processing applications which take hours to months to complete [51]. With the size of today's computing clusters continuing to grow, component failures and resource contention have become a common phenomenon in cloud infrastructure [25], [33]. As a result, the rate of machine service capacity may fluctuate significantly over the lifetime of a job. The same job may experience a far higher response time when executed at a different time on the same server [21]. These two dimensions of variability make efficient job scheduling for fast response time (also referred to as job flowtime) over large-scale computing clusters challenging.

To tackle variability in job processing requirements, various schedulers have been proposed to provide efficient resource sharing among heterogeneous applications. Widely deployed schedulers to-date include the Fair scheduler [3] and the Capacity scheduler [2]. It is well known that the Shortest Remaining Processing Time scheduler (SRPT) is optimal for minimizing the overall/ average job flowtime [19] on a single machine in the clairvoyant setting, i.e., when job processing times are known a priori. As such, many works have aimed to extend SRPT scheduling to yield efficient scheduling algorithms in the multiprocessor setting with the objective of reducing job flowtimes for different systems and programming frameworks [22], [35], [36], [53]. Under SRPT, job's residual precessing times are known to the job scheduler upon arrival and smaller jobs are given priority. However, if only the distribution of job sizes is known, it is shown in [4] that, Gittins index-based policy is optimal for minimizing the expected job flowtime under the Poission job arrivals in the single-server case. The Gittins index depends on knowing the service already allocated to each job and gives priority to the job with the highest index.

To deal with component failures and resource contention, computing clusters are exploiting redundant execution wherein multiple copies of the same job execute on available machines until the first completes. With redundancy, it is expected that one copy of the same job might complete quickly to avoid long completion times. For the Google MapReduce system, it has been shown that redundancy can decrease the average job flowtime by 44% [17]. Many other cloud computing systems apply simple heuristics to use redundancy and they have proven to be effective at reducing job flowtimes via practical deployments, e.g., [1], [7], [9], [14], [17], [31], [52].

Recently, researchers have started to investigate the effectiveness of scheduling redundant copies from a queuing perspective [15], [21], [38], [39], [42], [45]. These works

- *Huanle Xu and Kunxiao Zhou are with the School of Computer Science and Network Security, Dongguan University of Technology, Dongguan, Guangdong. E-mail: {xuhl,zhoukx}@dgut.edu.cn.*
- *Gustavo de Veciana is with the of Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA. E-mail: gustavo@ ece.utexas.edu.*
- *Wing Cheong Lau is with the Department of Information Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. E-mail: wclau@ie.cuhk.edu.hk.*

Part of this work has been presented in IEEE Infocom 2017.

assume a specific distribution of the job execution time where jobs follow the same distribution. However, they do not characterize the major cause leading to the variance of the job response time, namely, whether the variance is due to variability of job size or to variability in machine service capacity. In fact, if there is no variability in the machine service capacity, making multiple copies of the same job may not help and redundancy is a waste of resource.

To overcome the aforementioned limitations, we have developed a stochastic framework in our previous work [49] to explore the impact of variability in the machine service capacity. In this framework, the service capacity of each machine over time is modeled as a stationary process. To take full advantage of redundancy, [49] allows checkpointing [37] to preempt, migrate and perform dynamic partitioning [43] on its running jobs. By checkpointing, we mean the runtime system of a cluster takes a snapshot of the state of a job in progress so that its execution can be resumed from that point in the case of subsequent machine failure or job preemption [10]. Upon checkpointing, the state of the redundant copy which has made the most progress is propagated and cloned to other copies. In other words, all the redundant copies of a job can be brought to that most advance state and proceed to execute from this updated state.

A fundamental limitation of [49] is that checkpointing needs to be done periodically when a job is being processed. Such a checkpointing mechanism would incur large overheads when the cluster size is large while the scheduler needs to make scheduling decisions quickly. To tackle this limitation, in this paper, we limit the total number of checkpointings for each job. Moreover, we only allow checkpointing to occur on a job only if there is an arrival to or departure from the system. As such, the resultant algorithms are more scalable and applicable to real world implementations.

Most previous works studying job scheduling assume that clusters are working in the non-multitasking mode, i.e., each server (CPU Core) in the cluster can only serve one job at any time. However, multitasking is a reasonable model of current scheduling policies in CPUs, web servers, routers, etc [16], [44], [46]. In a multitasking cluster, each server may run multiple jobs simultaneously and jobs can share resources with different proportions. In this paper, we will also study scheduling algorithms, which determine checkpointing times, the number of redundant copies between successive checkpoints as well as the fraction of resource to be shares in both of the multitasking and non-multitasking settings.

Our Results

For non-multitasking clusters, we propose the SRPT+R algorithm where redundancy is used only when the number of active jobs is less than the number of servers. For clusters allowing multitasking, we first design the Fair+R Algorithm, which shares resources near equally among existing jobs, with priority given to jobs which arrived most recently. We then extend Fair+R Algorithm to yield the LAPS+R(β) Algorithm, which only shares resources amongst a fixed fraction of the active jobs. In summary, this paper makes the following technical contributions:

- *New Framework.* We present the first optimization framework to address the job scheduling problem with redundancy, subject to limited number of checkpointing times. Our optimization problems consider both the multitasking and non-multitasking scenarios.
- *New Techniques.* We introduce the notion of speedup function in both the multitasking and non-multitasking cases. Thanks to this, we develop a new dual-fitting approach to bound the competitive performance for both SRPT+R and Fair+R. Based on the speedup function, we also design a novel potential function accounting for redundancy to analyze the performance of LAPS+R(β) in the multitasking setting. By changing the speedup function, one can readily apply our dual-fitting approach as well as the potential function analysis to other resource allocation problems in the multi-machine setting with/ without multitasking.
- *New Results.* Under our optimization framework, SRPT+R achieves a much tighter competitive bound than other SRPT-based redundancy algorithms under different settings, e.g., [49]. Moreover, LAPS+R(β) is the first one to address the redundancy issue among those algorithms which work under the multitasking mode.

The rest of this paper is organized as follows. After reviewing the related work in Section 2, we introduce our system model and optimization framework in Section 3. In Section 4, we present SRPT+R and its performance bound in a non-multitasking cluster. We proceed to introduce the design and analysis for both Fair+R and LAPS+R(β) under the multitasking mode in Section 5. Before concluding our work in Section 7, we conduct several numerical studies in Section 6 to evaluate our proposed algorithms.

2 Related Work

In this section, we begin by giving a brief introduction to existing work on job schedulers. Then, we review the related work on redundancy schemes in large-scale computing clusters presented by priori research from the industry and academia.

The design of job schedulers for large-scale computing clusters is currently an active research area [12], [13], [35], [36], [50], [53]. In particular, several works have derived performance bounds towards minimizing the total job completion time [12], [13], [50] by formulating an approximate linear programming problem. By contrast, [34] shows that there is a strong lower bound on any online randomized algorithm for the job scheduling problem on multiple unit-speed processors with the objective of minimizing the overall job flowtime. Based on this lower bound, some works extend the SRPT scheduler to design algorithms that minimize the overall flowtimes of jobs which may consist of multiple small tasks with precedence constraints [35], [36], [50], [53]. The above work was conducted in the clairvoyant setting, i.e., the job size is known once the job arrives. For the non-clairvoyant setting, [26], [27], [28] design several multitasking algorithms under which machines are allocated to all jobs in the system and priorities are given to jobs which arrive most recently. All of the above studies assume accurate knowledge of machine service capacity and hence do not address dynamic scheduling of redundant copies for a job.

Production clusters and big data computing frameworks have adopted various approaches to use redundancy for running jobs. The initial Google MapReduce system launches redundant copies when a job is close to its completion [17]. Hadoop adopts another solution called LATE, which schedules a redundant copy for a running task only if its estimated progress rate is below certain threshold [1]. By comparison, Microsoft Mantri [9] schedules a new copy for a running task if its progress is slow and the total resource consumption is expected to decrease once a new redundant copy is made.

Researchers have proposed different schemes to take advantage of redundancy via more careful designs. For example, [14] proposes a smart redundancy scheme to accurately estimate the task progress rate and launch redundant copies accordingly. The authors in [7] propose to use redundancy for very small jobs when the extra loading is not high. As an extension to [7], they further develops GRASS [8], which carefully schedules redundant copies for approximation jobs. Moreover, [41] proposes Hopper to allocate computing slots based on the virtual job size, which is larger than the actual size. Hopper can immediately schedule a redundant copy once the progress rate of a task is detected to be slow. No performance characterization has been developed for these heuristics.

In our previous work, we have developed several optimization frameworks to study the design of scheduling algorithms utilizing redundancy [47], [48]. The proposed algorithms in [47] require the knowledge of exact distribution of the task response time. We also analyze performance bounds of the proposed algorithm which extends the SRPT Scheduler in [48] by adopting the potential function analysis. A fundamental limitation is that these resultant bounds are not scalable as they increase linearly with the number of machines. Recently, [20] proposes a simple model to address both machine service variability and job size variability. However, [20] only considers the FIFO scheduling policy on each server to characterize the average job response time from a queuing perspective.

Another body of research related to this paper focuses on the study of scheduling algorithms for jobs with intermediate parallelizability. In these works, e.g., [5], [11], [18], [24], [30], jobs are parallelizable and the service rate can be arbitrarily scaled. In particular, Samuli *et al.* present several optimal scheduling policies for different capacity regions in [5] but for the transient case only. [18] [11] and [24] propose similar multitasking algorithms for jobs wherein priorities are given to jobs which arrive the most recently. These works develop competitive performance bounds with respect to the total job flowtime by adopting potential function arguments. [30] also provides a competitive bound for the SRPT-based parallelizable algorithm in the multitasking setting. One limitation of [30] is that the resultant bound is potentially very large. By contrast, this paper is motivated by the setting where there is variability in the machine service capacity.

For the analysis of SRPT+R algorithm in Section 4.2, and Fair+R algorithm in 5.1, we adopt the dual fitting approach. Dual fitting was first developed by [6], [23] and is now widely used for the analysis of online algorithms [27], [28]. In particular, [6] and [27], [28] address linear objectives, and use the dual-fitting approach to derive competitive bounds for traditional scheduling algorithms without redundancy. By contrast, [23] focuses on a convex objective in the multitasking setting. By comparison, this paper includes integer constraints associated with the non-multitasking mode. Moreover, our setting of dual variables is novel in the sense that it deals with the dynamical change of job flowtime across multiple machines where other settings of dual variables can only deal with the change of job flowtime on one single machine.

We apply the potential function analysis to bound the performance of LAPS+R(β) in Section 5. Potential function is widely used to derive performance bounds with resource augmentation for online parallel scheduling algorithms e.g., [18], [30]. However, since we need to deal with redundancy and checkpointing, the design of our potential function is totally different from that in [18] and [30] which only address sublinear speedup.

While this paper adopts a framework similar to the one in [49] to model machine service variability, it differs from [49] in two major aspects. Firstly, the requirement of limiting the the total number of checkpointings results in a very different optimization problem which is much more difficult to solve from the one in [49]. To tackle this challenge, in this paper, we adopt both the dual fitting approach and potential function analysis to make approximations and bound the competitive performance. By contrast, [49] only applies the potential function analysis to derive performance bounds. Secondly, the current paper considers both the multi-tasking mode and non-multitasking mode to design corresponding online scheduling algorithms using redundancy. By contrast, the scheduling algorithms proposed in [49] can only work under the non-multitasking mode.

3 System Model

Consider a computing cluster which consists of M servers (machines) where the servers are indexed from 1 to M. Job j arrives to the cluster at time a_j and the job arrival process, $(a_1, a_2, \cdots, a_N)$, is an arbitrary deterministic time sequence. In addition, job j has a workload which requires p_j units of time to complete when processed on a machine at *unit* speed. Job j completes at time c_j and its flowtime f_j, is denoted by $f_j = c_j - a_j$. In this paper, we focus on minimizing the overall job flowtime, i.e., $\sum_{j=1}^{N} f_j$.

The service capacity of machines are assumed to be identically distributed random processes with stationary increments. To be specific, we let $S_i = (S_i(t)|t \geq 0)$ be a random process where $S_i(t,\tau) = S_i(\tau) - S_i(t)$ denote the *cumulative* service delivered by machine i in the interval $(t,\tau]$. The service capacity of a machine has unit mean speed and a peak rate of Δ, so for all $\tau > t \geq 0$, we have $S_i(t,\tau] \leq (\tau - t)\cdot\Delta$ almost surely and $\mathbb{E}[S_i(t,\tau]] = \tau - t$.

In this paper, our aim is to mitigate the impact of service variability by (possibly) varying the number of redundant copies with appropriate checkpointing. Checkpointing can make the most out of the allocated resources, i.e., start the processing of the possibly redundant copies at the most advanced state amongst the previously executing copies. In fact, we shall make the following assumption across the system:

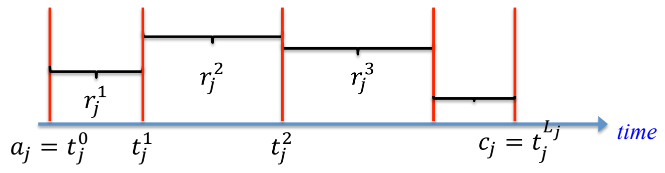

Fig. 1. The service process of job j.

Assumption 1. *A job j can be checkpointed only if there is an arrival to, or departure from, the system.*

Remark 1. *We refer to Assumption 1 as a scalability assumption as it limits the checkpointing overheads in the system.*

Below, we will first introduce a service model where each server can only serve one job at a time. In Section 3.2, we will discuss a service model which supports multitasking, i.e., a server can execute multiple jobs simultaneously.

3.1 Job processing in a Non-Multitasking Cluster

As illustrated in Fig. 1, one can view the service process of job j in a non-multitasking cluster by dividing its service period (from its arrival to its completion) into several subintervals, i.e., $\{(t_j^{k-1}, t_j^k]\}_k$ where t_j^k denotes the time when the kth checkpointing of job j occurs. The job arrival and completion times are also considered as checkpointing times, i.e., $t_j^0 = a_j$ and $t_j^{L_j} = c_j$ if job j experiences $(L_j + 1)$ checkpoints. During in $(t_j^{k-1}, t_j^k]$, job j is running on r_j^k redundant servers. Thus, together $\boldsymbol{t_j} = (t_j^k | k = 0, 1, \cdots, L_j)$ and $\boldsymbol{r_j} = (r_j^k | k = 1, 2, \cdots, L_j)$ capture the checkpoint times and the scheduled redundancy for job j.

We will let $g(r,t)$ denote the cumulative service delivered to a job on r redundant machines and checkpointed at the end of an interval of duration t. Clearly, $g(r,t)$ is equivalent to the amount of work processed by the redundant copy which has made the most progress. In this paper, we make the following assumption for $g(r,t)$:

Assumption 2. *We shall model (approximate) the cumulative service capacity under redundant execution, $g(r,t)$, by its mean, i.e.,*

$$g(r,t) = \mathbb{E}\Big[\max_{i=1,2,\cdots,r} S_i(0,t]\Big]. \tag{1}$$

Remark 2. *Assumption 2 essentially replaces the service capacity of the system with the mean but accounts for the mean gains one might expect when there are redundant copies executed.*

The following lemmas illustrate two important properties of $g(r,t)$:

Lemma 1. *For a fixed t, $\{g(r,t)\}_r$ is a concave sequence, i.e., $g(r,t) - g(r-1,t) \le g(r-1,t) - g(r-2,t)$.*

Proof. Let $H_r(0,t] = \max_{i=1,2,\cdots,r} S_i(0,t]$ and define $F(x,t)$ as the cumulative distribution function of random variable $S_i(0,t]$ for a fixed t. Thus, we have $\Pr(H_r(0,t] \le x) = F^r(x,t)$ and $g(r,t) = \mathbb{E}[H_r(t)] = \int_0^\infty (1 - F^r(x,t))dx$, which further implies that:

$$\begin{aligned} g(r,t) - g(r-1,t) &= \int_0^\infty F^{r-1}(x,t) \cdot (1 - F(x,t))dx \\ &\le \int_0^\infty F^{r-2}(x,t) \cdot (1 - F(x,t))dx \\ &= g(r-1,t) - g(r-2,t). \end{aligned} \tag{2}$$

This completes the proof. □

Lemma 1 states that the marginal increase of the mean service capacity in the number of redundant executions is decreasing.

Lemma 2. *For all $r \in \mathbb{N}$ and $r \le M$, $g(r,t) \le \min\{\Delta t, rt\}$.*

Proof. As shown in the proof of Lemma 1, $g(r,t) = \int_0^\infty (1 - F^r(x,t))dx$. Therefore, it follow that:

$$\begin{aligned} \int_0^\infty (1 - F^r(x,t))dx &= \int_0^\infty (1 - F(x,t)) \sum_{l=0}^{r-1} (F^l(x,t))dx \\ &\ge r \int_0^\infty (1 - F(x,t)) F^{r-1}(x,t)dx \\ &= r(g(r,t) - g(r-1,t)). \end{aligned} \tag{3}$$

which implies $g(r,t) \le \frac{r}{r-1} g(r-1,t)$. Hence, $g(r,t) \le rg(1,t)$. Moreover, we have $g(1,t) = \mathbb{E}[S_i(0,t]] = t$. Thus, we have:

$$g(n,t) \le nt. \tag{4}$$

Since $g_i(t) = S_i(0,t] \le \Delta t$, it follows that:

$$\mathbb{E}\Big[\max_{i=1,2,\cdots,n} \{g_i(t)\}\Big] \le \Delta t. \tag{5}$$

The result follows from (4) and (5). □

Lemma 2 states that the mean service capacity under redundant execution can grow at most linearly in the redundancy, rt, and is bounded by the peak service capacity of any single redundant copy, Δt.

Given Assumption 2, the last checkpoint time for job j, $t_j^{L_j}$, is also the completion time c_j and satisfies the following equation:

$$\sum_{k=1}^{L_j} g(r_j^k, t_j^k - t_j^{k-1}) = p_j. \tag{6}$$

In the sequel, we shall also make use of the speedup function, $h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t)$, defined as follows:

$$h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t) = \begin{cases} \frac{g(r_j^k, t_j^k - t_j^{k-1})}{t_j^k - t_j^{k-1}} & t \in (t_j^{k-1}, t_j^k], \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The speedup function captures the speedup that redundant execution is delivering in a checkpointing interval relative to a job execution on a unit speed machine. (6) can be reformulated in terms of the speedup as follows:

$$\int_{a_j}^{c_j} h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, \tau)d\tau = p_j. \tag{8}$$

Remark 3. *Note that the speedup depends, not only on the number of redundant copies being executed, but also, on all the times when checkpointing occurs. In this sense, $h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t)$ is not a causal function. However, in the following sections, $h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t)$ will be a convenient notation to study competitive performance bounds for our proposed algorithms.*

3.2 Job processing in a Multitasking cluster

With multitasking, a server can run several jobs simultaneously and the service a job receives on a server is proportional to the fraction of processing resource it is assigned.

We will model a cluster allowing multitasking as follows. Comparing with the service model in Subsection 3.1, we include another variable x_j^k, to characterize the fraction of resource assigned to job j in the kth subinterval, i.e.,

$(t_j^{k-1}, t_j^k]$. Here, we assume that job j shares the same fraction of processing resource on all the machines on which it is being executed. Let $\boldsymbol{x_j} = (x_j^k | k = 1, 2, \cdots, L_j)$ and we define another speedup function, $\hat{h}_j(\boldsymbol{t_j}, \boldsymbol{x_j}, \boldsymbol{r_j}, t)$, as follows:

$$\hat{h}_j(\boldsymbol{t_j}, \boldsymbol{x_j}, \boldsymbol{r_j}, t) = \begin{cases} x_j^k \cdot h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t) & t \in (t_j^{k-1}, t_j^k] \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Paralleling (8), the completion time of job j, c_j must satisfy the following equation:

$$\int_{a_j}^{c_j} \hat{h}_j(\boldsymbol{t_j}, \boldsymbol{x_j}, \boldsymbol{r_j}, \tau) d\tau = p_j \tag{10}$$

In the sequel, we will design and analyze algorithms under both the multitasking mode and the non-multitasking mode.

3.3 Competitive Performance Metrics

In this paper, we will study algorithms for scheduling, which involves determining checkpointing times, the number of redundant copies for jobs between successive checkpoints and in the multitasking setting the fraction of resource shares. Note that, when there is no variability in the machine's service capacity, our problem reduces to job scheduling on multiple unit-speed processors with the objective of minimizing the overall flowtime. This has been proven to be NP-hard even when preemption and migration are allowed and previous work [30], [32] 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.

Definition 1. *[32] 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 setup to bound the competitive performance of our proposed algorithms. With resource augmentation, the service capacity in each checkpointing interval under our algorithms is scaled by δ. Similarly, the value of the speedup functons, i.e., $h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t)$ and $\hat{h}_j(\boldsymbol{t_j}, \boldsymbol{x_j}, \boldsymbol{r_j}, t)$, under our algorithms is δ times that under the optimal algorithm of the same variables.

4 Algorithm Design in a Non-Multitasking Cluster

In a non-multitasking cluster, each server can only serve one job at any time. Before going to the details of algorithm design, we first state the optimal problem formulation. For ease of illustration, we let $\boldsymbol{y}_j = (\boldsymbol{t_j}, \boldsymbol{r_j}, L_j)$ denote the checkpointing trajectory of job j and $\boldsymbol{y} = (\boldsymbol{y}_j | j = 1, 2, \cdots, N)$ that for all jobs. Moreover, let $\mathbb{1}(A)$ denote the indicator function that takes value 1 if A is true and 0 otherwise. The optimal problem formulation is as follows:

$$\min_{\boldsymbol{y}} \sum_{j=1}^{N} (c_j - a_j) \tag{OPT}$$

such that (a), (b), (c), (d) are satisfied

(a) Job completion: The completion time of job j, c_j, satisfies: $\int_{a_j}^{c_j} h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t) dt = p_j, \ \ \forall j$.
(b) Resource constraint: The total number of redundant executions at any time $t \geq 0$ is no larger than the number of machines, M, i.e., $\sum_{j: a_j \leq t} \sum_{k=1}^{L_j} r_j^k \cdot \mathbb{1}(t \in (t_j^{k-1}, t_j^k]) \leq M, \ \ \forall t$.
(c) Checkpoint trajectory: The number of checkpoints for each job is between 2 and $2N$ since there are $2N$ job arrivals and departures, i.e., $L_j \in \{1, 2, \cdots, 2N-1\}$. The checkpoint times of job j, $\boldsymbol{t}_j$, satisfy: $\boldsymbol{t}_j \in \mathcal{T}_j^{L_j+1}$ where $\mathcal{T}_j^{L_j+1} = \{(t_0, t_1, \cdots, t_{L_j}) \in \mathbb{R}^{L_j+1} | a_j = t_0 < t_1 < \cdots < t_{L_j} = c_j\}$. Moreover, the number of redundant copies must be an integer, i.e., $\boldsymbol{r}_j \in \mathbb{N}^{L_j}$.
(d) Checkpointing overhead constraint: Job checkpoints must satisfy Assumption 1, i.e., for $0 \leq k \leq L_j$, $t_j^k \in \{a_j\}_j \cup \{c_j\}_j$.

Since the OPT problem is NP-Hard, we propose to design a heuristic to schedule redundant jobs, i.e., SRPT+R, which is a simple extension of the SRPT scheduler [19].

4.1 SRPT+R Algorithm and the performance guarantee

Let $p_j(t)$ denote the amount of the unprocessed work for job j at time t and $n(t)$ denote the number of active jobs at time t. In this section, we will assume without loss of generality that jobs have been ordered such that $p_1(t) \leq p_2(t) \leq \cdots \leq p_{n(t)}(t)$.

At a high level, the algorithm works as follows. When $n(t) \geq M$, the M jobs with smallest $p_j(t)$, i.e., Job 1 to M are each assigned to a server while the others wait. If $n(t) < M$, the job with the smallest $p_j(t)$, i.e., Job 1, is scheduled on $M - \lfloor \frac{M}{n(t)} \rfloor (n(t) - 1)$ machines and the others are scheduled on $\lfloor \frac{M}{n(t)} \rfloor$ machines each. Here, $\lfloor x \rfloor$ represents the largest integer which does not exceed x.

The corresponding pseudo-code is exhibited as Algorithm 1.

Our main result, characterizing the competitive performance of SRPT+R, is given in the following theorem:

Theorem 1. *SRPT+R is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive with respect to the total job flowtime.*

We will prove Theorem 1 by adopting the online dual fitting approach. The first step is to formulate a minimization problem which serves as an approximation to the optimal cost, OPT with a guarantee that the cost of the approximation is within a constant of OPT. We then formulate the dual problem for the approximation and exploit the fact that a feasible solution to this dual problem gives a lower bound on its cost, which in turn is a constant times the cost of the proposed algorithm.

Remark 4. *It is worth to note that, when there is no machine service variability, SRPT+R performs exactly the same as the traditional SRPT algorithm on multiple machines. As a result,*

Algorithm 1: SRPT+R Algorithm

```
1  while A job arrives at or departure from the system do
2      Sort the jobs in the order such that
       $p_1(t) \le p_2(t) \le \cdots \le p_{n(t)}(t)$ and count the
       number of redundant copies being executed for
       job $j$, $r_j$ ;
3      Initialize $M(t)$ to be the set of idle machines ;
4      if $n(t) < M$ then
5          for $j = 1, 2, \cdots, n(t)$ do
6              if $j = 1$ then
7                  $r_j(t) = M - (n(t)-1)\lfloor \frac{M}{n(t)} \rfloor$;
8              else
9                  $r_j(t) = \lfloor \frac{M}{n(t)} \rfloor$;
10             Checkpoint job $j$ and assigns its redundant
               executions to $r_j(t)$ machines which are
               uniformly chosen at random from
               $\{1, 2, \cdots, M\}$;
11     if $n(t) \ge M$ then
12         for $j = 1, 2, \cdots, n(t)$ do
13             if $j \le M$ then
14                 Checkpoint job $j$ and assign it to one
                   machine which is uniformly chosen at
                   random from $\{1, 2, \cdots, M\}$;
15             else
16                 Checkpoint job $j$;
```

our proposed dual fitting framework can also show that SRPT is $(1+\epsilon)$-speed, $(3+\frac{3}{\epsilon})$ competitive with respect to the overall job flowtime. When given small resource augmentation where $\epsilon \le \frac{1}{3}$, our result improves the recent result in [19], which states, SRPT on multiple identical machines is $(1+\epsilon)$-speed, $\frac{4}{\epsilon}$-competitive in terms of the overall job flowtime.

4.2 Proof of Theorem 1

To prove Theorem 1, we shall first both approximate the objective of OPT and relax Constraint (d) in OPT to obtain the following problem P1:

$$\min_{\boldsymbol{y}} \sum_{j=1}^{N} \int_{a_j}^{\infty} \frac{(t-a_j+2p_j)}{p_j} \cdot h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t)dt \tag{P1}$$

$$\text{s.t.} \int_{a_j}^{\infty} h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t)dt \ge p_j, \ \ \forall j,$$

$$\sum_{j:a_j \le t} \sum_{k=1}^{L_j} r_j^k \cdot \mathbb{1}(t \in (t_j^{k-1}, t_j^k]) \le M, \ \ \forall t,$$

$$L_j \in \{1, 2, \cdots, 2N-1\}, \ \ \boldsymbol{t_j} \in \mathcal{T}_j^{L_j+1}, \ \ \boldsymbol{r_j} \in \mathbb{N}^{L_j}, \ \forall j.$$

Let OPT denote the cost, i.e., the overall job flowtime, achieved by an optimal scheduling policy. The following lemma guarantees that the optimal cost of P1, denoted by $P1$, is not far from OPT.

Lemma 3. *$P1$ is upper bounded by $(1+2\Delta) \cdot OPT$, i.e., $P1 \le (1+2\Delta) \cdot OPT$.*

Let α_j and $\beta(t)$ denote the Lagrangian dual variables corresponding to the first and second constraint in P1 respectively. Define $\boldsymbol{\alpha} = (\alpha_j | j = 1, 2, \cdots, N)$ and $\boldsymbol{\beta} = (\beta(t) | t \in \mathbb{R}^+)$. The Lagrangian function associated with P1 can be written as:

$$\Phi(\boldsymbol{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{j=1}^{N} \int_{a_j}^{\infty} \frac{(t-a_j+2p_j)}{p_j} \cdot h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t)dt$$
$$+ \int_0^{\infty} \beta(t) \Big(\sum_{j:a_j \le t} \sum_{k=1}^{L_j} r_j^k \mathbb{1}(t \in (t_j^{k-1}, t_j^k]) - M \Big) dt$$
$$- \sum_{j=1}^{N} \alpha_j \Big(\int_{a_j}^{\infty} h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t)dt - p_j \Big),$$

with the dual problem for P1 given by:

$$\max_{\boldsymbol{\alpha} \ge \boldsymbol{0}, \boldsymbol{\beta} \ge \boldsymbol{0}} \ \min_{\boldsymbol{y}} \ \ \Phi(\boldsymbol{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{D1}$$
$$\text{s.t. } L_j \in \{1, 2, \cdots, 2N-1\}, \ \boldsymbol{r}_j \in \mathbb{N}^{L_j}, \ \boldsymbol{t}_j \in \mathcal{T}_j^{L_j+1}.$$

Applying weak duality theory for continuous programs [40], we can conclude that the optimal value to D1 is a lower bound for $P1$. Moreover, the objective of D1 can be reformulated as shown in (7).

Still it is difficult to solve D1 as it involves a minimization of a complex objective function of integer valued variables. However, it follows from Lemma 2 that $r_j^k \ge \mathbb{1}(t \in (t_j^{k-1}, t_j^k]) \cdot h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t)$ for all j and $t \ge a_j$, thus, we have that,

$$\sum_{k=1}^{L_j} r_j^k \mathbb{1}(t \in (t_j^{k-1}, t_j^k]) \ge \sum_{k=1}^{L_j} \mathbb{1}(t \in (t_j^{k-1}, t_j^k]) h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t)$$
$$= h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t).$$

Therefore, it can be readily shown that the second term in the R.H.S of $\Phi(\boldsymbol{y}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ in (7) is lower bounded by:

$$\int_0^{\infty} \sum_{j:a_j \le t} \Big[\Big(\frac{t-a_j}{p_j} + 2 - \alpha_j + \beta(t) \Big) \cdot h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t) \Big] dt.$$

As a result, for a fixed α_j and $\beta(t)$ such that for all $t \ge a_j$

$$\frac{t-a_j}{p_j} + 2 - \alpha_j + \beta(t) \ge 0, \tag{8}$$

the minimum of $\Phi(\boldsymbol{y}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ can be attained by setting all $\boldsymbol{r_j}$ to $\boldsymbol{0}$ and $\boldsymbol{t_j} = (a_j, c_j)$. In this solution, there are no other checkpoints for job j other than the job arrival and departure.

Therefore, restricting $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ to satisfy (8) would give a lower bound on D1 and results in the following optimization problem:

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\beta}} \ \ \sum_j \alpha_j p_j - M \int_0^{\infty} \beta(t)dt \tag{P2}$$
$$\textit{s.t.} \ \ \alpha_j - \beta(t) \le \frac{t-a_j}{p_j} + 2, \ \ \forall j, \ \ t \ge a_j,$$
$$\alpha_j \ge 0, \ \ \forall j$$
$$\beta(t) \ge 0, \ \ \forall t$$

Based on Lemma 3, we conclude that $P2 \le P1 \le (1+2\Delta) \cdot OPT$ where $P2$ is the optimal cost for P2.

Next, we shall find a setting of the dual variables in P2 such that the corresponding objective is lower bounded by $O(\epsilon) \cdot SR$ under a $(1+\epsilon)$-speed resource augmentation.

$$\Phi(\boldsymbol{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_j \alpha_j p_j - M \int_0^\infty \beta(t) dt + \int_0^\infty \sum_{j: a_j \le t} \Big[\big(\frac{t - a_j}{p_j} + 2 - \alpha_j\big) h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t) + \beta(t) \sum_{k=1}^{L_j} r_j^k \cdot \mathbb{1}(t \in (t_j^{k-1}, t_j^k]) \Big] dt. \quad (7)$$

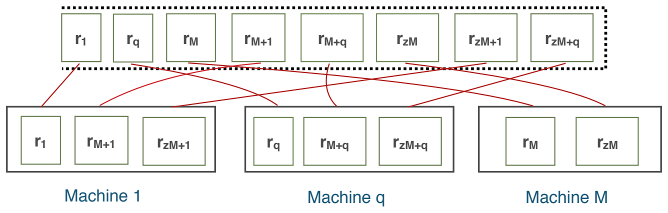

Fig. 2. The scheduling process of SRPT at time a_j where $n(a_j) = zM + q$ and there are no further job arrivals after a_j. Jobs are sorted based on the remaining size, which is denoted by r_j for job j, i.e., $r_j = p_j(a_j)$. Jobs indexed by $kM + i$ for some integer valued k and i are assigned to machine i.

To achieve this, we first consider a pure SRPT scheduling process that does not exploit job redundancy. We then use this to motivate a setting of dual variables which feasible for P2. Finally, we show that the objective for this setting of dual variables is at least $O(\epsilon)$ times the cost of SRPT, which is also lower bounded by $O(\epsilon) \cdot SR$ since the cost of SRPT is no smaller than SR.

4.2.1 *Setting of dual variables*

Observe that SRPT+R and SRPT only differ when $n(t) < M$ and that when this is the case SRPT only assigns a single machine to each active job. Since SRPT+R maintains the same scheduling order and each job is scheduled with at least the same number of copies as SRPT, we conclude that the cost of SRPT, denoted by $SRPT$, is a lower bound for SR, where SR denotes the overall job flowtime achieved SRPT+R.

In this section, we let $n(t)$ and $p_j(t)$ denote the number of active jobs and the size of the remaining workload of job j under SRPT respectively.

Let $\Theta_j = \{k : a_k \le a_j \le c_k\}$, the set of jobs that are active when job j arrives and $A_j = \{k \ne j : k \in \Theta_j$ and $p_k(a_j) \le p_j\}$, i.e., jobs whose residual processing time upon job j's arrival is less than job j's processing requirement. Define $\rho_j = |A_j|$, we shall set the dual variables as follows:

$$\alpha_j = \frac{1}{(1+\epsilon)p_j} \sum_{k=1}^{\rho_j} \Big(\lfloor \frac{n(a_j) - k}{M} \rfloor - \lfloor \frac{n(a_j) - k - 1}{M} \rfloor \Big) p_k(a_j) + \frac{1}{1+\epsilon} \Big(\lfloor \frac{n(a_j) - \rho_j - 1}{M} \rfloor + 1 \Big), \quad (9)$$

where $\epsilon > 0$ and

$$\beta(t) = \frac{1}{(1+\epsilon)M} n(t). \quad (10)$$

We show in the following lemma that this setting of dual variables is feasible.

Lemma 4. *The setting of dual variables in* (9) *and* (10) *is feasible to P2.*

Proof. Since $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are both nonnegative, it only remains to show $\alpha_j - \beta(t) \le \frac{t - a_j}{p_j} + 2$ for all j and $t \ge a_j$. First, α_j can be represented as follows:

$$\alpha_j = \frac{\sum_{k=0}^{z} p_{kM+q}(a_j) \mathbb{1}(kM + q \le \rho_j)}{(1+\epsilon)p_j} + \frac{\Big(\lfloor \frac{n(a_j) - \rho_j - 1}{M} \rfloor + 1 \Big)}{1+\epsilon}. \quad (11)$$

For ease of illustration, let Ω_1 and Ω_2 denote the two terms on the R.H.S of (11) respectively.

If $n(a_j) \le M$, we have $\alpha_j = \frac{1}{1+\epsilon}$ and the result follows. Therefore, we only consider $n(a_j) = zM + q > M$ and analyze the following three cases:

Case I: All the jobs in Θ_j have completed at time t. As depicted in Fig. 2, if there are no job arrivals after time a_j, then, jobs indexed by $km + q$ where k is a non-negative integer are all processed on Machine q. Since the service capacity of Machine q is $(t - a_j)$ during $(a_j, t]$, thus, it follows that,

$$t - a_j \ge \frac{1}{1+\epsilon} \sum_{k=0}^{z} p_{kM+q}(a_j). \quad (12)$$

In contrast, if there are other job arrivals after time a_j, Machine q needs to process an amount of work which exceeds $\sum_{k=0}^{z} p_{kM+q}(a_j)$, therefore, (12) still holds. Thus, we have that,

$$\begin{aligned} \frac{t - a_j}{p_j} - \Omega_1 &\ge \frac{\sum_{k=0}^{z} p_{kM+q}(a_j) \mathbb{1}(kM + q \ge \rho_j + 1)}{(1+\epsilon)p_j} \\ &\ge \frac{\sum_{k=0}^{z} \mathbb{1}(kM + q \ge \rho_j + 1)}{1+\epsilon} = \Omega_2. \end{aligned} \quad (13)$$

Case II: The jobs indexed from 1 to κ in Θ_j have completed and $\kappa \le \rho_j$. Let $\kappa = z_1 M + q_1$. Similar to Case I, it follows that,

$$t - a_j \ge \frac{1}{1+\epsilon} \sum_{k=0}^{z_1} p_{kM+q_1}(a_j). \quad (14)$$

In addition, the number of active jobs, $n(t)$, is no less than $n(a_j) - \kappa$. Therefore, we have:

$$\begin{aligned} \alpha_j &\overset{(ii)}{\le} \frac{\sum_{k=0}^{z_1} p_{kM+q_1}(a_j) + \frac{\lceil \frac{\rho_j - \kappa}{M} \rceil}{1+\epsilon}}{(1+\epsilon)p_j} + \frac{\Big(\lfloor \frac{n(a_j) - \rho_j - 1}{M} \rfloor + 1 \Big)}{1+\epsilon} \\ &\overset{(iii)}{\le} \frac{t - a_j}{p_j} + \frac{1}{1+\epsilon} \lceil \frac{\rho_j - \kappa}{M} \rceil + \frac{\Big(\lfloor \frac{n(a_j) - \rho_j - 1}{M} \rfloor + 1 \Big)}{1+\epsilon} \\ &\le \frac{t - a_j}{p_j} + \frac{1}{1+\epsilon} \big(\lfloor \frac{n(a_j) - \kappa}{M} \rfloor + 2 \big) \\ &\le \frac{t - a_j}{p_j} + \beta(t) + 2, \end{aligned} \quad (15)$$

where $\lceil x \rceil$ denotes the smallest integer which is no less than x and (ii) is due to that $\Omega_1 \le \frac{1}{(1+\epsilon)p_j} \sum_{k=0}^{z_1} p_{kM+q_1}(a_j) + \frac{1}{1+\epsilon} \cdot \lceil \frac{\rho_j - \kappa}{M} \rceil$. (iii) is due to (14).

Case III: The jobs indexed from 1 to κ in Θ_j have completed and $\kappa > \rho_j$. In this case, (14) still holds. Moreover, we have that $\sum_{k=0}^{z_1} p_{kM+q_1}(a_j) \geq \Omega_1 + \lfloor \frac{\kappa-\rho_j}{M} \rfloor p_j$. Therefore, it follows that:

$$\alpha_j \leq \frac{t-a_j}{p_j} - \frac{1}{1+\epsilon}\lfloor \frac{\kappa-\rho_j}{M} \rfloor + \frac{1}{1+\epsilon}\lceil \frac{n(a_j)-\rho_j}{M} \rceil$$
$$\leq \frac{t-a_j}{p_j} + \frac{1}{1+\epsilon}(\lfloor \frac{n(a_j)-\kappa}{M} \rfloor + 2) \quad (16)$$
$$\leq \frac{t-a_j}{p_j} + \beta(t) + 2.$$

Thus, we conclude that, for all the three cases above, the constraint between α_j and $\beta(t)$ is well satisfied. □

4.2.2 Performance bound

To bound the cost of the dual variables which are set in (9) and (10), we first show the following lemma to quantify the total job flowtime under SRPT in the transient case where there are no job arrivals after time t.

Lemma 5. *When there are no job arrivals after time t, the overall remaining job flowtime under SRPT scheduling, $F(t)$, is given by:*

$$F(t) = \sum_{j=1}^{n(t)} (\lfloor \frac{n(t)-j}{M} \rfloor + 1) p_j(t). \quad (17)$$

Proof. In this proof, we shall not assume resource augmentation. Let $f_j(t)$ denote the remaining flowtime for job j at time t. Thus, the job completion time, c_j is equal to $f_j(t)+t$. Since we have indexed jobs such that $p_1(t) \leq p_2(t) \leq \cdots \leq p_{n(t)}(t)$, under SRPT, it follows that $c_1 \leq c_2 \leq \cdots \leq c_{n(t)}$. When $n(t) \leq M$, (17) follows immediately since all jobs can be scheduled simultaneously and $f_j(t)$ is equal to $p_j(t)$.

Let us then consider the case where $n(t) > M$. Let $n(t) = zM + q$ where $z \geq 1$, $0 \leq q \leq M-1$ and z, q are non-negative integers. We first show that for all k such that $M \leq k \leq n(t)$, the following result holds:

$$\sum_{j=k-M+1}^{k} f_j(t) = \sum_{j=1}^{k} p_j(t). \quad (18)$$

As illustrated in Fig. 3, at any time between t and c_1, there are $(k-M)$ jobs waiting to be processed among those k jobs which complete first. Hence, the accumulated waiting time in this period is $(k-M)f_1(t)$. Similarly, at any time between c_1 and c_2, there are $(k-M-1)$ jobs waiting to be processed and they contribute $(k-M-1)\cdot(c_2-c_1) = (k-M-1)\cdot(f_2(t)-f_1(t))$ waiting time. Hence, the total waiting time of the k jobs is given by:

$$\sum_{j=0}^{k-M-1} (k-M-j)\cdot(f_{j+1}(t)-f_j(t)) = \sum_{j=1}^{k-M} f_j(t). \quad (19)$$

Therefore, the total remaining flowtime for these k jobs is as follows:

$$\sum_{j=1}^{k} f_j(t) = \sum_{j=1}^{k} p_j(t) + \sum_{j=1}^{k-M} f_j(t). \quad (20)$$

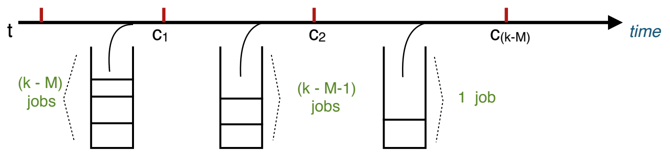

Fig. 3. The number of jobs waiting to be processed in different time periods where $k > M$.

By shifting terms in (20), we have: $\sum_{j=k-M+1}^{k} f_j(t) = \sum_{j=1}^{k} p_j(t)$. Summing up all job flowtime, it follows that:

$$\sum_{j=1}^{n(t)} f_j(t) = \sum_{j=1}^{q} f_j(t) + \sum_{k=1}^{z} \sum_{j=(k-1)M+q+1}^{kM+q} f_j(t)$$
$$\overset{(i)}{=} \sum_{j=1}^{q} p_j(t) + \sum_{k=1}^{z} \sum_{j=1}^{kM+q} p_j(t) \quad (21)$$
$$= \sum_{j=1}^{n(t)} (\lfloor \frac{n(t)-j}{M} \rfloor + 1) p_j(t),$$

where on the R.H.S. of (i), the first term is due to that the flowtime of the first q jobs is equal to their remaining job size and the second term is due to that $\sum_{j=(k-1)M+q+1}^{kM+q} f_j(t) = \sum_{j=1}^{kM+q} p_j(t)$. This completes the proof. □

Based on Lemma 5, if job j never arrive to the system and the subsequent jobs do not enter the system, the overall remaining job flowtime at time a_j is given by:

$$F_j^{'}(a_j) = \sum_{k=1}^{n(a_j)-1} (\lfloor \frac{n(a_j)-1-k}{M} \rfloor + 1) p_k(a_j). \quad (22)$$

In contrast, when job j arrives and the subsequent jobs do not arrive to the system at time a_j, the overall remaining job flowtime at time a_j is as follows:

$$F_j(a_j) = \sum_{k=1}^{\rho_j} (\lfloor \frac{n(a_j)-k}{M} \rfloor + 1) p_k(a_j)$$
$$+ \left(\lfloor \frac{n(a_j)-\rho_j-1}{M} \rfloor + 1\right) p_j \quad (23)$$
$$+ \sum_{k=\rho_j+1}^{n(a_j)} (\lfloor \frac{n(a_j)-k}{M} \rfloor + 1) p_k(a_j).$$

Therefore, one can view α_j as the incremental increase of the overall job flowtime caused by the arrival of job j by taking the difference of (22) and (23) and then dividing by $(1+\epsilon)p_j$. Since we are using a $(1+\epsilon)$-speed resource augmentation, thus, $\sum_j p_j\alpha_j$ exactly characterizes the overall job flowtime in SRPT, i.e., $\sum_j \alpha_j p_j = SRPT$.

Moreover, $\beta(t)$ reflects the loading condition of the cluster in our setting, thus, $M\int_0^\infty \beta(t) = \frac{1}{1+\epsilon} SRPT$. Therefore, we have $\sum_j \alpha_j p_j - M\int_0^\infty \beta(t)dt = \frac{\epsilon}{1+\epsilon} SRPT$.

Based on Lemma 3, we conclude that $\frac{\epsilon}{1+\epsilon} SR \leq \frac{\epsilon}{1+\epsilon} SRPT \leq P2 \leq P1 \leq (1+2\Delta)\cdot OPT$. This implies $SR \leq O(\frac{1}{\epsilon})OPT$ and completes the proof of Theorem 1. □

5 Algorithm Design for Multitasking Processors

In this section, we design scheduling algorithms for clusters supporting multitasking. Besides checkpointing times and level of redundancy, one must introduce additional variables, $\boldsymbol{x} = (\boldsymbol{x_j} : j = 1, 2, \cdots, N)$ where $\boldsymbol{x_j} = (x_j^k | k = 1, 2, \cdots, L_j)$ are the fractions of resource shares to be allocated to each job during checkpointing intervals. To be specific, we first design the Fair+R Algorithm which is an extension of the Fair Scheduler. Fair+R allows all jobs in the cluster to (near) equally share resources in the cluster, with priority given to those which arrive most recently. We then generalize Fair+R to design the LAPS+R(β) algorithm, which is an extension of LAPS (the Latest Arrival Processor Sharing). The main idea of LAPS is to share resources only among a certain fraction of jobs in the cluster [18]. However, the initial version of LAPS only considers the speed scaling among different jobs, our proposed LAPS+R(β) Algorithm extends this such that redundant copies of jobs can be made dynamically. In this section, we assume without loss of generality that jobs have been ordered such that $a_1 \le a_2 \le \cdots \le a_{n(t)}$.

5.1 Fair+R Algorithm and the performance guarantee

Let $n(t) = kM + l$ denote the number of jobs which are active in the cluster at time t.

At a high level, Fair+R works as follows. When $n(t) \ge M$, the kM jobs which arrive the most recently, i.e., jobs indexed from $(l+1)$ to $n(t)$, are each assigned to one server and gets a resource share of $\frac{1}{k}$. Each server processes k jobs simultaneously. By contrast, if $n(t) < M$, the latest arrival job, i.e., Job $n(t)$, is scheduled on $M - \lfloor \frac{M}{n(t)} \rfloor (n(t) - 1)$ machines and the others are each scheduled on $\lfloor \frac{M}{n(t)} \rfloor$ machines. In this case, there is no multitasking.

The corresponding pseudo-code is exhibited in the panel named Algorithm 2. Our main result for Fair+R is given in the following theorem:

Theorem 2. *Fair+R is (4+ϵ)-speed $O(\frac{1}{\epsilon})$-competitive with respect to the total job flowtime.*

Algorithm 2: Fair+R Algorithm

```
1  while A job arrives to or departure from the system do
2      Sort the jobs in the order such that
       $a_1 \le a_2 \le \cdots \le a_{n(t)}$ ;
3      Compute $n(t) = kM + l$;
4      if $n(t) \ge M$ then
5          for $j = l+1, l+2 \cdots, n(t)$ do
6              $r_j(t) = 1$ and $x_j(t) = 1/k$;
7      else
8          $r_{n(t)}(t) = M - \lfloor \frac{M}{n(t)} \rfloor (n(t) - 1)$ and
           $x_{n(t)}(t) = 1$;
9          for $j = 1, 2, \cdots, n(t) - 1$ do
10             $r_j(t) = \lfloor \frac{M}{n(t)} \rfloor$ and $x_j(t) = 1$;
11     Checkpoint all jobs and assign job $j$'s redundant
       executions to $r_j(t)$ machines which are uniformly
       chosen at random from $\{1, 2, \cdots, M\}$ with a
       resource share of $x_j(t)$;
```

5.2 Proof of Theorem 2

Paralleling the proof of Theorem 1, we adopt the dual-fitting approach to prove Theorem 2. Let $\boldsymbol{z}_j = (\boldsymbol{t_j}, \boldsymbol{x_j}, \boldsymbol{r_j}, L_j)$ and $\boldsymbol{z} = (\boldsymbol{z_j} | j = 1, 2, \cdots, N)$, we first formulate an approximate optimization problem as follows:

$$\min_{\boldsymbol{z}} \sum_{j=1}^{N} \int_{a_j}^{\infty} \frac{1}{p_j}(t - a_j + \frac{p_j}{4}) \widetilde{h}_j(\boldsymbol{t_j}, \boldsymbol{x_j}, \boldsymbol{r_j}, t) dt \tag{P3}$$
$$s.t. \int_{a_j}^{\infty} \widetilde{h}_j(\boldsymbol{t_j}, \boldsymbol{x_j}, \boldsymbol{r_j}, t) dt \ge p_j, \quad \forall j,$$
$$\sum_{j: a_j \le t} \sum_k x_j^k r_j^k \cdot \mathbb{1}(t \in (t_j^{k-1}, t_j^k]) \le M, \quad \forall t,$$
$$L_j \in \{1, 2, \cdots, 2N-1\}, \quad \boldsymbol{t_j} \in \mathcal{T}_j^{L_j+1}, \quad \boldsymbol{r_j} \in \mathbb{N}^{L_j}, \forall j.$$
$$0 < x_j^k \le 1, \quad \forall j, 1 \le k \le L_j.$$

Observe that P3 and P1 differ in both the objective and the second constraint since job j gets a resource share of $x_j^k r_j^k$ when $t \in (t_j^{k-1}, t_j^k]$.

The dual problem associated with P3 is similar to that of P1 and we only need to modify the first constraint of P2 to yield the following inequality:

$$\alpha_j - \beta(t) \le \frac{t - a_j}{p_j} + \frac{1}{4} \quad \forall j; t \ge a_j. \tag{24}$$

Paralleling Lemma 3, we have $P4 \le P3 \le (1 + \frac{1}{4}\Delta) \cdot OPT$ where $P4$ and $P3$ are the optimal values of the dual problem and P3 respectively.

Denote by $A(t)$ the set which contains all jobs that are still active in the cluster at time t under Fair+R. Thus, $n(t) = |A(t)|$. We shall set α_j as follows:

$$\alpha_j = \int_{a_j}^{c_j} \alpha_j(\tau) d\tau, \tag{25}$$

where

$$\alpha_j(t) = \frac{\sum_{k: a_k \le a_j} \mathbb{1}(k \in A(t)) \cdot \mathbb{1}(n(t) \ge M) \widetilde{h}_k(\boldsymbol{t}_k, \boldsymbol{x}_k, \boldsymbol{r}_k, t)}{(4+\epsilon) M p_j} + \frac{\mathbb{1}(n(t) < M) \widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t)}{4(4+\epsilon) p_j}. \tag{26}$$

and the setting of $\beta(t)$ is given by:

$$\beta(t) = \frac{1}{(4+\epsilon)M} n(t). \tag{27}$$

Next, we proceed to check the feasibility of these dual variables. Observe that α_j and $\beta(t)$ are nonnegative for all j, t and thus we only need to show they satisfy (24).

Lemma 6. *The dual variable settings in (25) and (27) satisfy the constraint in (24).*

Lemma 7. *Under the choice of dual variables in (25) and (27), $\sum_{j=1}^N \alpha_j p_j - M \int_0^\infty \beta(t) dt \ge \frac{\epsilon}{16+4\epsilon} FR$ where FR is the cost of Fair+R.*

Lemma 7 implies that $FR \le \frac{(16+4\epsilon)P4}{\epsilon} \le \frac{16+4\epsilon}{\epsilon} \cdot (1 + \frac{1}{4}\Delta) \cdot OPT = O(\frac{1}{\epsilon}) OPT$. This completes the proof of Theorem 2. □

5.3 LAPS+R(β) Algorithm and the performance guarantee

The algorithm depends on the parameter $\beta \in (0, 1)$. Say, $\beta = 1/2$, then the algorithm essentially schedules the $\frac{1}{2}n(t)$ most recently arrived jobs. If there are fewer than M such jobs, they are each assigned an roughly equal number of servers for execution without multitasking. If $\frac{1}{2}n(t) > M$, each job will roughly get a share of $\frac{M}{\frac{1}{2}n(t)}$ on some machine.

For a given number of active jobs $n(t)$, and parameter β, $z \in \mathbb{N}$, $\alpha \in \{0, 1, \cdots, M-1\}$ and $\gamma \in [0, 1)$ such that $\beta n(t) = zM + \alpha + \gamma$.

The LAPS+R(β) Algorithm operates as follows. At time t, if $z = 0$, jobs indexed from $(n(t) - \alpha)$ to $(n(t) - 1)$ are scheduled on $\lfloor \frac{M}{\alpha+1} \rfloor$ machines each, and Job $n(t)$ is scheduled on the remaining $(M - \alpha\lfloor \frac{M}{\alpha+1} \rfloor)$ machines. In this case, there is no multitasking. By contrast, if $z \geq 1$, jobs indexed from $(n(t) - zM - \alpha)$ to $(n(t) - 1)$ are each assigned a single machine and get a resource share of $\frac{1}{z+1}$. And, Job $n(t)$ is scheduled on $(M - \alpha)$ machines with a $\frac{1}{z+1}$ share of its resources.

The corresponding pseudo-code is exhibited as Algorithm 3 in the panel below.

Algorithm 3: LAPS+R(β) Algorithm

```
1  while A job arrives at or departure from the system do
2      Sort the jobs in the order such that $a_1 \geq a_2 \geq \cdots \geq a_{n(t)}$;
3      Compute $\beta n(t) = zM + \alpha + \gamma$ where $\gamma < 1$ and $\alpha < M$;
4      if $z \geq 1$ then
5          $r_{n(t)}(t) = (M - \alpha)$ and $x_{n(t)}(t) = \frac{1}{z+1}$;
6          for $j = n(t) - zM - \alpha, \cdots, n(t) - 1$ do
7              $r_j(t) = 1$ and $x_j(t) = \frac{1}{z+1}$;
8      if $z < 1$ then
9          $r_{n(t)}(t) = M - \alpha\lfloor \frac{M}{\alpha+1} \rfloor$ and $x_{n(t)}(t) = 1$;
10         for $j = n(t) - \alpha, \cdots, n(t) - 1$ do
11             $r_j(t) = \lfloor \frac{M}{\alpha+1} \rfloor$ and $x_j(t) = 1$;
12     for $j = 1, 2, \cdots, n(t) - zM - \alpha - 1$ do
13         $x_j(t) = r_j(t) = 0$;
14     Checkpoint all jobs and assign job $j$'s redundant executions to $r_j(t)$ machines which are uniformly chosen at random from $\{1, 2, \cdots, M\}$ with a resource share of $x_j(t)$;
```

5.3.1 *Performance guarantee for LAPS+R(β) and our techniques*

Let OPT and LR denote the cost of the optimal scheduling policy and LAPS+R(β) respectively. The main result in this section, characterizing the competitive performance of LAPS+R(β), is given in the following theorem:

Theorem 3. *LAPS+R(β) is $(2 + 2\beta + 2\epsilon)$-speed $O(\frac{1}{\beta\epsilon})$-competitive with respect to the total job flowtime.*

The dual fitting approach fails in this setting so we adopt the use of a potential function, which is widely used to derive performance bounds with resource augmentation for online parallel scheduling algorithms e.g., [18], [29]. The main idea of this method is to find a potential function which combines the optimal schedule and LAPS+R(β). To be specific, let $LR(t)$ and $OPT(t)$ denote the accumulated job flowtime under LAPS+R(β) with a $(2 + 2\beta + 2\epsilon)$-speed resource augmentation and the optimal schedule, respectively. We define a potential function, $\Lambda(t)$, which satisfies the following properties:

1) Boundary Condition: $\Lambda(0) = \Lambda(\infty) = 0$.
2) Jumps Condition: the potential function may have jumps only when a job arrives or completes under the LAPS+R(β) schedule, and if present, it must be decreased.
3) Drift Condition: with a $(2 + 2\beta + 2\epsilon)$-speed resource augmentation, for any time t not corresponding to a jump, and some constant c, we have that,

$$\frac{d\Lambda(t)}{dt} \leq -\epsilon\beta \cdot \frac{dLR(t)}{dt} + c \cdot \frac{dOPT(t)}{dt}. \tag{28}$$

By integrating 28 and accounting for the negative jump and the boundary condition, one can see that the existence of such a potential function guarantees that $LR \leq \frac{c}{\beta\epsilon}OPT$ under a $(2 + 2\beta + 2\epsilon)$-speed resource augmentation.

5.4 Proof of Theorem 3

To prove Theorem 3, we shall propose a potential function, $\Lambda(t)$, which satisfies all the three properties specified above.

5.4.1 *Defining potential function, $\Lambda(t)$*

Consider a checkpointing trajectory for job j under LAPS+R(β) and the optimal schedule, denoted by $(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j)$ and $(\boldsymbol{t}_j^*, \boldsymbol{x}_j^*, \boldsymbol{r}_j^*)$ respectively. Let $\psi^*(t)$ be the jobs that are still active at time t under the optimal scheduling and denote by $\psi(t)$ the set of jobs that are active under LAPS+R(β). Thus, we have that $|\psi(t)| = n(t)$. Further let $n_j(t)$ denote the number of jobs which are active at time t and arrive no later than job j under LAPS+R(β). Define the cumulative service difference between the two schedules for job j at time t, i.e., $\pi_j(t)$, as follows:

$$\pi_j(t) = \max\left[\int_{a_j}^{t} \widetilde{h}_j(\boldsymbol{t}_j^*, \boldsymbol{x}_j^*, \boldsymbol{r}_j^*, \tau)d\tau - \int_{a_j}^{t} \widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, \tau)d\tau\ ,\ 0\right], \tag{29}$$

Let $\delta = 2 + 2\beta + 2\epsilon$ and define

$$f(n_j(t)) = \begin{cases} 1 & \beta n_j(t) \leq M, \\ \frac{M}{\beta n_j(t)} & \text{otherwise.} \end{cases} \tag{30}$$

Note that $f(n_j(t))$ takes the minimum of 1 and $\frac{M}{\beta n_j(t)}$ where the latter is roughly the total resource allocated to job j under LAPS+R(β) if $n_j(t)$ jobs were active at time t.

Our potential function is given by:

$$\Lambda(t) = \sum_{j \in \psi(t)} \Lambda_j(t), \tag{31}$$

where $\Lambda_j(t)$ is the ratio between (29) and (30), i.e.,

$$\Lambda_j(t) = \frac{\pi_j(t)}{\delta \cdot f(n_j(t))}.$$

5.4.2 *Changes in $\Lambda(t)$ caused by job arrival and departure*

Clearly, our potential function satisfies the boundary condition. Indeed, since each job is completed under LAPS+R(β), thus, $\psi(t)$ will eventually be empty, and $\Lambda(0) = \Lambda(\infty) = 0$.

Let us consider possible jump times. When job j arrives to the system at time a_j, $\pi_j(a_j) = 0$ and $f(n_j(t))$ does not change for all $k \neq j$. Therefore, we conclude that the job arrival does not change the potential function $\Lambda(t)$. When a job leaves the system under LAPS+R(β), $f(n_j(t))$ can only increase if job j is active at time t, leading to a decrease in $\Lambda_j(t)$. As a consequence, the job arrival or departure does not cause any increase in the potential function, $\Lambda(t)$, thus, the jump condition on the potential function is satisfied.

5.4.3 *Changes of $\Lambda(t)$ caused by job processing*

Beside job arrivals and departures under LAPS+R(β), there are no other events leading to changes in $f(n_j(t))$ and thus changes in $\Lambda_j(t)$ depend only on $\pi_j(t)$, see definition of $\Lambda_j(t)$ in (31). Specifically, for all $t \notin \{a_j\}_j \cup \{c_j\}_j$, we have that,

$$\frac{d\Lambda(t)}{dt} = \sum_{j\in\psi(t)} \frac{d\Lambda_j(t)}{dt} = \sum_{j\in\psi(t)} \frac{d\pi_j(t)/dt}{\delta \cdot f(n_j(t))},$$

where we let $\frac{d\pi_j(t)}{dt} = \lim_{\tau\to 0^+} \frac{\pi_j(t+\tau)-\pi_j(t)}{\tau}$ and thus $\frac{d\pi_j(t)}{dt}$ exists for all $t \geq 0$. Moreover, we have:

$$\begin{aligned}\frac{d\pi_j(t)}{dt} \leq\ & \mathbb{1}(j \in \psi^*(t))\widetilde{h}_j(\boldsymbol{t}_j^*, \boldsymbol{x}_j^*, \boldsymbol{r}_j^*, t)\\ & - \mathbb{1}(j \notin \psi^*(t))\widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t),\end{aligned} \tag{32}$$

indeed, either $j \in \psi^*(t)$ so job j has not completed under the optimal policy and the drift is bounded by the first term in (32), or $j \notin \psi^*(t)$ and the job has completed under the optimal policy, the difference term in (29) is positive and its derivative is given by the the second term in (32). Therefore, for all $t \notin \{a_j\}_j\cup\{c_j\}_j$, we have the following upper bound:

$$\begin{aligned}\frac{d\Lambda(t)}{dt} \leq & \sum_{j\in\psi(t)} \frac{\mathbb{1}(j \in \psi^*(t))\widetilde{h}_j(\boldsymbol{t}_j^*, \boldsymbol{x}_j^*, \boldsymbol{r}_j^*, t)}{\delta \cdot f(n_j(t))}\\ & - \sum_{j\in\psi(t)} \frac{\mathbb{1}(j \notin \psi^*(t))\widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t)}{\delta \cdot f(n_j(t))}\\ \leq & \underbrace{\sum_{j\in\psi^*(t)} \frac{\widetilde{h}_j(\boldsymbol{t}_j^*, \boldsymbol{x}_j^*, \boldsymbol{r}_j^*, t)}{\delta \cdot f(n_j(t))}}_{\Gamma^*(t)} - \underbrace{\sum_{j\in\psi(t)\setminus\psi^*(t)} \frac{\widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t)}{\delta \cdot f(n_j(t))}}_{\Gamma(t)}\end{aligned} \tag{33}$$

where $\psi(t) \setminus \psi^*(t)$ contains all the jobs that are in $\psi(t)$ but not in $\psi^*(t)$. For ease of illustration, let $\Gamma^*(t)$ and $\Gamma(t)$ denote the two terms on the R.H.S. of (33). In the sequel, we bound these two terms.

5.4.4 *An upper bound of $\Gamma^*(t)$*

When $\beta n_j(t) \geq M$, we have $f(n_j(t)) = M/\beta n_j(t)$, thus, $\frac{\widetilde{h}_j(\boldsymbol{t}_j^*,\boldsymbol{x}_j^*,\boldsymbol{r}_j^*,t)}{f(n_j(t))} = \frac{\widetilde{h}_j(\boldsymbol{t}_j^*,\boldsymbol{x}_j^*,\boldsymbol{r}_j^*,t)}{M/\beta n_j(t)}$. By contrast, when $\beta n_j(t) \leq M$, it follows that $f(n_j(t)) = 1$, so $\frac{\widetilde{h}_j(\boldsymbol{t}_j^*,\boldsymbol{x}_j^*,\boldsymbol{r}_j^*,t)}{f(n_j(t))} = \widetilde{h}_j(\boldsymbol{t}_j^*, \boldsymbol{x}_j^*, \boldsymbol{r}_j^*, t)$, which is upper bounded by Δ based on Lemma 2.

Therefore, we have:

$$\frac{\widetilde{h}_j(\boldsymbol{t}_j^*, \boldsymbol{x}_j^*, \boldsymbol{r}_j^*, t)}{\delta f(n_j(t))} \leq \frac{1}{\delta}\Big(\frac{\widetilde{h}_j(\boldsymbol{t}_j^*, \boldsymbol{x}_j^*, \boldsymbol{r}_j^*, t)}{M/\beta n_j(t)} + \Delta\Big),$$

and

$$\begin{aligned}\Gamma^*(t) &\leq \sum_{j\in\psi^*(t)} \frac{1}{\delta}\Big(\frac{\widetilde{h}_j(\boldsymbol{t}_j^*, \boldsymbol{x}_j^*, \boldsymbol{r}_j^*, t)}{M/\beta n_j(t)} + \Delta\Big)\\ &\leq \frac{\Delta|\psi^*(t)|}{\delta} + \sum_{j\in\psi^*(t)} \beta n(t)\frac{\widetilde{h}_j(\boldsymbol{t}_j^*, \boldsymbol{x}_j^*, \boldsymbol{r}_j^*, t)}{\delta M}\\ &\leq \Delta|\psi^*(t)|/\delta + \beta n(t)/\delta,\end{aligned} \tag{34}$$

where the last inequality is due to

$$\sum_{j\in\psi^*(t)} \widetilde{h}_j(\boldsymbol{t}_j^*, \boldsymbol{x}_j^*, \boldsymbol{r}_j^*, t) \leq \sum_{j\in\psi^*(t)}\sum_k x_j^k r_j^k \cdot \mathbb{1}(t \in (t_j^{k-1}, t_j^k]) \leq M,$$

for all t.

5.4.5 *An upper bound of $\Gamma(t)$*

First, $\Gamma(t)$ ban be represented as:

$$\Gamma(t) = \sum_{j\in\psi^*(t)\cap\psi(t)} \frac{\widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t)}{\delta f(n_j(t))} - \sum_{j\in\psi(t)} \frac{\widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t)}{\delta f(n_j(t))}. \tag{35}$$

To get an upper bound of $\Gamma(t)$, we will consider two cases. In particular, $\beta n(t) = zM + \alpha + \gamma$, we consider the case where $z = 0$ and that where $z \geq 1$.

Case 1: Suppose $z = 0$, in this case, $\lceil \beta n(t) \rceil \leq M$. Since $n_j(t) \leq n(t)$ for all $1 \leq j \leq n(t)$, it then follows that $\beta n_j(t) \leq M$ and $f(n_j(t)) = 1$, which implies, for all $j \in \psi^*(t) \cap \psi(t)$, $\frac{\widetilde{h}_j(\boldsymbol{t}_j,\boldsymbol{x}_j,\boldsymbol{r}_j,t)}{\delta f(n_j(t))} \leq \Delta$ since $\widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t) \leq \delta\Delta$ as we are using a δ-speed resource augmentation. Thus, the first term on the R.H.S. of (35) is upper bounded by:

$$\sum_{j\in\psi^*(t)\cap\psi(t)} \frac{\widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t)}{\delta f(n_j(t))} \leq \sum_{j\in\psi^*(t)\cap\psi(t)} \Delta \leq |\psi^*(t)|\Delta. \tag{36}$$

Consider $j \in \psi(t)$ where $n(t) - \alpha \leq j \leq n(t)$ and $t \in [t_j^{k-1}, t_j^k)$ for some $k \in \{1, 2, \cdots, L_j\}$. Then, the number of redundant executions for job j, $r_j^k \geq \lfloor \frac{M}{\alpha+1} \rfloor \geq 1$. Thus, $\widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t) \geq \delta$ and $\frac{\widetilde{h}_j(\boldsymbol{t}_j,\boldsymbol{x}_j,\boldsymbol{r}_j,t)}{\delta f(n_j(t))} \geq 1$. Combining (35) 'and (36), we then have:

$$\Gamma(t) \leq \Delta|\psi^*(t)| - (\alpha + 1) \leq \Delta|\psi^*(t)| - \beta n(t), \tag{37}$$

Case 2: Suppose $z \geq 1$, then, $\lceil \beta n(t) \rceil > M$ and $\frac{M}{\beta n(t)} \geq \frac{1}{z+1}$. Similarly, we consider job $j \in \psi(t)$ where $n(t) - kM - \alpha \leq j \leq n(t)$ and $t \in (t_j^{k-1}, t_j^k]$. Based on the scheduling policy of LAPS+R(β), we have that, $x_j^k = \frac{1}{z+1}$ and $r_j^k \geq 1$. Therefore, $\widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t)$ is bounded by:

$$\frac{\delta}{z+1} \leq \widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t) \leq \frac{\delta \cdot \Delta}{z+1}. \tag{38}$$

Moreover, we have: $\min[1\ ,\ \frac{M}{\beta n_j(t)}] \geq \min[1\ ,\ \frac{M}{\beta n(t)}] \geq \frac{1}{z+1}$. Therefore, it follows that:

$$\frac{1}{z+1} \leq \min\big[1\ ,\ \frac{M}{\beta n_j(t)}\big] \leq f(n_j(t)) \leq \frac{M}{\beta n_j(t)}. \tag{39}$$

Combining (38) and (39), we have that, for all $n(t) - kM - \alpha \leq j \leq n(t) - 1$,

$$\frac{1/(z+1)}{M/\beta n_j(t)} \leq \frac{\widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t)}{\delta f(n_j(t))} \leq \Delta. \tag{40}$$

Substituting (40) into (35), it then follows that,

$$\begin{aligned}\Gamma(t) &\le \sum_{j\in\psi^*(t)\cap\psi(t)} \Delta - \sum_{j=n(t)-zM-\alpha}^{n(t)} \frac{1/(z+1)}{M/\beta n_j(t)} \\ &\le \Delta|\psi^*(t)| - \frac{\beta z M(n(t)-\frac{zM}{2}-\frac{\alpha}{2})}{M(z+1)} \\ &\le \Delta|\psi^*(t)| - \beta(\frac{1}{2}-\frac{\beta}{4})n(t),\end{aligned} \tag{41}$$

where the second inequality is due to that $n_j(t) = j$ and the last inequality is because $zM + \alpha \le \beta n(t)$ and $\frac{z}{z+1} \ge 1/2$. Based on **Case 1** and **Case 2**, we have: $\Gamma(t) \le \Delta|\psi^*(t)| - \beta(\frac{1}{2}-\frac{\beta}{4})n(t)$. Thus, combining (33) and (34), we have the following upper bound for the drift, $\frac{d\Lambda(t)}{dt}$:

$$\begin{aligned}\frac{d\Lambda(t)}{dt} &\le \Gamma^*(t) + \Gamma(t) \\ &\le \Delta|\psi^*(t)|/\delta + \beta n(t)/\delta + \Delta|\psi^*(t)| - \beta(\frac{1}{2}-\frac{\beta}{4})n(t). \\ &= \frac{(\delta+1)\Delta}{\delta}|\psi^*(t)| + \frac{\beta(1-\delta(\frac{1}{2}-\frac{\beta}{4}))}{\delta}n(t) \\ &\le \frac{(\delta+1)\Delta}{\delta}|\psi^*(t)| - \frac{\epsilon\beta}{2\delta}n(t),\end{aligned} \tag{42}$$

where the last inequality is due to $\delta = 2 + 2\beta + 2\epsilon$ and $\delta(\frac{1}{2}-\frac{\beta}{4}) \ge 1 + \frac{\epsilon}{2}$.

Based on (42), we then have that,

$$\begin{aligned}0 = \Lambda(\infty) - \Lambda(0) &\le \int_0^\infty \frac{d\Lambda(t)}{dt}dt \\ &\le \frac{(\delta+1)\Delta}{\delta}\int_0^\infty |\psi^*(t)|dt - \frac{\epsilon\beta}{2\delta}\int_0^\infty n(t)dt \\ &= \frac{(\delta+1)\Delta}{\delta}OPT - \frac{\epsilon\beta}{2\delta}LR,\end{aligned} \tag{43}$$

where the first inequality is due to that there exist negative jumps during the evolution of $\Lambda(t)$. This completes the proof of Theorem 3. □

6 Numerical Studies

In this section, we conduct several numerical studies to evaluate our proposed algorithms in both the multi-tasking and non-multi-tasking setting. As pointed out in [33], the Gamma distribution is a good fit for the failure model of most parallel and distributed computing systems. Therefore, we apply the Gamma distribution to generate machine service rates in a cluster with 100 machines over a period which lasts 100000 units of time.

To be more specific, we categorize the service process of each machine into two classes, namely, the Available Period (AP) and Unavailable Period (UP). As depicted in Fig. 4, each UP follows an AP. During an available period, the rate of the machine service capacity is uniformly distributed in $[2, 3]$. On the other hand, when the machine is processing jobs in an unavailable period, its rate is uniformly distributed in $[0, 0.3]$. In addition, we apply the statistics of the trace data collected from a computational grid platform (see [33]) to generate a series of available and unavailable periods for each machine independently. The length of an AP is Gamma distributed with $k = 0.34$ and $\theta = 94.35$

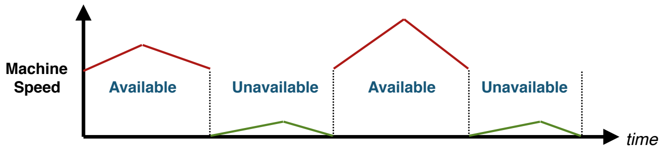

Fig. 4. Fluctuation of machine service rates in different time periods.

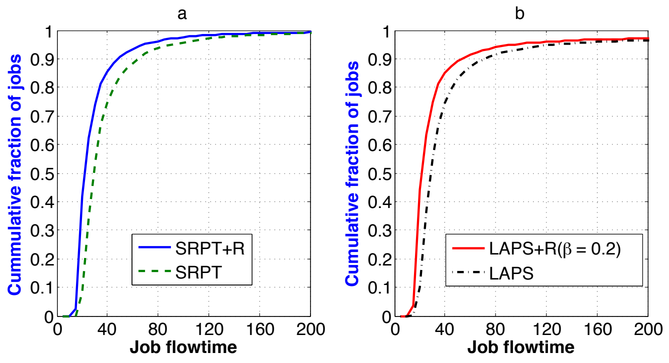

Fig. 5. The comparison between algorithms with and without redundancy. Panel a shows the CDF of the job flowtimes under both SPRT+R and SRPT. Panel b shows the CDF of the job flowtimes under LAPS+R(β) with $\beta = 0.2$ and LAPS.

where k and θ are the shape parameter and scale parameter respectively. In contrast, the length of an UP is Gamma distributed with $k = 0.19$ and $\theta = 39.92$. We also normalize all the distributions such that the mean service rate is one.

In all the following evaluations, we consider time is slotted and the scheduling decisions are made at the beginning of each time slot. The jobs arrive at the cluster following a Poisson Process with rate λ and the workload of each job is Pareto distributed as shown below.

$$\mathrm{P}\{p_j \le x\} = \begin{cases} 1-(\frac{b}{x})^\alpha & x \ge b, \\ 0 & \text{otherwise,}\end{cases}$$

where $b = 20$ and $\alpha = 2$. It can be readily shown that the mean of the job workload is 40.

In the following simulations, we will compare the average as well as the cumulative distribution function (i.e., CDF) of job flowtimes for different algorithms.

6.1 Benefit of Redundancy

In this subsection, we implement scheduling algorithms with both redundancy and no redundancy to characterize the benefit of redundant execution. We set the job arrival rate λ to one and depict the simulation results in Fig. 5 and Fig. 6. As shown in Fig. 5, more than 85% of jobs can complete within 40 units of time under SRPT+R. As a comparison, only 75% of jobs complete within 40 units of time under the SRPT scheme. It's worthy noting that this result also applies to LAPS+R(β) and LAPS. Moreover, Fig. 6 shows that, with redundancy, the average job flowtime can be reduced by nearly 25% under all the scheduling algorithms.

6.2 Comparison of various algorithms

We conducted a more comprehensive comparison of various algorithms by tuning values of λ. Following the simulation parameters configured at the beginning of Section 6, we can readily show that, $\lambda = 2.5$ reaches the heavy traffic limit

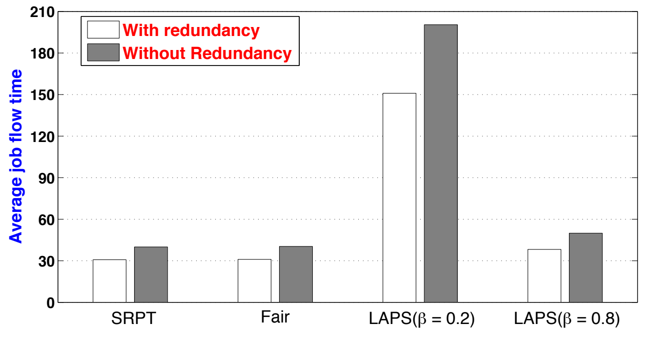

Fig. 6. Average job flowtime under different scheduling algorithms with and without redundancy.

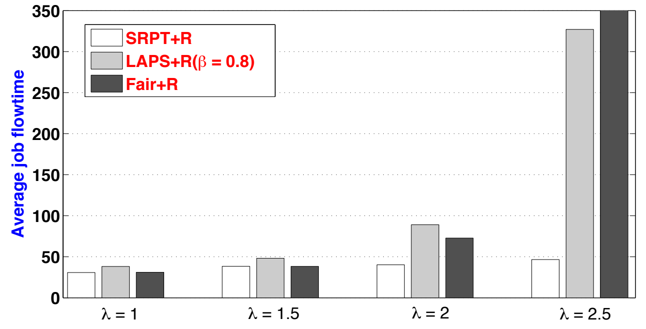

Fig. 7. Comparison between different algorithms in terms of average job flowtime for different λ.

above which the system is overloaded. As such, we tune λ from 1 to 2.5 in this simulation. Observe in Fig. 7 that the average job flowtimes under SRPT+R and Fair+R are roughly the same under $\lambda = 1$ and $\lambda = 1.5$. However, when λ increases, SRPT+R tends to perform much better than both LAPS+R(β) and Fair+R. For $\lambda = 2$, the average job flowtime under both LAPS+R($\beta = .8$) and Fair+R is two times that under SRPT+R. More importantly, when λ hits the heavy traffic limit, the average job flowtime under both LAPS+R(β) and Fair+R increases significantly in λ while it doesn't change much under SRPT+R. In addition, Fair+R outperforms LAPS+R(β) when λ is below 2. Conversely, if λ is above 2, LAPS+R(β) performs better than Fair+R.

6.3 The impact of β in LAPS+(β)

Since β has a high impact on the performance of LAPS+(β), in this subsection, we tune the values of β to illustrate the performance of LAPS+(β) under different settings.

We depict the comparison results under the heavy traffic regime where $\lambda = 2.5$ in Fig. 8. It shows that, when β decreases, the number of jobs with small flowtime (less than 200 units of time) increases. Therefore, small jobs benefit more than large jobs under a small β as they have higher priorities to be allocated resources in the cluster. In addition, when $\beta = .6$, the average job flowtime attains its minimum.

As illustrated in Fig. 9, when $\lambda = 1$, almost all of the jobs in the cluster can complete within 200 units of time under different settings for β. When the job arrival rate is low, the jobs with small workloads can get large fractions of shared resource under a small value of β. In this case, the benefit of redundancy is marginal and tuning down the value of β does not help much for small jobs. However, in terms of the average job flowtime, a smaller β leads to a worse performance. The reason behind is that a large job

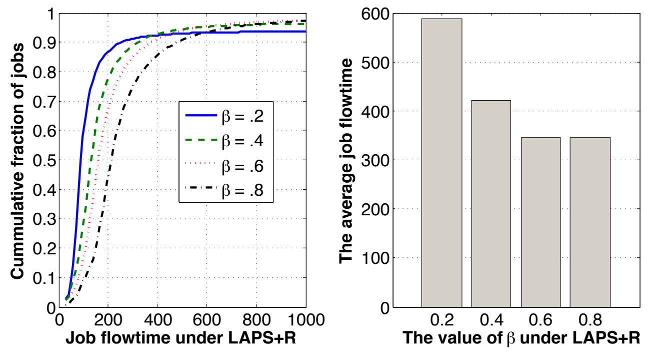

Fig. 8. The job flowtime under different β in LAPS(β) when $\lambda = 2.5$.

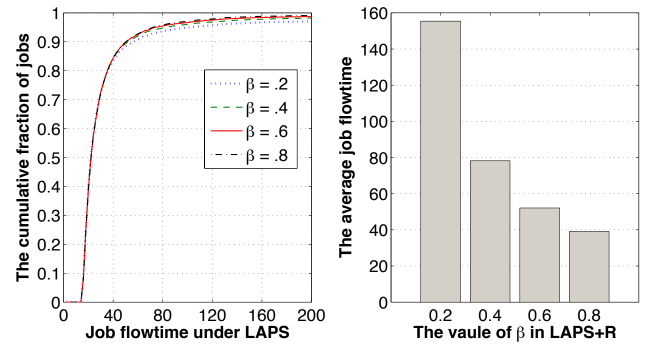

Fig. 9. The job flowtime under different β in LAPS+R(β) when $\lambda = 1$.

usually has a very small chance to obtain shared resource under a small value of β in LAPS+R(β) since it takes a long time to complete, resulting in a large flowtime. Though the number of large jobs is small, the amount of total job flowtime contributed but those large ones is significant.

7 Conclusions and Future Directions

This paper is an attempt to address the impact of two key sources of variability in parallel computing clusters: job processing times and machine processing rate. Our primary aim and contribution was to introduce a new speedup function account for redundancy, and provide the fundamental understanding on how job scheduling and redundant execution algorithms with limited number of checkpointings can help to mitigate the impact of variability on job response time. As the need of delivering predictable service on shared cluster and computing platforms grows, approaches, such as ours, will likely be an essential element of any possible solution. Extensions of this work to non-clairvoyant scenarios, the case of jobs with associated task graphs etc, are likely next steps towards developing the foundational theory and associated algorithms to address this problem.

References

[1] Apache. http://hadoop.apache.org, 2013.
[2] Capacity Scheduler. http://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html, 2013.
[3] Fair Scheduler. http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html, 2013.
[4] S. Aalto, U. Ayesta, and R. Righter. On the gittins index in the M/G/1 queue. *Queuing Systems*, 63(1):437–458, December 2009.
[5] S. Aalto, A. Penttinen, P. Lassila, and P. Osti. On the optimal trade-off between SRPT and opportunistic scheduling. In *Proceedings of Sigmetrics*, June 2011.

[6] S. Anand, N. Garg, and A. Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *Proceedings of SODA*, 2002.

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

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

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

[10] G. Bronevetsky, D. Marques, and K. Pingali. Application-level checkpointing for shared memory programs. In *ASPLOS*, 2004.

[11] H. L. Chan, J. Edmonds, and K. Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In *SPAA*, pages 1–10, 2009.

[12] 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*, pages 3074–3082, March 2011.

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

[14] Q. Chen, C. Liu, and Z. Xiao. Improving MapReduce performance using smart speculative execution strategy. *IEEE Transactions on Computers*, 63(4), April 2014.

[15] S. Chen, Y. Sun, U. C. Kozat, L. Huang, P. Sinha, G. Liang, X. Liu, and N. B. Shroff. When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds. In *Infocom*, April 2014.

[16] S. Das, V. Narasayya, F. Li, and M. Syamala. CPU sharing techniques for performance isolation in multi-tenant. *Proceedings of the VLDB Endowment*, 7(1), September 2013.

[17] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of OSDI*, pages 137–150, December 2004.

[18] J. Edmonds and K. Pruhs. Scalably scheduling processes with arbitrary speedup curves. *ACM Transaction on Algorithms*, 8(28), 2012.

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

[20] K. Gardner, M. Harchol-Balter, and A. Scheller-Wolf. A better model for job redundancy: Decoupling server slowdown and job size. In *IEEE MASCOTS*, pages 1–10. IEEE, 2016.

[21] K. Gardner, M. Harchol-Balter, A. Scheller-Wolf, M. Velednitsky, and S. Zbarsky. Redundancy-d: The power of d choices for redundancy. In *Operation Research*, to appear, 2017.

[22] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. ACM SIGCOMM, August 2014.

[23] A. Gupta, R. Krishnaswamy, and K. Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In *Proceedings of WAOA*, pages 173–186, 2002.

[24] A. Gupta, B. Moseley, S. Im, and K. Pruhs. Scheduling jobs with varying parallelizability to reduce variance. In *SPAA: 22nd ACM Symposium on Parallelism in Algorithms and Architectures*, 2010.

[25] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello. Modeling and tolerating heterogeneous failures in large parallel system. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[26] S. Im, J. Kulkarni, and B. Moseley. Temporal fairness of round robin: Competitive analysis for lk-norms of flow time. In *SPAA*, pages 155–160, 2015.

[27] S. Im, J. Kulkarni, and K. Munagala. Competitive algorithms from competitive equilibria: non-clairvoyant scheduling under polyhedral constraints. In *Proceedings of STOC*, pages 313–322, 2014.

[28] S. Im, J. Kulkarni, K. Munagala, and K. Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *Proceedings of FOCS*, pages 531–540, 2014.

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

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

[31] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceeding of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, March 2007.

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

[33] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In *CCGrid*, pages 398–407, 2010.

[34] S. Leonardia and D. Raz. Approximating total flow time on parallel machines. *Journal of Computer and System Sciences*, 73(6), September 2007.

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

[36] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlos. On scheduling in map-reduce and flow-shops. In *Proceedings of SPAA*, pages 289–298, June 2011.

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

[38] Z. Qiu and J. F. Pérez. Assessing the impact of concurrent replication with canceling in parallel jobs. In *MASCOTS*, 2014.

[39] Z. Qiu and J. F. Pérez. Enhancing reliability and response times via replication in computing clusters. In *Infocom*, April 2015.

[40] T. W. Reiland. Optimality conditions and duality in continuous programming I. convex programs and a theorem of the alternative. *Journal of Mathematical Analysis and Applications*, 77(1):297 – 325, 1980.

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

[42] N. Shah, K. Lee, and K. Ramchandran. When do redundant requests reduce latency? In *Annual Allerton Conference on Communication, Control, and Computing*, Oct 2013.

[43] 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.

[44] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and C. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *RTSS*, pages 288 – 299, 1996.

[45] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Low latency via redundancy. In *CoNEXT*, 2013.

[46] A. Wierman, L. L. H. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems: Optimality and robustness. *Performance Evaluation*, pages 601–622, December 2012.

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

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

[49] H. Xu, W. C. Lau, Z. Yang, G. de Veciana, and H. Hou. Mitigating service variability in mapreduce clusters via task cloning: A competitive analysis. In *IEEE Transactions on Parallel and Distributed Systems*, http://ieeexplore.ieee.org/document/7890998, 2017.

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

[51] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: fault-tolerant streaming computation at scale. In *SOSP*, pages 423–438, 2013.

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

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

APPENDIX A
PROOF OF LEMMA 3

Proof. Consider an optimal solution to OPT, $\boldsymbol{y}^*$, whose corresponding job completion time for job j is denoted by

c_j^*. Thus, for all $j = 1, 2, \cdots, N$, $\boldsymbol{y}^*$ and c_j^* satisfy:

$$\int_{a_j}^{c_j^*} h_j(\boldsymbol{t_j^*}, \boldsymbol{r_j^*}, t)dt = p_j. \tag{44}$$

Moreover, it follows that $h_j(\boldsymbol{t_j^*}, \boldsymbol{r_j^*}, t) = 0$ for all $t \geq c_j^*$, thus, we have that:

$$\int_{a_j}^{\infty} h_j(\boldsymbol{t_j^*}, \boldsymbol{r_j^*}, t)dt = p_j, \tag{45}$$

and it follows that:

$$\begin{aligned}&\int_{a_j}^{\infty} \frac{1}{p_j}(t-a_j)h_j(\boldsymbol{t_j^*}, \boldsymbol{r_j^*}, t)dt = \int_{a_j}^{c_j^*} \frac{1}{p_j}(t-a_j)h_j(\boldsymbol{t_j^*}, \boldsymbol{r_j^*}, t)dt\\ &\leq \int_{a_j}^{c_j^*} \frac{1}{p_j}(c_j^*-a_j)h_j(\boldsymbol{t_j^*}, \boldsymbol{r_j^*}, t)dt = c_j^* - a_j.\end{aligned} \tag{46}$$

Following Lemma 2, it can be readily shown that $h_j(\boldsymbol{t_j^*}, \boldsymbol{r_j^*}, t) \leq \Delta$. Therefore, we have:

$$p_j = \int_{a_j}^{c_i^*} h_j(\boldsymbol{t_j^*}, \boldsymbol{r_j^*}, t)dt \leq \Delta(c_j^* - a_j). \tag{47}$$

Combining (46) and (47), we have:

$$\int_{a_j}^{\infty} \frac{(t-a_j+2p_j)}{p_j} \cdot h_j(\boldsymbol{t_j^*}, \boldsymbol{r_j^*}, t)dt \leq (1+2\Delta)(c_j^* - a_j).$$

Since the optimal solution to OPT must be feasible for P1, it follows that:

$$\begin{aligned}P1 &\leq \sum_{j=1}^{N} \int_{a_j}^{\infty} \frac{(t-a_j+2p_j)}{p_j} \cdot h_j(\boldsymbol{t_j^*}, \boldsymbol{r_j^*}, t)dt\\ &\leq (1+2\Delta)\sum_{j=1}^{N}(c_j^* - a_j) = (1+2\Delta)OPT.\end{aligned} \tag{48}$$

This completes the proof. □

Appendix B
Proof of Lemma 6

Proof. First, we have:

$$\begin{aligned}&\frac{1}{4(4+\epsilon)p_j}\int_{a_j}^{c_j} \mathbb{1}(n(t) < M) \cdot \widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t)dt\\ &\leq \frac{1}{4(4+\epsilon)p_j}\int_{a_j}^{c_j} \widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t)dt = \frac{1}{4}.\end{aligned} \tag{49}$$

Next, we proceed to show the following result holds.

$$\begin{aligned}&\frac{\int_{a_j}^{c_j} \sum_{k:a_k \leq a_j} \mathbb{1}(j \in A(\tau)) \cdot \mathbb{1}(n(\tau) \geq M)\widetilde{h}_k(\boldsymbol{t}_k, \boldsymbol{x}_k, \boldsymbol{r}_k, \tau)d\tau}{(4+\epsilon)Mp_j}\\ &\leq \frac{t-a_j}{p_j} + \frac{1}{(4+\epsilon)M}n(t).\end{aligned} \tag{50}$$

To achieve this, we divide the job set $\Psi_j = \{k : a_k \leq a_j\}$ into two separate sets: $\Psi_j^1 = \{k : c_k \leq t\} \cap \Psi_j$ and $\Psi_j^2 = \{k : c_k > t\} \cap \Psi_j$. For the first set, we have:

$$\begin{aligned}&\frac{\int_{a_j}^{c_j} \sum_{k:k\in\Psi_j^1} \mathbb{1}(k \in A(\tau)) \cdot \mathbb{1}(n(\tau) \geq M) \cdot \widetilde{h}_k(\boldsymbol{t}_k, \boldsymbol{x}_k, \boldsymbol{r}_k, \tau)d\tau}{M}\\ &\leq \frac{\int_{a_j}^{t} \sum_{k:k\in\Psi_j^1} \mathbb{1}(k \in A(\tau)) \cdot \mathbb{1}(n(\tau) \geq M) \cdot \widetilde{h}_k(\boldsymbol{t}_k, \boldsymbol{x}_k, \boldsymbol{r}_k, \tau)d\tau}{M}.\end{aligned} \tag{51}$$

Based on the scheduling principle of Fair+R, it follows that:

$$\sum_k \mathbb{1}_{k\in A(t)} \cdot \mathbb{1}_{n(t)\geq M} \cdot \widetilde{h}_k(\boldsymbol{t}_k, \boldsymbol{x}_k, \boldsymbol{r}_k, t) \leq (4+\epsilon)M. \tag{52}$$

Therefore, the L.H.S of (51) is upper bounded by $(4+\epsilon)(t-a_j)$. For all jobs in Ψ_j^2, we have:

$$\begin{aligned}&\int_{a_j}^{c_j} \sum_{k:k\in\Psi_j^2} \mathbb{1}(k \in A(\tau)) \cdot \mathbb{1}(n(\tau) \geq M) \cdot \widetilde{h}_k(\boldsymbol{t}_k, \boldsymbol{x}_k, \boldsymbol{r}_k, \tau)d\tau\\ &= \sum_{k:k\in\Psi_j^2} \int_{a_j}^{c_k} \mathbb{1}(k \in A(\tau)) \cdot \mathbb{1}(n(\tau) \geq M) \cdot \widetilde{h}_k(\boldsymbol{t}_k, \boldsymbol{x}_k, \boldsymbol{r}_k, \tau)d\tau\\ &\overset{(ii)}{\leq} \sum_{k:a_k\leq t\leq c_k\leq c_j} p_j \leq n(t)p_j,\end{aligned} \tag{53}$$

where (ii) is due to that, for any job who arrives before j, its amount of work processed between the range $[a_j, c_k]$ is upper bounded by p_j.

Combining all inequalities above, the lemma immediately follows. This completes the proof. □

Appendix C
Proof of Lemma 7

Proof. First, it can be readily shown that:

$$M\int_0^{\infty} \beta(t)dt = \frac{1}{4+\epsilon}n(t)dt = \frac{RF}{4+\epsilon}. \tag{54}$$

Next, we proceed to show $\sum_{j=1}^{N} \alpha_j \geq \frac{RF}{4}$. To achieve this, we consider the following two cases:

Case I: $n(t) \geq M$. In this case, it's easy to verify that $\alpha_j(t) = 0$ for $j \leq l$ and $\alpha_j(t) = \frac{j-l}{kMp_j}$ for $l < j < n(t)$. Therefore, it follows that:

$$\sum_{j=1}^{N} \alpha_j(t)p_j = \sum_{j=l+1}^{n(t)} \frac{j-l}{kM} = \frac{kM+1}{2} \geq \frac{n(t)}{4} \tag{55}$$

Case II: $n(t) < M$. In this case, we have: $\widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, t) \geq 4+\epsilon$ since we are using a resource augmentation of $(4+\epsilon)$-speed. Hence, the following equation holds:

$$\sum_{j=1}^{N} \alpha_j(t)p_j \geq \frac{1}{4}\sum_{j=1}^{n(t)} \mathbb{1}(n(t) < M) = \frac{n(t)}{4}. \tag{56}$$

As such, we have:

$$\begin{aligned}\sum_{j=1}^{N} \alpha_j p_j &= \sum_{j=1}^{N}\int_{a_j}^{c_j} \alpha_j(\tau)p_j d\tau = \int_0^{\infty}\sum_{j=1}^{N} \alpha_j(\tau)p_j d\tau\\ &\geq \frac{1}{4}\int_0^{\infty} n(\tau)d\tau = \frac{RF}{4}.\end{aligned} \tag{57}$$

The result follows combining (54) and (57). This completes the proof. □