Addressing Job Processing Variability Through Redundant Execution and Opportunistic Checkpointing: A Competitive Analysis

Huanle Xu[‡], Gustavo de Veciana[§] and Wing Cheong Lau[*]
[‡]College of Computer Science and Technology, Dongguan University of Technology
[§]Department of Electrical and Computer Engineering, The University of Texas at Austin
[*]Department of Information Engineering, The Chinese University of Hong Kong

***Abstract*—The completion times of jobs in a computing cluster may be influenced by a variety of factors including job size and machine processing variability. In this paper, we explore online resource allocation policies which combine size-dependent scheduling with redundant execution and opportunistic checkpointing to minimize the overall job flowtime. We introduce a simplified model for the job service capacity of a computing cluster while leveraging redundant execution/checkpointing. In this setting, we propose two resource allocation algorithms, SRPT+R and LAPS+R(β) subject to checkpointing overhead not exceeding the number of jobs which are processed. We provide new theoretical performance bounds for these algorithms: SRPT+R is shown to be $O(\frac{1}{\epsilon})$ competitive under $(1+\epsilon)$-speed resource augmentation, while LAPS+R(β) is shown to be $O(\frac{1}{\beta\epsilon})$ competitive under $(2+2\beta+2\epsilon)$-speed resource augmentation.**

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

I. Introduction

Job traces from large-scale computing clusters indicate that the job completion time can vary substantially [8], [9]. This variability has many sources including variability in job size and in machine processing capacity. Furthermore, the job profiles in production clusters are becoming increasingly diverse as small latency-sensitive jobs coexist with large batch processing applications which take hours to months to complete [50]. Moreover, with the size of today's computing clusters continuing to grow, intermittent/partial component failures, resource contention and congestion across networks have become a common phenomenon in cloud infrastructure [25], [32]. As a result, the actual service capacity of a machine may fluctuate significantly over the lifetime of a job. In fact, 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) on large-scale computing clusters challenging.

To deal with variability in job size, various schedulers have been proposed, which can 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 SRPT scheduler (Shortest Remaining Processing Time) is optimal for minimizing the overall/ average job flowtime [19] on a single machine in the clairvoyant setting. 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], [34], [35], [52]. Under SRPT, job sizes are known to the job scheduler upon arrival and smaller job are given priority. When only the distribution of job sizes is known, it has been shown in [4] that, Gittins index-based policy is optimal for minimizing the expected job flowtime under the Poisson Arrival Process again on a single server. The Gittins index depends on knowing the service given to-date to each job and gives priority to the job with the highest index. If the job size distribution belongs to the New-Better-than-Used-in-Expectation (NBUE) class, the Gittins index policy reduces to the First-Come-First-Served discipline. If the job size distribution is of the Decreasing-Hazard-Rate (DHR) class, this policy reduces to the Least-Attained-Service scheme. Extensions of the the Gittins index policy to the multiple-processor setting is not available.

To tackle poor job progress caused by possible machine service variability, computing clusters can exploit redundant execution wherein multiple copies of the same job execute on available machines until the first one completes. With redundancy, it is expected that one copy of the same job may complete quickly to avoid long completion times. Measurement statistics on Google's MapReduce system have shown that redundancy can reduce the average job flowtime by 44% [17]. Other large-scale parallel processing frameworks in practice have applied similar redundancy-based heuristics which have been proven to be efficient for reducing job flowtimes in real-world practical deployments [1], [7], [9], [14], [17], [30], [51].

Recently, researchers have started to investigate the effectiveness of scheduling redundant copies from a queuing perspective [15], [21], [37], [38], [41], [44]. These works assume a specific distribution of the job execution time where the service time of each job follows the same distribution. However, they do not directly account for whether job response time variability is due to the difference in job size or fluctuation in machine processing rate. Indeed, if there is no variability in machine service capacity, scheduling multiple copies of the same job may not help at all and redundancy is a waste of resource.

With the aforementioned observations in mind, in this paper, we explore the impact of variability in both job size and

Algorithm	Minimize	Machine Speed under Resource Augmentation	Upper Bound on Competitive Ratio	Technique	Support Multitasking	Use redundancy to Mitigate Machine Service Variability
SRPT+R (Section IV)	$\sum_{j=1}^{N} f_j$	$1+\epsilon$	$O(\frac{1}{\epsilon})$	Dual Fitting	N	Y
LAPS+R(β) (Section V)	$\sum_{j=1}^{N} f_j$	$2(1+\beta+\epsilon)$	$O(\frac{1}{\beta\epsilon})$	Potential Function	Y	Y
WLAPS(β), [Gupta10]	$\sum_{j=1}^{N} f_j^2$	$2+\epsilon$	$O(\frac{1}{\beta\epsilon})$	Potential Function	Y	N
LAPS(β), [Edmonds12]	$\sum_{j=1}^{N} f_j$	$1+\beta+\epsilon$	$O(\frac{1}{\beta\epsilon})$	Potential Function	Y	N
Intermediate -SRPT, [Moseley14]	$\sum_{j=1}^{N} f_j$	1	$O(1)\cdot 4^{1/(1-\alpha)} \cdot \log P$	Potential Function	Y	N

TABLE I

Summary of the results for different algorithms where f_j denotes the job flowtime. Note that only our proposed algorithms, i.e., the SRPT+R addresses the issue of variability in machine service capacity. The bound for the last algorithm, i.e., Intermediate-SRPT contains a factor, $\log P$, where P is the ratio between the largest and the smallest job size.

machine service capacity. We consider algorithms which can prioritize job scheduling and dynamically vary the number of redundant copies so as to minimize the overall job flowtime of the system. We propose a simple model for a cluster's service capacity to account for redundant execution under which a machine can preempt a running job and later resume its execution. To make use of the already completed work, we adopt the checkpointing technique [36], which is often supported by parallel processing frameworks, such as Spark, in practice to preempt, migrate and perform dynamic partitioning [42] on its active 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 case of subsequent machine failure or job preemption [10]. An additional advantage of combining checkpointing with job redundancy is that the system can propagate and clone the state of the redundant copy of a job which has made the largest progress so far 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 based on this new state after checkpointing. As such, checkpointing allows the system to opportunistically exploit variability in machine service speed.

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], [43], [45]. 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 share under both of the multitasking and non-multitasking modes.

Our Results

In a non-multitasking cluster, 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 design the LAPS+R(β) algorithm, which shares resources among a fixed fraction of the active jobs, with priority given to jobs which arrived most recently. In summary, this paper has made the following technical contributions:

- After reviewing the related work in Section II, in Section III, we propose a framework to model the impact of machine processing variability in a system exploiting redundancy/checkpointing.
- Under the resource augmentation setting [31], we apply a new dual fitting framework in Section IV to show that SRPT+R without multitasking is $(1+\epsilon)$-speed, $O(\frac{1}{\epsilon})$-competitive in terms of the overall job flowtime. To the best of our knowledge, this is the first work to analyze the performance bound of scheduling algorithms with redundancy via a dual fitting approach.
- We show in Section V that the LAPS+R(β) algorithm is $2(1+\beta+\epsilon)$-speed, $O(\frac{1}{\beta\epsilon})$-competitive by adopting the potential function argument. Moreover, our setting of the potential function is quite different from that in the previous work, e.g., [18].

Table I compares the results of our competitive ratio analysis with the related state-of-the-art.

II. Related Work

The design of job schedulers for large-scale computing clusters is currently an active research area [12], [13], [34], [35], [49], [52]. In particular, several works have derived performance bounds on algorithms geared at minimizing the total job completion time by formulating an approximate linear programming problem [12], [13], [49]. By contrast, other works, e.g., [34], [35], [52] derive performance bounds for algorithms with respect to the the total job flowtime. Leonardia et al. show in [33] 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 to minimize the overall job flowtime. Based on this lower bound, [34], [35], [52] extended the SRPT scheduler to design algorithms that minimize the overall flowtime of jobs, each consisting of multiple small tasks with precedence constraints. [12], [13], [34], [35], [49], [52] are conducted in the clairvoyant

setting where the job size is known once the job arrives. For the non-clairvoyant setting, [26]–[28] designed several multitasking algorithms (i.e., a server can serve multiple jobs simultaneously) in which machines are allocated to all active jobs and priority is given to the most-recently-arrived jobs. However, all of these studies assume accurate knowledge of machine service capacity and do not address dynamic scheduling of redundant copies for a job.

Production clusters and big data computing frameworks have adopted various approaches which use redundancy to speedup job processing. The initial Google MapReduce system launched redundant copies when a job is close to its completion [17]. Hadoop adopts another solution, i.e., LATE, which schedules a redundant copy for an active 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. Chen *et al.* propose a smart redundancy scheme in [14] to accurately estimate the task progress rate and launch redundant copies accordingly. Ananthanarayanan *et al.* propose in [7] to use redundancy for very small jobs when the extra workload is not high. As an extension to [7], Ananthanarayanan introduces GRASS [8], which carefully schedules redundant copies for approximation jobs. Moreover, Ren *et al.* propose Hopper [40] 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. However, no performance characterization has been derived 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] using a resource augmentation argument. A fundamental limitation is that these resultant bounds are not scalable as they increase linearly with the number of machines. Recently, Gardner *et al.* propose a simple model in [20] 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 work 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], [29], 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. [11], [18] and [24] propose similar algorithms which allow multitasking for jobs wherein priority is given to the most-recently-arrived-jobs. These works develop competitive performance bounds with respect to the total job flowtime. Sungjin *et al.* also provide a competitive bound for the SRPT-based parallelizable algorithm under multitasking in [29]. One limitation of [29] is that the resulting bound is potentially very large [1]. By contrast, our work is motivated by the setting where there is variability in machine service capacity.

For the analysis of SRPT+R algorithm in Section IV, 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] focus on a convex objective in the multitasking mode. By comparison, in our work, we include 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 V. Potential function is widely used to derive performance bounds with resource augmentation for online parallel scheduling algorithms e.g., [18], [29]. However, since we need to deal with redundancy and checkpointing, the design of our potential function is totally different from that in [18] and [29] which only address sublinear speedup.

III. System Model

Consider a computing cluster which consists of M servers [2]. where the servers are indexed from 1 to M. Job j arrives at 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 working 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. Thus, for all $\tau > t \geq 0$, we have $S_i(t,\tau] \leq (\tau - t) \cdot \Delta$ almost surely and $\mathbb{E}\big[S_i(t,\tau]\big] = \tau - t$.

As discussed Section I, the aim of this paper 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

[1] In [29], when $\alpha \to 0$, the competitive bound approaches ∞.

[2] Each server can either represent a CPU core or a machine.

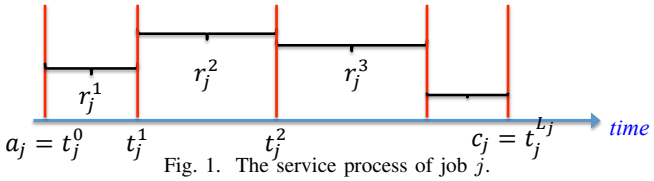

Fig. 1. The service process of job j.

copies at the most advanced state amongst the previously executing copies. In fact, we shall make the following assumption across the system:

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 III-B, we will discuss a service model which supports multitasking, i.e., a server can execute multiple jobs simultaneously.

A. 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 $(t_j^{k-1}, t_j^k]$, r_j^k redundant copies of job j are being run on different servers. Thus, together $\boldsymbol{t_j} = \left(t_j^k | k = 0, 1, \cdots, L_j\right)$ and $\boldsymbol{r_j} = \left(r_j^k | k = 1, 2, \cdots, L_j\right)$ capture the checkpoint times and the scheduled redundancy for job j.

Let $g(r,t)$ be 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., $g(r,t) - g(r-1,t) \leq g(r-1,t) - g(r-2,t)$.*

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 $r \leq M$, $g(r,t) \leq \min\{\Delta t, rt\}$.*

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. Refer to [46] for the detailed proofs of Lemma 1 and Lemma 2.

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:

$$\sum_{k=1}^{L_j} g(r_j^k, t_j^k - t_j^{k-1}) = p_j. \tag{2}$$

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

$$h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t) = \begin{cases} \frac{g(r_j^k, t_j^k - t_j^{k-1})}{t_j^k - t_j^{k-1}} & t \in (t_j^{k-1}, t_j^k], \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

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. (2) can be reformulated in terms of the speedup as follows:

$$\int_{a_j}^{c_j} h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, \tau) d\tau = p_j. \tag{4}$$

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.*

B. 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 III-A, 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 $\boldsymbol{x_j} = \left(x_j^k | k = 1, 2, \cdots, L_j\right)$ and we define another speedup function, $\widetilde{h}_j(\boldsymbol{t_j}, \boldsymbol{x_j}, \boldsymbol{r_j}, t)$, as follows:

$$\widetilde{h}_j(\boldsymbol{t_j}, \boldsymbol{x_j}, \boldsymbol{r_j}, t) = \begin{cases} x_j^k \cdot h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t) & t \in (t_j^{k-1}, t_j^k], \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Paralleling (4), the completion time of job j, c_j must satisfy the following equation:

$$\int_{a_j}^{c_j} \widetilde{h}_j(\boldsymbol{t_j}, \boldsymbol{x_j}, \boldsymbol{r_j}, \tau) d\tau = p_j. \tag{6}$$

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

C. Competitive Performance Metrics

In this paper, we will study scheduling algorithms, which involve 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 is equivalent to the standard job scheduling problem on multiple unit-speed processors with the objective of minimizing the overall job flowtime on multiprocessors [19]. However, it has been shown that the latter problem is NP-hard even when preemption and migration are allowed and no online algorithm can achieve a constant competitive ratio. As such, previous work [29], [31] has adopted a resource augmentation analysis to bound the competitive performance. Under such analysis, the performance of the offline 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. *[31] An online algorithm is δ-speed, c-competitive if the algorithm's objective is within a factor of c of the offline 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 $\widetilde{h}_j(\boldsymbol{t_j}, \boldsymbol{x_j}, \boldsymbol{r_j}, t)$, under our algorithms is δ times that under the offline optimal algorithm of the same variables.

IV. Algorithm Design and Performance Guarantee

In our model, each server can only serve one job at a time. Before going into the details of algorithm design, we first state the offline 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 $\boldsymbol{y} = (\boldsymbol{y}_j | j = 1, 2, \cdots, N)$ for all jobs. Moreover, let $\mathbb{1}(A)$ denote the indicator function that takes value 1 if A is true and 0 otherwise. The offline optimal problem formulation is as follows:

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

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

(a) Job completion: The completion time of job j, c_j, satisfies: $\int_{a_j}^{c_j} h_j(\boldsymbol{t_j}, \boldsymbol{r_j}, t)dt = p_j, \quad \forall j$.

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

(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., $L_j \in \{1, 2, \cdots, 2N-1\}$. Given a feasible L_j, the checkpoint times of job j, $\boldsymbol{t}_j$, satisfy: $\boldsymbol{t}_j \in \mathcal{T}_j^{L_j+1}$ where $\mathcal{T}_j^{L_j+1} = \left\{(t_0, t_1, \cdots, t_{L_j}) \in \mathbb{R}^{L_j+1} | a_j = t_0 < t_1 < \cdots < t_{L_j} = c_j\right\}$. Moreover, the number of redundant copies must be integer valued, 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 Problem OPT is NP-Hard, we will propose a heuristic for job scheduling, named, SRPT+R, which is a simple extension of the SRPT scheduler [19].

A. SRPT+R Algorithm

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 while 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 in the panel below.

Algorithm 1: SRPT+R Algorithm

```
while A job arrives at or departure from the system do
    Sort the jobs in the order such that
    p_1(t) ≤ p_2(t) ≤ ··· ≤ p_{n(t)}(t) and count the number
    of redundant copies being executed for job j, r_j ;
    Initialize M(t) to be the set of idle machines ;
    if n(t) < M then
        for j = 1, 2, ··· , n(t) do
            if j = 1 then
                r_j(t) = M − (n(t) − 1)⌊M/n(t)⌋;
            else
                r_j(t) = ⌊M/n(t)⌋;
            Checkpoint job j and assigns its redundant
            executions to r_j(t) machines which are
            uniformly chosen at random from
            {1, 2, ··· , M};
    if n(t) ≥ M then
        for j = 1, 2, ··· , n(t) do
            if j ≤ M then
                Checkpoint job j and assign it to one
                machine which is uniformly chosen at
                random from {1, 2, ··· , M};
            else
                Checkpoint job j;
```

B. Performance guarantee for SRPT+R

We will let OPT and SR denote the cost, i.e., the overall job flowtime, achieved by the offline optimal scheduling policy OPT, and our proposed SRPT+R algorithm respectively. Our

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

$$\Phi(\boldsymbol{y},\boldsymbol{\alpha},\boldsymbol{\beta}) = \sum_j \alpha_j p_j - M\int_0^{\infty}\beta(t)dt + \int_0^{\infty}\sum_{j:a_j\le t}\Big[(\frac{t-a_j}{p_j}+2-\alpha_j)h_j(\boldsymbol{t_j},\boldsymbol{r_j},t) + \beta(t)\sum_{k=1}^{L_j} r_j^k\cdot\mathbb{1}(t\in(t_j^{k-1},t_j^k])\Big]dt. \tag{8}$$

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.*

1) Proof of Theorem 1 and our techniques: In this section, we will prove Theorem 1 by adopting the dual fitting approach. The key step is to formulate a minimization problem whose cost serves as an approximation to the offline optimal cost, OPT with a guarantee that it is within a constant factor of OPT. To achieve this, we shall both approximate the objective of OPT and relax Constraint (d) in OPT to obtain the following problem P1:

$$\min_{\boldsymbol{y}} \sum_{j=1}^{N}\int_{a_j}^{\infty}\frac{(t-a_j+2p_j)}{p_j}\cdot h_j(\boldsymbol{t_j},\boldsymbol{r_j},t)dt \tag{P1}$$
$$\text{s.t.} \int_{a_j}^{c_j} h_j(\boldsymbol{t_j},\boldsymbol{r_j},t)dt \ge p_j, \quad \forall j,$$
$$\sum_{j:a_j\le t}\sum_{k=1}^{L_j} r_j^k\cdot\mathbb{1}(t\in(t_j^{k-1},t_j^k]) \le M, \quad \forall t,$$
$$L_j\in\{1,2,\cdots,2N-1\}, \quad \boldsymbol{t_j}\in\mathcal{T}_j^{L_j+1}, \quad \boldsymbol{r_j}\in\mathbb{N}^{L_j}, \ \forall j.$$

The following lemma shows that the optimal cost of P1, denoted by $P1$, is not far from that achieved by the offline optimal policy, OPT.

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

Refer to [46] for the detailed proof of Lemma 3.

Let $\boldsymbol{\alpha} = (\alpha_j\big|j=1,2,\cdots,N)$ and $\boldsymbol{\beta} = (\beta(t)|t\in\mathbb{R}^+)$ denote the Lagrangian dual variables corresponding to the first and second constraint in P1 respectively. The Lagrangian function associated with P1 can be written as shown in (7) with the dual problem for P1 given by:

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

Applying weak duality theory for continuous programs [39], we can conclude that the optimal value for D1 is a lower bound for $P1$. Moreover, the objective of D1 can be reformulated as in (8).

It is still difficult to solve D1 as it involves a minimization of a complex nonlinear objective function of integer 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:

$$\sum_{k=1}^{L_j} r_j^k\cdot\mathbb{1}(t\in(t_j^{k-1},t_j^k]) \ge \sum_{k=1}^{L_j}\mathbb{1}(t\in(t_j^{k-1},t_j^k])\cdot h_j(\boldsymbol{t_j},\boldsymbol{r_j},t) = h_j(\boldsymbol{t_j},\boldsymbol{r_j},t), \tag{9}$$

where the last equality is due to the fact that $h_j(\boldsymbol{t_j},\boldsymbol{r_j},t)=0$ for $t > c_j = t_j^{L_j}$. Based on (9), it can be readily shown that the second term on the R.H.S of $\Phi(\boldsymbol{y},\boldsymbol{\alpha},\boldsymbol{\beta})$ in (8) 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 $t\ge a_j$,

$$\frac{t-a_j}{p_j}+2-\alpha_j+\beta(t)\ge 0, \tag{10}$$

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 (10) would give a lower bound on D1 and result in the following optimization problem:

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

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

2) Setting of dual variables: Next, we shall find a setting of dual variables, i.e., $(\boldsymbol{\alpha},\boldsymbol{\beta})$ that is feasible for P2. Moreover, the cost of this setting should relate to SR. To achieve this, we set the dual variables in the sequel based on another scheduling policy, SRPT, which uses a $(1+\epsilon)$-speed resource augmentation.

Observe that SRPT+R and SRPT only differ when the number of active jobs is less than 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 upper bound for SR.

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

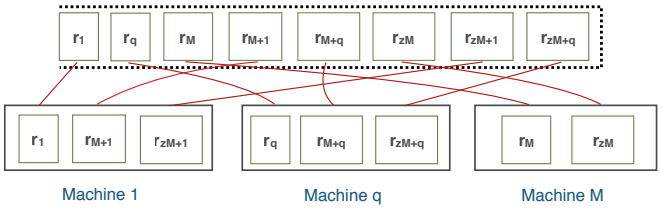

Fig. 2. The scheduling process of SRPT at time a_j where $n(a_j) = zM + q$ and there are no further job arrivals after a_j. Jobs are sorted based on the remaining size, which is denoted by r_j for job j, i.e., $r_j = p_j(a_j)$. Jobs indexed by $kM + i$ for some integer valued k and i are assigned to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

job j under SRPT with $(1+\epsilon)$-speed resource augmentation respectively.

Let $\Theta_j = \{k : a_k \le a_j \le c_k\}$, i.e., the set of jobs that are active when job j arrives and $A_j = \{k \ne j : k \in \Theta_j \text{ and } p_k(a_j) \le p_j\}$, i.e., jobs whose residual workload upon job j's arrival is less than job j's processing requirement, and let $\rho_j = |A_j|$. Our setting of dual variables is illustrated in the following lemma.

Lemma 4. *For all $j \in \{1, 2, \cdots, N\}$ and $t \ge 0$, let*

$$\alpha_j = \frac{1}{(1+\epsilon)p_j} \sum_{k=1}^{\rho_j} \Big(\lfloor \frac{n(a_j) - k}{M} \rfloor - \lfloor \frac{n(a_j) - k - 1}{M} \rfloor \Big) p_k(a_j) + \frac{1}{1+\epsilon} \Big(\lfloor \frac{n(a_j) - \rho_j - 1}{M} \rfloor + 1 \Big), \tag{11}$$

and

$$\beta(t) = \frac{1}{(1+\epsilon)M} n(t). \tag{12}$$

This setting is feasible for Problem P2, i.e., $\alpha_j, \beta(t) \ge 0$ and $\alpha_j - \beta(t) \le \frac{t - a_j}{p_j} + 2$.

Refer to [46] for the detailed proof of Lemma 4.

3) Performance bound of SRPT+R: To bound the cost associated with the choices for the dual variables in Lemma 4, we first show a lemma, which quantifies the total job flowtime under SRPT in the transient case when there are no job arrivals after time t.

Lemma 5. *Suppose there are $n(t)$ jobs in the system at time t such that their remaining workload are $p_1(t) \le p_2(t) \le \cdots, \le p_{n(t)}(t)$ and there are no further arrivals after time t, then, the overall remaining job flowtime under SRPT scheduling, $F(t)$, is given by:*

$$F(t) = \frac{1}{1+\epsilon} \sum_{j=1}^{n(t)} (\lfloor \frac{n(t) - j}{M} \rfloor + 1) p_j(t). \tag{13}$$

Refer to [46] for the detailed proof of Lemma 5, where the main idea is to construct the exact scheduling process under SRPT when job j arrives the cluster at time a_j and no further job arrives as illustrated in Fig. 2.

Based on Lemma 5, if job j never arrives at the cluster and the subsequent jobs do not enter the cluster, the overall remaining job flowtime at time a_j under SRPT is given by:

$$F_j'(a_j) = \frac{1}{1+\epsilon} \sum_{k=1}^{n(a_j)-1} (\lfloor \frac{n(a_j) - 1 - k}{M} \rfloor + 1) p_k(a_j). \tag{14}$$

By contrast, when job j arrives at time a_j but the subsequent jobs do not enter the cluster, the overall remaining job flowtime at time a_j under SRPT is given by:

$$\begin{aligned} F_j(a_j) = & \frac{1}{1+\epsilon} \sum_{k=1}^{\rho_j} (\lfloor \frac{n(a_j) - k}{M} \rfloor + 1) p_k(a_j) \\ & + \frac{1}{1+\epsilon} \Big(\lfloor \frac{n(a_j) - \rho_j - 1}{M} \rfloor + 1 \Big) p_j \\ & + \frac{1}{1+\epsilon} \sum_{k=\rho_j+2}^{n(a_j)} (\lfloor \frac{n(a_j) - k}{M} \rfloor + 1) p_k(a_j). \end{aligned} \tag{15}$$

Therefore, one can view α_j as the difference of (15) and (14), i.e., the incremental increase of the overall job flowtime caused by the arrival of job j and divided by $(1+\epsilon)p_j$. As a result, $\sum_j p_j \alpha_j$ corresponds to the overall job flowtime under SRPT, i.e., $\sum_j \alpha_j p_j = SRPT$. Moreover, $(1+\epsilon)M\beta(t)$ is the number of active jobs in the cluster at time t, so, $M \int_0^\infty \beta(t)dt = \frac{1}{1+\epsilon} SRPT$. 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 $\frac{\epsilon}{1+\epsilon} SR \le \frac{\epsilon}{1+\epsilon} SRPT \le P2 \le P1 \le \big(1 + 2\Delta\big) \cdot OPT$. This implies $SR \le O(\frac{1}{\epsilon}) OPT$ and completes the proof of Theorem 1. □

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, 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.*

V. Algorithm Design for Multitasking Processors

In this section, we design a scheduling algorithm for clusters supporting the multitasking mode. Besides, checkpoint times and level of redundancy, there is another decision set of variables, $\boldsymbol{x} = (\boldsymbol{x_j} : j = 1, 2, \cdots, N)$ where $\boldsymbol{x_j} = (x_j^k | k = 1, 2, \cdots, L_j)$, i.e, the fractions of resource shares to be allocated to each job during checkpointing intervals. To be specific, we propose the LAPS+R(β) algorithm which is an extension of LAPS (the Latest Arrival Processor Sharing algorithm). In LAPS, resource is shared among active jobs in the cluster and priority is given to those which arrived most recently. However, the LAPS algorithm proposed in [18] only considers resource sharing without redundant executions. Our LAPS+R(β) algorithm generalizes this to allow redundant copies of jobs to be varied dynamically. In this section, we assume, without loss of generality, that jobs have been ordered such that $a_1 \ge a_2 \ge \cdots \ge a_{n(t)}$, i.e., from the oldest job to the most recently arrived one.

A. LAPS+R(β) Algorithm

The algorithm depends on the parameter $\beta \in (0, 1)$. Say, when $\beta = 1/2$, the algorithm essentially schedules the $\frac{1}{2}n(t)$ most-recently-arrived jobs. If there are fewer than M such

Algorithm 2: LAPS+R(β) Algorithm

while *A job arrives at or departure from the system* **do**
  Sort the jobs in the order such that $a_1 \geq a_2 \geq \cdots \geq a_{n(t)}$;
  Compute $\beta n(t) = zM + \alpha + \gamma$ where $\gamma < 1$ and $\alpha < M$;
  if $z \geq 1$ **then**
    $r_{n(t)}(t) = (M - \alpha)$ and $x_{n(t)}(t) = \frac{1}{z+1}$;
    for $j = n(t) - zM - \alpha, \cdots, n(t) - 1$ **do**
      $r_j(t) = 1$ and $x_j(t) = \frac{1}{z+1}$;
  if $z < 1$ **then**
    $r_{n(t)}(t) = M - \alpha\lfloor\frac{M}{\alpha+1}\rfloor$ and $x_{n(t)}(t) = 1$;
    for $j = n(t) - \alpha, \cdots, n(t) - 1$ **do**
      $r_j(t) = \lfloor\frac{M}{\alpha+1}\rfloor$ and $x_j(t) = 1$;
  for $j = 1, 2, \cdots, n(t) - zM - \alpha - 1$ **do**
    $x_j(t) = r_j(t) = 0$;
  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 $x_j(t)$;

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 $n(t)$, let 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 $\frac{1}{z+1}$. Furthermore, 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 2 in the panel above.

B. Performance guarantee for LAPS+R(β) and our techniques

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

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

Proof Sketch. The dual fitting approach fails in this setting to prove Theorem 2, so we adopt the potential function analysis approach. The main idea of this method is to design a potential function which tracks the difference between the offline optimal schedule and LAPS+R(β).

Consider a checkpointing trajectory for job j under LAPS+R(β) and the offline 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 offline optimal scheduling and denote by $\psi(t)$ the set of jobs that are active under LAPS+R(β). Thus, we have: $|\psi(t)| = n(t)$. 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:

$$\pi_j(t) = \max\Big[\int_{a_j}^{t} (\widetilde{h}_j(\boldsymbol{t}_j^*, \boldsymbol{x}_j^*, \boldsymbol{r}_j^*, \tau - \widetilde{h}_j(\boldsymbol{t}_j, \boldsymbol{x}_j, \boldsymbol{r}_j, \tau))d\tau\ ,\ 0\Big], \tag{16}$$

Let $\delta = 2(1 + \beta + \epsilon)$ and define

$$f(n_j(t)) = \begin{cases} 1 & \beta n_j(t) \leq M, \\ \frac{M}{\beta n_j(t)} & \text{otherwise.} \end{cases} \tag{17}$$

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 LAPS+R(β) would allocate to job j if $n_j(t)$ jobs were active at time t. We define a potential function, $\Lambda(t)$, whose evolution may have jumps as well as continuous change. Our potential function is given by:

$$\Lambda(t) = \sum_{j \in \psi(t)} \Lambda_j(t), \tag{18}$$

where $\Lambda_j(t)$ is the ratio between (16) and (17), i.e., $\Lambda_j(t) = \frac{\pi_j(t)}{\delta \cdot f(n_j(t))}$.

Let $LR(t)$ and $OPT(t)$ denote the accumulated job flowtime under LAPS+R(β) with a $2(1 + \beta + \epsilon)$-speed resource augmentation and the offline optimal schedule, respectively. We show that $\Lambda(t)$ satisfies the following three properties:

- Boundary Condition: $\Lambda(0) = \Lambda(\infty) = 0$.
- Jump 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 correspond to a decrease in the value of the potential function..
- Drift Condition: with a $2(1 + \beta + \epsilon)$-speed resource augmentation, for any time t not corresponding to a jump, and some constant c_1 and c_2, we have:

$$\frac{d\Lambda(t)}{dt} \leq -\frac{\beta\epsilon}{c_1} \cdot \frac{dLR(t)}{dt} + \frac{1}{c_2} \cdot \frac{dOPT(t)}{dt}. \tag{19}$$

By integrating (19) and accounting for the negative jump and the boundary condition, one can see that the existence of such a potential function guarantees that $LR \leq \frac{c_1}{c_2} \cdot \frac{1}{\beta\epsilon} OPT$ under a $2(1 + \beta + \epsilon)$-speed resource augmentation.

Refer to [46] for the detail proof of Theorem 2.

VI. Conclusions and Future Directions

This paper is the first attempt to address the impact of two key sources of variability in parallel computing clusters: job size and machine processing variability. Our primary aim and contribution was to provide the fundamental understanding on how job scheduling and redundant execution/ checkpointing algorithms 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 *IEEE Infocom*, March 2011.

[13] F. Chen, M. Kodialam, and T. V. 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, S. Zbarsky, Pittsburgh, S. Doroudi, M. Harchol-Balter, and E. H. Aalto. Reducing latency via redundant requests: Exact analysis. pages 347–360. ACM SIGMETRICS, June 2015.

[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 *ACM SPAA*, 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*, 2015.

[27] S. Im, J. Kulkarni, and K. Munagala. Competitive algorithms from competitive equilibria: non-clairvoyant scheduling under polyhedral constraints. In *Proceedings of STOC*, 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, K. Pruhs, and E. Torng. Competitively scheduling tasks with intermediate parallelizability. In *SPAA*, June 2014.

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

[31] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47:214–221, 2000.

[32] 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.

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

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

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

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

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

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

[39] 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.

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

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

[42] 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.

[43] 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.

[44] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Low latency via redundancy. In *International Conference on emerging Networking EXperiments and Technologies(CoNEXT)*, 2013.

[45] 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.

[46] H. Xu, G. de Veciana, and W. C. Lau. Addressing job processing variability through redundant execution and opportunistic checkpointing: A competitive analysis. https://www.dropbox.com/s/fp61cpqns3sbliq/podc_backup.pdf?dl=0.

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

[50] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: fault-tolerant streaming computation at scale. In *ACM Symposium on Operating Systems Principles (SOSP)*, pages 423–438, 2013.

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

[52] 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.