Optimization for Speculative Execution in Big Data Processing Clusters

Huanle Xu, *Student Member, IEEE* and Wing Cheong Lau, *Senior Member, IEEE*

Abstract—A big parallel processing job can be delayed substantially as long as one of its many tasks is being assigned to an unreliable or congested machine. To tackle this so-called straggler problem, most parallel processing frameworks such as MapReduce have adopted various strategies under which the system may speculatively launch additional copies of the same task if its progress is abnormally slow when extra idling resource is available. In this paper, we focus on the design of speculative execution schemes for parallel processing clusters from an optimization perspective under different loading conditions. For the lightly loaded case, we analyze and propose one cloning scheme, namely, the Smart Cloning Algorithm (SCA) which is based on maximizing the overall system utility. We also derive the workload threshold under which SCA should be used for speculative execution. For the heavily loaded case, we propose the Enhanced Speculative Execution (ESE) algorithm which is an extension of the Microsoft Mantri scheme to mitigate stragglers. Our simulation results show SCA reduces the total job flowtime, i.e., the job delay/ response time by nearly 6 percent comparing to the speculative execution strategy of Microsoft Mantri. In addition, we show that the ESE Algorithm outperforms the Mantri baseline scheme by 71 percent in terms of the job flowtime while consuming the same amount of computation resource.

Index Terms—Job scheduling, speculative execution, cloning, straggler detection, optimization

1 Introduction

Empirical performance studies of large-scale computing clusters have indicated that the completion time of a job [7] is often significantly and unnecessarily prolonged by one or a few so-called "stragglers" or straggling tasks, i.e., tasks which are unfortunately assigned to either a failing or overloaded server within a cluster of hundreds of thousands of commodity servers.

To mitigate stragglers, recent big data frameworks such as the MapReduce system or its variants have adopted various preventive or reactive speculation strategies under which the system launches extra (backup) copies of a task on alternative machines in a judicious manner. In particular, there exist two main classes of speculative execution strategies, namely, the Cloning approach [5] and the Straggler-Detection-based one [6], [7], [13], [16], [20], [31], [34], [41]. Under the Cloning approach, extra copies of a task are scheduled in parallel with the initial task as long as the computation cost of the task is expected to be low and the system resource is available. For the Straggler-Detection-based approach, the progress of each task is monitored by the system and backup copies are launched only when a straggler is detected.

As one may expect, the cloning-based strategy is only suitable for a lightly loaded cluster as it launches the clones in a greedy, indiscriminately fashion. On the other hand, the straggler-detection based strategy is applicable to both the lightly-loaded and heavily-loaded regimes but at the expense of extra system instrumentation and performance overhead as discussed in [10]. The situation is particularly challenging when the progress of a large number of tasks have to be tracked. However, previous works do not compare the performance between these two different speculation approaches. Furthermore, most of the existing speculative execution schemes are based on simple heuristics and do not consider the optimization based on specific performance objectives.

With the aforementioned observations in mind, in this paper, we take a more systematic, optimization-based approach for the design and analysis of speculative execution schemes. Our objective is to optimize two performance metrics which are the total job delay/ response time (which is also referred as job flowtime) and the computation cost by defining utility functions. The optimizations are conducted by coordinating speculating with job scheduling, which is an opportunity to gain significant performance improvement compared to speculation-only policies. We also characterize the differences between the Cloning approach and the Straggler-Detection based speculative execution scheme through both theoretical analysis and extensive simulations. In particular, we have made the following technical contributions:

- After reviewing the related work in Section 2, we introduce the system model in Section 3 and propose a generalized optimization framework to maximize the overall system utility which is a combination of the overall job flowtimes and computation costs.
- We derive the cut-off workload threshold between the lightly-loaded and heavily-loaded operating regimes of a computing cluster in Section 4. Based on this workload threshold, the applicability of a speculative execution strategy can be analyzed for different operating regimes.

- *H. Xu and W.C. Lau are with the Department of Information Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. E-mail: {xh112, wclau}@ie.cuhk.edu.hk.*

Manuscript received 2 Oct. 2015; revised 2 May 2016; accepted 4 May 2016. Date of publication 9 May 2016; date of current version 18 Jan. 2017. Recommended for acceptance by R. Kwok.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2016.2564962

- In Section 5, we present a specific Smart Cloning Algorithm (SCA), which serves as an approximate solution to our optimization framework in a lightly-load cluster. We also propose the Enhanced Speculative Execution (ESE) algorithm for a heavily loaded cluster in Section 6 by extending the speculative execution strategy of Microsoft Mantri [7].
- In Section 7, we conduct extensive trace-driven simulations to compare the performance of SCA and ESE with the Microsoft Mantri's scheme. We show that both SCA and ESE reduce the average job flowtime substantially. In addition, ESE only consumes a small amount of computation resources. Before summarizing our findings and concluding the paper in Section 9, we also discuss how to implement our proposed algorithms under Hadoop Yarn in Section 8.

2 Background and Related Work

In this section, we begin by giving a brief introduction to job scheduling in big data processing clusters and then present an overview of existing schedulers proposed by both academia and industrial research. After that, we review the related work of speculative execution in these clusters.

2.1 Job Scheduling in a MapReduce-Like Cluster

In a big data processing cluster like MapReduce and its variants or derivatives, different applications/ jobs need to share and compete for resources in the cluster. Thus, job scheduling plays a very important role. Throughout the whole paper, we only consider the centralized scheduling paradigm under which a global scheduler of the cluster manages all jobs where each job may consist of many small tasks. The scheduler allocates resources across jobs and also handles straggling tasks.

Widely deployed schedulers to-date include the fair scheduler [4] and the capacity scheduler [3]. However, the main goal of these schedulers is to provide fair and efficient resource sharing among different organizations. As such, other key performance metrics such as the job response time have not received adequate considerations under their designs.

To enhance system performance, the design of job schedulers for MapReduce-like systems has been an active research area lately [11], [12], [23], [25], [35], [40], [42]. In particular, several works focus on deriving performance bounds for minimizing the total job completion time [11], [12], [40]. Tan et al. design the *Coupling scheduler* [35], which mitigates the starvation problem caused by reduce tasks in large jobs. It is well known in scheduling literature that the SRPT (Shortest Remaining Processing Time) scheduler is optimal for minimizing the overall flowtime on a single machine where there is one task per job. As such, some works extend the SRPT scheduler to minimize the total job flowtime in different settings [23], [25], [40], [42]. However, all of these studies assume accurate knowledge of task durations and hence do not support speculative copies to be dynamically scheduled.

2.2 Speculative Execution Policies

Several speculative execution strategies have been proposed for MapReduce-like systems. The initial Google MapReduce system only begins to launch backup tasks when a job is close to completion [16]. This scheme is easy to implement but it would unnecessarily launch backup copies for tasks of normal progress.

The speculative execution strategies in the initial versions of Hadoop [2] and Microsoft Dryad [20] closely follow that of the Google MapReduce system. However, Zaharia et al. present a new strategy called LATE (*Longest Approximate Time to End*) in [41] for the Hadoop-0.21 implementation. It monitors the progress rate of each task and estimates their remaining time to completion. Tasks with progress rate below certain threshold are chosen as backup candidates and the one with the longest remaining time is given the highest priority. The system also imposes a limit on the maximum number of backup tasks in the cluster. By contrast, Microsoft Mantri [7] proposes a new speculative execution strategy for Dryad in which the system estimates the remaining time to finish (i.e., t_{rem}), for each task and predicts the required service time of a relaunched copy of the task (i.e., t_{new}). Once a server becomes available, the Mantri system makes a decision on whether to launch a backup task based on the statistics of t_{rem} and t_{new}. Mantri would schedule a duplicate if the total computation cost is expected to decrease while it does not explore the trade-offs between the job completion time (flowtime) and the computation cost.

To accurately and promptly identify stragglers, Chen et al. propose a Smart Speculative Execution strategy in [13] and Sun et al. present an Enhanced Self-Adaptive MapReduce Scheduling Algorithm in [34]. The limitation is that those works only focus on the optimization of task level rather than job level performance.

Ananthanarayanan et al. proposes to mitigate the straggler problem by cloning every small job and avoid the extra delay caused by the straggler monitoring/detection process [5]. When most of the jobs in the system are small, the cloned copies only consume a small amount of additional resources. As an extension from [5], Ananthanarayanan further presents GRASS [6], which carefully adopts the Detection-based approach to trim stragglers for approximation jobs. GRASS also provides a unified solution for normal jobs. Recently, Ren et al. propose Hopper [31], a speculation-aware scheduler, which coordinates job scheduling with speculative execution. In Hopper, the scheduler allocates computing slots based on the virtual job size, which is larger than the actual size, and can immediately schedule a speculative copy once a straggler is detected.

For most of the speculative execution schemes presented above, the speculation algorithms are designed independently of job scheduling. Hopper and the recently proposed SRPTMS+C [38] are the only exceptions. However, Hopper still has several downsides that can degrade the cluster performance. First, Hopper is non-work-conserving: it is possible for its scheduler to keep a computing slot idle as a reservation for a future straggler while other jobs/tasks already queue up for computation resource.[1] Second, the job size is computed/estimated based on only the number of tasks instead of taking the product with the task service time (i.e., the time between the task is launched and the task

1. Refer to Fig. 2 of [31] for an example.

TABLE 1
The Notations for the Job Service Parameters

Notation	Corresponding meaning
λ	Job arrival rate.
a_i	Arrival time of job J_i.
m_i	Number of tasks consisted in job J_i.
C_i	Time when job J_i completes its work.
Γ_i	Flowtime of job J_i.
X_i^j	Service time of task δ_i^j without speculative execution.
c_i^j	Total number of copies scheduled for task δ_i^j.
$w_i^{j,k}$	Time when the kth copy of task δ_i^j is scheduled.
$T_i^{j,k}$	Service time of the kth copy scheduled for task δ_i^j.
T_i^j	Service time of task δ_i^j under cloning.
C_i^j	Completion time of task δ_i^j under speculative execution.
$F_i(x)$	The CDF function of X_i^j.
ξ	The maximum number of copies allowed for a task.

is finished). In practice, the task service times have shown to be varying widely even among tasks of the same job. (e.g., a Map task versus a Reduce task). As a comparison, in our work, we incorporate the task service time when estimating the job size. Moreover, SRPTMS+C is limited to investigate the cloning approach only whereas the work in this paper combines job scheduling with speculative execution and judiciously applies proactive cloning or reactive speculation under different operating regimes.

Another body of work related to this paper investigate a study on the effectiveness of scheduling redundant copies from a queuing perspective. In particular, Vulimiri et al. characterize when a global redundancy policy improves latency performance of the whole system [37]. However, this work does not consider killing the unfinished copies of the same task. Chen et al. adopts the approach of redundant requests in storage codes and theoretically analyzes its optimality when the service time of each request is exponentially distributed [14]. Based on these works, Qiu et al. adopt the MAP model to represent task arrivals and study the distribution of task-response time when redundancy is applied [26], [27], [28], [29]. Moreover, Gardner et al. present in [17] an exact analysis of systems with redundancy when the service time for the redundant class follows an exponential distribution. One fundamental limitation of [14], [17], [26], [27], [28], [29] is that they do not theoretically characterize the efficiency of redundancy when the task service time follows a more general distribution. Besides exponential distribution, [21] and [33] also analyze how different redundancy strategy can influence the latency and the computation cost when the job service time follows a heavy-everywhere or light-everywhere distribution. However, their derived results do not hold when the service time follows other hevay-tailed distributions (e.g., the Pareto Distribution) and thus can not be applied to our work.

3 System Model

Consider a big data processing cluster which consists of M servers (machines). A set of jobs $J = \{J_1, J_2, \ldots, J_N\}$ arrive at this cluster at a rate of λ jobs per unit time and the arrival time of job J_i is denoted by a_i. Upon arrival, job J_i joins a global queue managed by a scheduler, waiting to be scheduled for execution. Job J_i consists of a set of m_i tasks where m_i is a deterministic number and we let δ_i^j denote the jth task of job J_i.

In this cluster, each server can only hold one task at any time. We let X_i^j, which is a random variable, denote the service time (i.e., duration) of task δ_i^j without any speculative execution. For all $j \in \{1, 2, \ldots, m_i\}$, X_i^j follows the same distribution, which is characterized by the cumulative distribution function (CDF), $F_i(x)$, i.e., $\Pr(X_i^j < x) = F_i(x)$. In this paper, we shall make the following assumption for $F_i(x)$:

Assumption 1. *For all $i \in \{1, 2, \ldots, N\}$, $F_i(x)$ is known to the scheduler when job J_i arrives to the cluster.*

Usually, $F_i(x)$ can be estimated based on the application job J_i runs and the size of the input data of J_i. Moreover, we consider in this paper that $F_i(x)$'s are different across jobs.

3.1 Job Service Process under Speculative Execution

In this paper, we consider that time is slotted and task preemption is not allowed so as to limit the system overhead. The scheduler makes scheduling decisions, i.e., which tasks to schedule on idle machines along with how many copies to launch at the beginning of each time slot. All the scheduling variables are summarized in Table 1.

Let c_i^j denote the total number of copies scheduled for task δ_i^j where the kth copy is launched at time $w_i^{j,k}$. Thus, we have the following constraint for $w_i^{j,k}$:

$$w_i^{j,k} \in \mathbb{N} \text{ and } w_i^{j,k} \geq a_i, \ \forall 1 \leq i \leq N, 1 \leq j \leq m_i, 1 \leq k \leq c_i^j. \tag{1}$$

In addition, we let $T_i^{j,k}$ denote the service time of the kth copy for task δ_i^j. To simplify modeling, we consider that, for $1 \leq k \leq c_i^j$, $T_i^{j,k}$'s are i.i.d random variables and $\Pr\{T_i^{j,k} \leq x\} = F_i(x)$. Once a copy finishes, the task completes and all the other replicas for the same task are killed. Therefore, the completion time of task δ_i^j, denoted by C_i^j, satisfies:

$$C_i^j = \min_{k \in \{1,2,\ldots,c_i^j\}} (w_i^{j,k} + T_i^{j,k}), \ \forall 1 \leq i \leq N, 1 \leq j \leq m_i. \tag{2}$$

Let C_i denote the time that Job J_i leaves the system, i.e., when its last task completes. Therefore, $C_i = \max_{j=1,2,\ldots,m_i} C_i^j$ and the job flowtime, denoted by $\Gamma_i = C_i - a_i$, can be formulated as follows:

$$\Gamma_i = \max_{j=1,2,\ldots,m_i} C_i^j - a_i, \quad \forall 1 \leq i \leq N. \tag{3}$$

3.2 Problem Formulation

In this paper, we focus on two performance metrics, i.e., the job flowtime, Γ_i, and the computation cost [21], which is defined as the overall time spent by the servers serving job J_i, i.e., $\sum_{j=1}^{m_i} \sum_{k=1}^{c_i^j} (\mathbb{E}[C_i^j] - w_i^{j,k})$.

Usually, the job flowtime is the service metric that a user cares about, while the computation cost is directly related to the cost to be paid by the user. However, On the one hand, the expected task completion time under speculative execution, i.e., C_i^j, decreases in the number of its scheduled

copies, c_i^j. And the marginal decrease of C_i^j also decreases in c_i^j, thus, we consider that each task would be scheduled at most ξ copies where ξ is a system parameter determined by the scheduler, i.e., $c_i^j \leq \xi$. On the other hand, the expected computation cost of task δ_i^j usually increases in c_i^j [21]. As such, the two performance metrics we address are difficult to be optimized at the same time (the detection approach is an exception). Therefore, we define a utility function for each job to trade-off between these two metrics, i.e., $U_i = -(\mathbb{E}[\Gamma_i] + \gamma \cdot \sum_{j=1}^{m_i} \sum_{k=1}^{c_i^j} (\mathbb{E}[C_i^j] - w_i^{j,k}))$ where γ can be interpreted as a price paid for using the computation resources, though rescaled to be comparable with the job flowtime.

Let $z_i = \{(c_i^j, w_i^{j,1}, \ldots, w_i^{j,c_i^j}) | j = 1, 2, \ldots, m_i\}$ and $z = \{z_i | i = 1, 2, \ldots, N\}$, then, our objective becomes to maximize the total utility of all jobs in the cluster via determining z, which yields the optimization problem, P1, as shown below:

$$\min_{z} \sum_{i=1}^{N} \mathbb{E}[\Gamma_i] + \gamma \cdot \sum_{i=1}^{N} \sum_{j=1}^{m_i} \sum_{k=1}^{c_i^j} (\mathbb{E}[C_i^j] - w_i^{j,k}) \tag{P1}$$

$$\text{s.t.} \sum_{i=1}^{N} \sum_{j=1}^{m_i} \sum_{k=1}^{c_i^j} \mathbb{1}(w_i^{j,k} \leq l) \cdot \mathbb{1}(w_i^{j,k} + T_i^{j,k} > l) \leq M, \quad \forall l,$$

$$c_i^j \in \{1, 2, \ldots, \xi\}, \ \forall 1 \leq i \leq N, 1 \leq j \leq m_i,$$

$$\text{Constraints } (1), (2), (3),$$

where $\mathbb{1}(A)$ denotes the indicator function that takes value 1 if A is true and 0 otherwise. The first constraint states that the total number of task copies executed at any time slot is no more than M.

Remark 1. One can set the value of γ in P1 to be very small if job flowtime is of greater concern than the computation cost and vice versa. Although similar trade-offs can be controlled via the other parameter ξ, namely, a small value of ξ will increase job flowtime while lowering computation cost, the impact of ξ is expected to diminish after its value increases beyond a certain threshold. After that, the effects of γ is expected to become dominant.

P1 is an online stochastic optimization problem, whose solutions involve determining the scheduling time for each task as well as the number of cloned copies. Furthermore, the optimization problem of minimizing the total flowtime of N jobs that are released over time on a single machine without preemption [22] is reducible to P1. Since the former problem is strongly NP-hard, we conclude that P1 is also strongly NP-hard.

4 Deriving the Cutoff Threshold for Different Operating Regimes

Since P1 is strongly NP-hard, we shall find approximation solutions to tackle it. As discussed in Section 1, there exist two classes of speculation strategies, i.e., the Cloning approach and Detection-based one. However, the Cloning-based strategy launches all copies for a task simultaneously [5]. Otherwise, the later scheduled copy can easily lag behind those earlier launched ones and thus make cloning a waste of resource. As such, the cloning scheme is only suitable for a lightly loaded cluster. By contrast, the Straggler-detection based strategy launches duplicates in a judicious manner and is applicable to any loading conditions. Therefore, there exists a cutoff threshold to separate the cluster workload into the lightly-loaded and heavily-loaded regimes. When the workload is below this threshold, cloning can yield good system performance in terms of the average job flowtime. Conversely, when the workload exceeds this threshold, only the Straggler-Detection-based strategy can help to reduce the job flowtime.

We will derive the cutoff workload threshold, λ^U, which allows us to separate our subsequent analysis into the lightly loaded versus heavily loaded regimes. To simplify the analysis, we shall focus on the mean task delay in the cluster, that is, when $\lambda < \lambda^U$, cloning should not overload the system and must reduce the mean task delay as well.

We approximate the task arrival as a Poisson process with rate $\overline{m} \cdot \lambda$ where $\overline{m} = \sum_{i=1}^{N} m_i / N$. In addition, we consider that each task should have at least two copies since the task which does not have any extra copy scheduled may delay the completion of the entire job.

4.1 A First Upper Bound for λ^U

Let T_i^j denote the service time of task δ_i^j under the cloning approach. Thus, $T_i^j = \min_{k=1,2,\ldots,c_i^j} T_i^{j,k}$ since all copies of task δ_i^j are launched together. As such, the cumulative distribution function of T_i^j is given by,

$$\begin{aligned} \Pr\{T_i^j \leq x\} &= 1 - \Pr(T_i^j > x) \\ &= 1 - \prod_{k=1}^{c_i^j} \Pr(T_i^{j,k} > x) = 1 - (1 - F_i(x))^{c_i^j}. \end{aligned} \tag{4}$$

Thus, the mean of T_i^j is characterized by:

$$\mathbb{E}[T_i^j] = \int_0^\infty x d(1 - (1 - F_i(x))^{c_i^j}) = \int_0^\infty (1 - F_i(x))^{c_i^j} dx. \tag{5}$$

Note that the mean of the total time spent by all machines serving task δ_i^j is equal to $c_i^j \mathbb{E}[T_i^j]$. Hence, the mean of the total time spent by the cluster serving all jobs, Θ, is given by,

$$\Theta = \mathbb{E}\Big[\sum_{i=1}^{N} \sum_{j=1}^{m_i} c_i^j T_i^j\Big] = \sum_{i=1}^{N} \sum_{j=1}^{m_i} c_i^j \mathbb{E}\left[T_i^j\right],$$

and the mean of the total service time for a single job is Θ/N.

Since there are M servers in the cluster, the average job processing rate is given by,

$$\frac{M}{\Theta/N} = \frac{NM}{\sum_{i=1}^{N} \sum_{j=1}^{m_i} c_i^j \mathbb{E}\left[T_i^j\right]}.$$

To keep the system not overloaded, the job arrival rate must be bounded by the job processing rate, which then yields the first upper bound, λ_1, for λ^U, i.e.,

$$\lambda_1 = \frac{NM}{\sum_{i=1}^{N}\sum_{j=1}^{m_i} c_i^j \mathbb{E}\left[T_i^j\right]}. \tag{6}$$

4.2 A Second Upper Bound for λ^U

Nevertheless, the efficiency of cloning is not guaranteed by (6). An efficient cloning strategy should have a smaller task delay than a strategy which does not make speculative execution. This argument must also hold when each task has only two copies.

The task service process can be modeled as an M/G/N queue, however, the exact analysis for the mean response time has been an open problem. However, a well-known approximation in [18] involves evaluating an M/M/N queue using a complicated Erlang-C model, which is not applicable to our analysis. Therefore, we approximate this M/G/N queue by considering the task service process on each individual machine, which is modeled using an M/G/1 queue with arrival rate, $\lambda_t = \frac{\lambda \overline{m}}{M}$ where $\overline{m} = \frac{\sum_{i=1}^{N} m_i}{N}$.

Applying the Pollaczek-Khinchine formula in [15], the average task delay without speculative execution, W, is given by:

$$W = \frac{\lambda_t \mathbb{E}[S^2]}{2(1-\lambda_t \mathbb{E}[S])} + \mathbb{E}[S]. \tag{7}$$

where S is the single task delay without speculative execution and its mean and second moment are given in [39].

With cloning, the equivalent task arrival rate to each machine, is $\lambda_c = 2\lambda_t$ since each task has two copies. Denote by W_c the average task delay under cloning, thus, W_c is given by,

$$W_c = \frac{\lambda_t \mathbb{E}[S_c^2]}{1-2\lambda_t \mathbb{E}[S_c]} + \mathbb{E}[S_c], \tag{8}$$

where S_c is the task delay when a task is launched two copies and its mean and second moment are given in [39].

Under the $M/G/1$ approximation, the efficiency of cloning is guaranteed by $W_c \leq W$, i.e.,

$$\frac{\lambda_t \mathbb{E}[S_c^2]}{2(1-\lambda_t \mathbb{E}[S_c])} + \mathbb{E}[S_c] \leq \frac{\lambda_t \mathbb{E}[S^2]}{1-2\lambda_t \mathbb{E}[S]} + \mathbb{E}[S]. \tag{9}$$

Let λ_t^* be the solution to the corresponding equality of (9), i.e., $W_c = W$, thus, $\lambda_2 = \frac{\lambda_t^* M}{\overline{m}}$ gives a second upper bound for λ^U.

4.3 Pareto Distribution as an Illustrative Example

It is shown in [5], [6], [31], [32] that, task durations in big data computing clusters often follow Pareto tails, i.e., $\Pr(X_i^j > x) = \theta(x^{-\beta})$. In particular, [6] has suggested that the shape parameter, β for Facebook and Bing traces is 1.259. Therefore, we use the following Pareto distribution to better illustrate the derived bounds in (6) and (9):

$$F_i(x) = \begin{cases} 1 - (\frac{\mu_i}{x})^{\alpha} & x \geq \mu_i, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

Substitute (10) into (5), it follows that $\mathbb{E}[T_i^j] = \frac{\mu_i c_i^j \alpha}{c_i^j \alpha - 1}$ and $c_i^j \mathbb{E}[T_i^j] = \frac{\mu_i (c_i^j)^2 \alpha}{c_i^j \alpha - 1}$. Since $c_i^j \geq 2$ and $c_i^j \mathbb{E}[T_i^j]$ increases in c_i^j, we have that,

$$c_i^j \mathbb{E}[T_i^j] \geq \frac{4\mu_i \alpha}{2\alpha - 1}, \tag{11}$$

where the minimum is attained at $c_i^j = 2$. Combine (6) and (11), λ_1 is upper bounded by,

$$\lambda_1 \leq \frac{M(2\alpha-1)}{4\alpha \overline{\mu \cdot m}}, \tag{12}$$

where $\overline{\mu \cdot m} = \frac{\sum_{i=1}^{N} \mu_i m_i}{N}$.

With the Pareto distribution in (10), when $\alpha > 2$, (9) can be reformulated as:

$$\frac{\lambda_t \frac{\overline{\mu^2}\alpha}{\alpha-1}}{1-2\lambda_t \frac{2\overline{\mu}\alpha}{2\alpha-1}} + \frac{2\overline{\mu}\alpha}{2\alpha-1} \leq \frac{\lambda_t \frac{\overline{\mu^2}\alpha}{\alpha-2}}{2(1-\lambda_t \frac{\overline{\mu}\alpha}{\alpha-1})} + \frac{\overline{\mu}\alpha}{\alpha-1}, \tag{13}$$

where $\overline{\mu} = \frac{\sum_{i=1}^{N} m_i \mu_i}{\sum_{i=1}^{N} m_i}$ and $\overline{\mu^2} = \frac{\sum_{i=1}^{N} m_i \mu_i^2}{\sum_{i=1}^{N} m_i}$.

Let $\overline{\mu^2} = k(\overline{\mu})^2$, we can further simplify (13) and express λ_t^* as follows:

$$\lambda_t^* = \frac{-a + \sqrt{a^2 + 16(2\alpha-1)(\alpha^2-3\alpha+2)(2k\alpha-4\alpha+8-k)}}{(8k\alpha-16\alpha+32-4k)\alpha\overline{\mu}},$$

where

$$a = \frac{k(\alpha^2-4\alpha+3)(2\alpha-1)^2}{\alpha} + 12\alpha^2 - 34\alpha + 20.$$

Combine λ_1 and λ_t^* together, λ^U is given by,

$$\lambda^U = \min\left[\frac{M(2\alpha-1)}{4\alpha\overline{\mu \cdot m}}, \frac{M\lambda_t^*}{\overline{m}}\right]. \tag{14}$$

On the one hand, if the parameter, α satisfies: $1 < \alpha < 2$, $\mathbb{E}[S^2]$ is not defined (i.e., goes to infinity) and so is W. Thus, cloning helps to reduce task latency under any λ_t when the system is not overloaded. In this case, λ^U is dominated by λ_1 in (12) only.

On the other hand, if α approaches ∞, (13) becomes:

$$\frac{\lambda_t \overline{\mu^2}}{1-2\lambda_t\overline{\mu}} + \overline{\mu} \leq \frac{\lambda_t \overline{\mu^2}}{2-2\lambda_t\overline{\mu}} + \overline{\mu}, \tag{15}$$

which holds if and only if $\lambda_t = 0$. In this case, cloning leads to a larger delay and is never efficient.

5 Optimal Cloning in the Lightly Loaded Regime

In a lightly loaded cluster, i.e., $\lambda < \lambda^U$, we will design a cloning algorithm by coordinating job scheduling with task cloning, which serves as an approximate solution to P1.

5.1 P2: A First Approximation for P1

As discussed at the beginning of Section 4, we assume that all copies of the same task are launched simultaneously in a cluster using task cloning. However, the tasks of the same

job can potentially be scheduled in different time slots. In that case, it is not easy to express the job flowtime in terms of the distribution function of tasks and thus makes P1 intractable.

Note that, when the cluster is lightly loaded, there is a large room for making clones for all unfinished tasks in the cluster most of the time. Thus, we solve another approximate problem, P2, under which all the tasks from the same job are scheduled together and cloned for the same number of times if there are enough servers. In this way, we can simplify the modeling of the job flowtime in P1 and only need to determine the number of copies scheduled for each task.

Consider a particular time slot l where $\chi(l) = \{J_{l_1}, J_{l_2}, \ldots, J_{l_K}\}$ contains all jobs which have not been scheduled yet by time l and $N(l)$ denotes the remaining number of idle machines. In addition, we consider the case that $\sum_i^K m_{l_i} < N(l)$, i.e., there is space to do task cloning. Let $\boldsymbol{c}_l = \{c_{l_i} | i = 1, 2, \ldots, K\}$ where c_{l_i} denotes the number of copies launched for a task in job J_{l_i}, then our objective is to maximize the total utility of $\chi(l)$ by choosing $\boldsymbol{c}_l$, which yields the optimization problem, P2, as shown below,

$$\begin{aligned}
\min_{\boldsymbol{c}_l} \ & \sum_{i=1}^{K} \mathbb{E}[\Gamma_{l_i}] + \gamma \cdot \sum_{i=1}^{K} \sum_{j=1}^{m_{l_i}} c_{l_i} \cdot \mathbb{E}[T_{l_i}^j] && \text{(P2)} \\
\text{s.t.} \ & \sum_{i}^{K} m_{l_i} \cdot c_{l_i} \le N(l), \\
& 1 \le c_{l_i} \le \xi, \quad \forall i, 1 \le j \le m_i, \\
& T_{l_i}^j = \min_{k=1,2,\ldots,c_{l_i}} T_{l_i}^{j,k}, \quad \forall 1 \le i \le K, 1 \le j \le m_{l_i}, \\
& \Gamma_{l_i} = \max_{j=1,2,\ldots,K} T_{l_i}^j + l - a_{l_i}, \quad \forall 1 \le i \le K.
\end{aligned}$$

We ignore the integer constraint of c_{l_i} in P2 for simplicity. Although it is difficult to interpret the physical meaning of $T_{l_i}^j$ when c_{l_i} is fractional, we will formally give the expression of the objective in terms of $\boldsymbol{c}_l$ for any positive valued c_{l_i} via replacing the last two constraints. As such, the feasible solution of P2 may not be integers.

5.2 Optimal Solutions to P2

Before deriving the optimal solutions to P2, we first show that P2 is a convex optimization problem.

Let $H_{l_i}^j(x)$ denote the CDF $T_{l_i}^j$, based on (4), $H_{l_i}^j(x)$ is given by:

$$H_{l_i}^j(x) = 1 - (1 - F_{l_i}(x))^{c_{l_i}}, \tag{16}$$

and the first moment of $T_{l_i}^j$, $\mathbb{E}[T_{l_i}^j]$, satisfies: $\mathbb{E}[T_{l_i}^j] = \int_0^\infty (1 - H_{l_i}^j(x))dx$, which implies that,

$$\sum_{j=1}^{m_{l_i}} c_{l_i} \cdot \mathbb{E}[T_{l_i}^j] = m_{l_i} c_{l_i} \int_0^\infty (1 - F_{l_i}(x))^{c_{l_i}} dx. \tag{17}$$

Let $D_{l_i} = \max\{T_{l_i}^1, T_{l_i}^2, \ldots, T_{l_i}^{m_i}\}$ and $\Omega_{l_i}(x)$ denote the CDF of D_{l_i}. Thus, $\Omega_{l_i}(x)$ is given by,

$$\begin{aligned}
\Omega_{l_i}(x) &= \Pr(D_{l_i} < x) = \prod_{j=1}^{m_{l_i}} \Pr(T_{l_i}^j < x) \\
&= \prod_{j=1}^{m_{l_i}} H_{l_i}^j(x) = (1 - (1 - F_{l_i}(x))^{c_{l_i}})^{m_{l_i}}.
\end{aligned} \tag{18}$$

Moreover, we have that, $\mathbb{E}[\Gamma_{l_i}] = \mathbb{E}[D_{l_i}] + l - a_i$ where $\mathbb{E}[D_{l_i}]$ is given by,

$$\mathbb{E}[D_{l_i}] = \int_0^\infty (1 - \Omega_{l_i}(x))dx. \tag{19}$$

Furthermore, the following lemma states that the computation cost of a job, i.e., $c_{l_i} \cdot \mathbb{E}[T_{l_i}^j]$, is a convex function of c_{l_i} under some mild conditions.

Lemma 1. *$c_{l_i} \cdot \int_0^\infty (1 - F_{l_i}(x))^{c_{l_i}} dx$ is a convex function of c_{l_i} provided that the hazard rate, a.k.a., $q_{l_i}(x) = \frac{dF_{l_i}(x)/dx}{1 - F_{l_i}(x)}$ is non-increasing in x.*

Similarly, the mean of the job flowtime, i.e., $\mathbb{E}[\Gamma_{l_i}]$ is also a convex function of c_{l_i} when $F_{l_i}(x)$ has a non-decreasing hazard rate in x. Readers may refer to [39] for the detailed proof of Lemma 1.

Remark 2. As shown in [1], most heavy-tailed distributions have a non-increasing hazard rate, e.g., the Pareto distribution has a hazard rate of $\frac{\alpha}{x}$.

Since the first two constraints in P2 are linear, we shall adopt convex optimization techniques to solve P2 and the global optimum can be attained. However, the widely-used Dual Approach (i.e., presented in Chapter 6 of [8]) usually suffers from a low convergence rate, which may incur a long delay for job scheduling. To design a fast scheduling algorithm, we adopt the ADMM method (Alternating Direction Methods of Multipliers) presented in [9] to solve P2.

5.2.1 ADMM Framework

Before illustrating the detailed solution approach, we first introduce the basics of the generalized L-block ADMM, which solves problems of the following form:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \sum_{i=1}^{L} f_i(x_i) \tag{20a}$$

$$s.t. \ A\boldsymbol{x} = \sum_{i=1}^{L} A_i x_i = b, \tag{20b}$$

$$x_i \in Z_i, \quad i = 1, 2, \ldots, L, \tag{20c}$$

where $x_i \in \mathbb{R}^{n_i}$ and $\boldsymbol{x} = (x_i | i = 1, 2, \ldots, L)$. In addition, $A_i \in \mathbb{R}^{p \times n_i}$, $b \in \mathbb{R}^p$ and $Z_i's$ are non-empty polyhedral sets. $f_i(x_i)'s$ are proper convex functions and thus the objective function, $f(\boldsymbol{x})$ is separable over L sets of variables.

The ADMM approach first introduces the augmented Lagrangian function given by:

$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda * (b - A\boldsymbol{x}) + \rho ||A\boldsymbol{x} - b||_2^2,$$

where $\lambda \in \mathbb{R}^{1\times p}$ is the Lagrangian multiplier and $\rho \geq 0$ is a constant.

A generalized ADMM algorithm then performs the following iterative update: at the rth iteration, the primal variable blocks, i.e., x_i's, are updated in the Gauss-Seidel fashion, and the dual multiplier, λ is updated using the newly-updated primal variables, i.e.,

$$x_i^{r+1} = \underset{x_i \in Z_i}{\operatorname{argmin}} L(x_1^{r+1}, \ldots, x_i, x_{i+1}^{r+1}, \ldots, \lambda^r), \tag{21a}$$

$$\lambda^{r+1} = \lambda^r + \eta(b - A\boldsymbol{x}^{r+1}), \tag{21b}$$

where $\eta > 0$ is a constant stepsize and x_i^{r+1} denotes the value of x_i at the rth iteration.

5.2.2 Solving P2 via Using ADMM

The formulation in P2 has a separable objective function but the linear constraint does not satisfy the equality condition. To fit into the ADMM framework, we introduce Variable $c_{l_{K+1}}$ such that $0 \leq c_{l_{K+1}} \leq N(l)$ to the first constraint in P1, which then becomes an equality, i.e., $\sum_{i=1}^{K+1} m_{l_i} \cdot c_{l_i} = N(l)$ where $m_{l_{K+1}} = 1$. Moreover, we let $f_{K+1}(c_{l_{K+1}}) = 0$ and

$$f_i(c_{l_i}) = \mathbb{E}[\Gamma_{l_i}] + \gamma \cdot \sum_{j=1}^{m_{l_i}} c_{l_i} \cdot \mathbb{E}[T_{l_i}^j]; 1 \leq i \leq K, \tag{22}$$

which are separable functions. And the corresponding polyhedral sets are as follows:

$$Z_{l_i} = \{x \in \mathbb{R} : 1 \leq x \leq \xi\}; 1 \leq i \leq K,$$

$$Z_{l_{K+1}} = \{x \in \mathbb{R} : 0 \leq x \leq N(l)\}.$$

After this transformation, we apply the ADMM method to solve P2. To be specific, we first introduce the following augmented Lagrangian function:

$$\Psi(\boldsymbol{c_l}, \lambda) = \sum_{i=1}^{K+1} f_i(c_{l_i}) + \lambda \cdot \left[N(l) - \sum_{i=1}^{K+1} m_{l_i} \cdot c_{l_i}\right] + \rho \left|\left|N(l) - \sum_{i=1}^{K+1} m_{l_i} \cdot c_{l_i}\right|\right|_2^2.$$

Then, at the rth iteration, we update the following two equations sequentially:

$$c_{l_i}^{r+1} = \underset{c_{l_i} \in Z_{l_i}}{\operatorname{argmin}} \Psi(c_{l_1}^{r+1}, \ldots, c_{l_i}, c_{l_{i+1}}^r, \ldots, \lambda^r), \tag{23a}$$

$$\lambda^{r+1} = \lambda^r + \eta\left[N(l) - \sum_{i}^{K+1} m_{l_i} \cdot c_{l_i}^{r+1}\right]. \tag{23b}$$

An argument similar to Lemma 1 shows that, the second derivative of $f_i(c_{l_i})$ with respect to c_{l_i}, is lower bounded by $\frac{-1}{\xi}\int_0^\infty \frac{q'_{l_i}(x)}{(q_{l_i}(x))^2}(1 - F_{l_i}(x))^r dx$ where $q'_{l_i}(x)$ is the first-order derivative of $q_{l_i}(x)$. As such, $f_i(c_{l_i})$ is a strongly convex function when provided that the hazard rate of $F_{l_i}(x)$, i.e., $q_{l_i}(x)$, strictly decreases in x. As such, we shall let ρ and η be the same and then choose a proper ρ by utilizing the strongly convex property of the function $f_i(c_{l_i})$ to achieve a sublinear convergence rate [24] for the iterates in (23).

5.2.3 Characterizing the Convergence Performance Using Pareto Distribution as One Example

In this part, we characterize the convergence performance of ADMM using the Pareto distribution in (10) as an illustrative example. Specifically, we provide a choice for ρ and η in the following theorem, to guarantee a fast convergence rate for the iterates in (23).

Theorem 1. *When $F_{l_i}(t)$ is given by (10) and choosing*

$$\rho = \eta \leq \min_{i=1,2,\ldots,K}\left\{\frac{4\mu_{l_i}\ln(1+m_{l_i})}{\alpha(K+1)^2\xi^3}\right\}, \tag{24}$$

the sequence of iterates $\{(\boldsymbol{c_l^r}, \lambda^r)\}_r$ generated by (23) converge sublinearly to an optimal primal-dual solution of P2. Moreover, it holds that $\Lambda^r \leq o(1/r)$, where Λ^r is defined as:

$$\Lambda^r = \left|\left|\sum_{i=1}^{K+1} m_{l_i} \cdot c_{l_i}^r - N(l)\right|\right|_2^2 + \sum_{i=1}^{K} ||m_{l_i}c_{l_i}^r - m_{l_i}c_{l_i}^{r-1}||_2^2.$$

Readers may refer to [39] for the detailed proof. Note that Theorem 1 guarantees a sublinear convergence rate of $o(1/r)$ for the iterates in (23), which is already quite efficient for solving the optimization problem P2 in practice.

However, we can further choose ρ and η sufficiently small to enjoy a faster convergence rate. Hong *et al.* have shown the linear convergence of ADMM under the assumption that the objective function satisfies certain error bound conditions [19]. Using the same technique as that in the proof of Theorem 1, we show that, for all $i \in \{1, 2, \ldots, K\}$, f_{l_i} is continuously differentiable with a uniform Lipschitz continuous gradient when $F_{l_i}(t)$ is given by (10), i.e., there exists some ν_{l_i} such that,

$$||\nabla f_{l_i}(y_1) - \nabla f_{l_i}(y_2)||_2 \leq \nu_{l_i}||y_1 - y_2||_2.$$

Hence, we conclude that f_{l_i} satisfies the assumptions in [19]. Then, Theorem 3.1 in [19] indicates that, if we select η bounded by some parameters associated with certain error bound conditions, the sequence of iterates, $\{(\boldsymbol{c_l^r}, \lambda^r)\}_r$ should converge linearly to an optimal primal-dual solution of P2. Unfortunately, the theorems in [19] do not provide any quantitative formula to determine the concrete threshold values for ρ and η in practice. Nevertheless, in our implementation for solving P2, we choose η such that it is one order of magnitude smaller than the parameters illustrated in (24), i.e., $\rho = \eta = \frac{1}{10}\min_{i=1,2,\ldots,K}\left\{\frac{4\mu_{l_i}\ln(1+m_{l_i})}{\alpha(K+1)^2\xi^3}\right\}$. According to our experience, such setting yields a convergence rate faster than o (1/r) in practice. In the following example, the ADMM approach converges in 20 iterations in this setting while it needs more than 70 iterations to converge in the setting using the stepsize, $\min_{i=1,2,\ldots,K}\left\{\frac{4\mu_{l_i}\ln(1+m_{l_i})}{\alpha(K+1)^2\xi^3}\right\}$.

Fig. 1 depicts the results of a Matlab-based simulation experiment on a PC with a 2.6 GHz i5 Intel Core to compare the convergence rate between the ADMM method and the standard Dual Approach. In this experiment, we consider

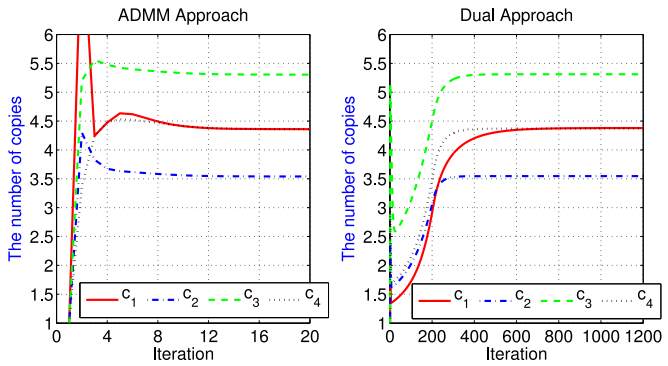


Fig. 1. The convergency performance of the ADMM method and the dual approach. c_i represents the number of copies for each task in Job J_i where $i \in \{1, 2, 3, 4\}$.

that, there are four jobs waiting to be scheduled in a particular time slot l, i.e., $\chi(l) = \{J_1, J_2, J_3, J_4\}$. The cluster has 200 available machines and the number of tasks for each job are 10, 20, 5 and 10 respectively. Moreover, we let $\mu_1 = 1, \mu_2 = 3, \mu_3 = 4, \mu_4 = 2$ and $\alpha = 2$. The number of copies for each task is bounded by $\xi = 20$. As shown in Fig. 1, the sequence of iterates converges in 20 iterations under the ADMM method while it needs more than 1,000 iterations to converge for the Dual Approach. Moreover, the running time of the Dual Approach and the ADMM method are 30 milliseconds and 80 milliseconds respectively.

5.3 The Design of the Smart Cloning Algorithm (SCA) in a Lightly-Loaded Cluster

After running P2, we shall take the integer part of the solutions such that the number of copies allocated to each task is integer valued. However, following the analysis in Section 5.1, there exists a case where there is no space for task cloning at the beginning of a particular time slot, i.e., $\sum_i^K m_{l_i} > N(l)$, though such a case seldom happens in a lightly-loaded cluster. In this scenario, it does not make sense to solve P2. Instead, we design an alternative scheduling scheme to allocate the servers to these jobs by adopting a smallest remaining workload first scheme, which is extended from the SRPT scheduler. Based on P2 and the extended SRPT scheduler, we propose the Smart Cloning Algorithm below.

SCA consists of two separate parts. At the beginning of each time slot, we first schedule the remaining tasks of unfinished jobs and then check whether the condition $\sum_i m_{l_i} < N(l)$ is satisfied. If the condition is satisfied, we solve P2 to determine the number of clones for each task. Otherwise, i.e., $\sum_i m_{l_i} \geq N(l)$, we sort $\chi(l)$, i.e., the set of unscheduled jobs, according to the increasing order of the workload in each J_{l_i} and schedule the jobs based on this order and only launch one copy for each task. Note that the resultant workload is the product of m_{l_i} and $\mathbb{E}[X_i^j]$. The corresponding pseudo-code is given in Algorithm 1.

Remark 3. When $F_i(x)$ has a strictly decreasing hazard rate, the ADMM method converges within $O(1/\epsilon)$ steps where ϵ is the satisfaction level. In each step, the ADMM method needs to scan all unscheduled jobs and iterates $N(l)$ times. Thus, we conclude that the time complexity of the ADMM approach is $O(N(l)/\epsilon)$. However, when $\sum_i^K m_{l_i} < N(l)$ is not satisfied, SCA only schedules all unfinished tasks and does not run ADMM. Thus, we conclude that the complexity of SCA is bounded by $O\big(\max[|\chi(l)| \cdot \max_{i=1,2,\ldots,N} m_i, O(N(l)/\epsilon)]\big)$.

Algorithm 1. Smart Cloning Algorithm

Input: The jobs in the cluster associated with their running status at time slot l;
Output: Scheduling decisions for time slot l.

```
schedule the unassigned tasks of the running jobs in the cluster with the fewest remaining first;
update N(l) and χ(l);
if N(l) == 0 then
    return;
if Σ_i^K m_{l_i} < N(l) then
    solve P2 using the ADMM method and assign duplicates of the tasks in J_{l_i} based on the solutions of P2;
else
    sort jobs in χ(l) based on the increasing order of the remaining workload;
    for Job J_{l_i} in χ(l) do
        assign only one copy for each task of J_{l_i} and update N(l);
        if N(l) == 0 then
            return;
return;
```

6 Design of a Straggler-Detection-Based Algorithm for the Heavily Loaded Regime

In a heavily loaded cluster, i.e., $\lambda \geq \lambda^U$, the cloning-based scheme is not efficient and there is no room to make a replica for all tasks. Therefore, we adopt the detection-based approach to design approximate solutions for P1.

As discussed in Section 7.1, the detection approach proposed by Mantri [7] is demonstrated to be more efficient than other schemes. In Mantri, the system estimates the remaining time to finish (i.e., t_{rem}), for each task and predicts the required service time of a relaunched copy of the task (i.e., t_{new}). And a speculative copy is launched when the probability, $\mathbb{P}(t_{rem} > t_{new}\frac{c+1}{c})$, is above δ where c copies of the task are currently running. However, one can see that this speculative execution approach suffers from several fundamental limitations. First, it launches one extra copy for the running task conservatively, namely, only if the total amount of computation resource consumed decreases. Second, the accuracy of the estimation for t_{new} has a heavy impact on the task completion time. For example, if the estimation for t_{new} is much larger than the true value, the straggler may lose the chance to have its duplicates made.

Due to the aforementioned limitations, we propose to modify Mantri's scheme to yield our *enhanced* detection approach. We avoid estimating the running time of a new duplicate, i.e., t_{new}, and schedule an extra copy for the straggler more aggressively. To be specific, in our scheme, the scheduler will assign $(c_i^j - 1)$ duplicates for task δ_i^j iff it is detected to be a straggler in some time slot, i.e., its estimated remaining running time, $t_i^{j,rem}$, satisfies: $t_i^{j,rem} \geq \sigma_i \cdot \mathbb{E}[X_i^j]$, where σ_i and c_i^j are two parameters to be optimized. We adopt the mean task service time, i.e., $\mathbb{E}[X_i^j]$ to avoid estimating t_{new}.

Remark 4. In our detection approach, we need to estimate the remaining time of a straggler, i.e., $t_i^{j,rem}$. There are many estimation schemes proposed in the literature, in particular, Chen et al. [13] propose a smart strategy to promptly estimate the remaining time of a task, which achieves high accuracy. Therefore, we do not consider the estimation error of $t_i^{j,rem}$ in this paper and assume that both our algorithm and Mantri estimate the remaining time of a task perfectly.

6.1 P3: A Second Approximation for P1

Under the detection-based speculation scheme, a duplicate is launched only if a straggler is detected. As such, the scheduling sequence, $(w_i^{j,2}, \ldots, w_i^{j,c_i^j})$ depend on $w_i^{j,1}$ and $T_i^{j,1}$. In this case, we choose to formulate a computation cost minimization problem, i.e., P3, which serves as an approximation to P1. The reason of minimizing the computation cost only is that if a task consumes few computation resource and completes quickly, its occupying resource can be released soon and utilized by subsequent tasks. Therefore, minimizing the computation cost will definitely lead to a reduction in the job flowtime.

Define the random variable R_i^j as the computation cost of task δ_i^j in the cluster. For convenience, denote by θ_i^j the event that $t_i^{j,rem} > \sigma_i \mathbb{E}[X_i^j]$. Then, minimizing the computation cost of a task yields the following optimization problem:

$$\min_{\sigma_i, c_i^j} \ \mathbb{E}\left[R_i^j \middle| \theta_i^j\right] \Pr(\theta_i^j) + \mathbb{E}\left[R_i^j \middle| \overline{\theta_i^j}\right] \Pr(\overline{\theta_i^j}) \qquad \text{(P3)}$$

$$\text{s.t.} \ \ \theta_i^j := t_i^{j,rem} > \sigma_i \cdot \mathbb{E}[X_i^j].$$

where $\overline{A}$ denotes the event that A does not happen.

6.2 Finding the Optimal Solutions to P3

Observe in P3 that, we only need to optimize the straggler detection condition, which involves determining σ_i and the number of duplicates, c_i^j. In the sequel, we will first find an optimal c_i^j under a fixed σ_i. After that, we will use the optimized c_i^j to determine σ_i.

6.2.1 Deriving the Optimal Number of Duplicates

In the detection scheme, the duplicates of a task are scheduled after it runs for a certain amount of time. Without loss of generality, we consider task δ_i^j, whose duplicates are made after it runs for t units of time. Thus, θ_i^j happens if $T_i^{j,1} \geq \sigma_i \cdot \mathbb{E}[X_i^j] + t$.

Let V_i^j denote the running time of the duplicates for task δ_i^j, thus, $V_i^j = \min\{T_i^{j,2}, T_i^{j,3}, \ldots, T_i^{j,c_i^j}\}$ if $c_i^j \geq 2$ and $V_i^j = \infty$ if $c_i^j = 1$. Let $D_i^j \triangleq \min[T_i^{j,1} - t, V_i^j]$, we have that,

$$\mathbb{E}[R_i^j] = \mathbb{E}[t + c_i^j D_i^j].$$

and it follows that,

$$\arg\min_{c_i^j} \mathbb{E}[R_i^j] = \arg\min_{c_i^j} \mathbb{E}[c_i^j D_i^j].$$

In addition, we have that, $c_i^j D_i^j = T_i^{j,1} - t$ when θ_i^j does not happen, which implies that, $\mathbb{E}\left[c_i^j D_i^j \middle| \overline{\theta_i^j}\right] = \mathbb{E}\left[T_i^{j,1} \middle| \overline{\theta_i^j}\right] - t$. Hence, the optimal value of c_i^j is given by:

$$(c_i^j)^* = \arg\min_{c_i^j} \mathbb{E}\left[c_i^j D_i^j \middle| \theta_i^j\right].$$

Our main result, characterizing the optimal solution of c_i^j, i.e., $(c_i^j)^*$, is given in the following theorem:

Theorem 2. *$(c_i^j)^*$ is at most two, provided that $F_i(x)$ has a non-increasing hazard rate and*

$$\int_0^\infty \bar{F}_i^2(x)dx \geq \frac{2}{3}\mathbb{E}[X_i^j],$$

where $\bar{F}_i(x) = 1 - F_i(x)$.

Readers may refer to [39] for the detailed proof of Theorem 2. In addition, when $\sigma_i > 2$ and $c_i^j = 2$, we have that $\mathbb{E}[c_i^j D_i^j] \leq c_i^j \mathbb{E}[V_i^j] = 2\mathbb{E}[X_i^j] < \sigma_i \mathbb{E}[X_i^j]$, which implies that $(c_i^j)^*$ is exactly two when the condition in Theorem 2 holds.

6.2.2 Finding the Optimal Detection Condition

In this section, we derive the optimal solution for σ_i in P3 via minimizing $\mathbb{E}[R_i^j]$ and setting c_i^j to two. However, the optimal solution depends on the time when a straggler is detected. To tackle this issue, we shall reformulate $\mathbb{E}[R_i^j]$ in P3.

Let π_i^j denote the event that $T_i^{j,1} > \sigma_i \mathbb{E}[X_i^j]$. By conditioning on π_i^j, $\mathbb{E}[R_i^j]$ is expressed as:

$$\mathbb{E}[R_i^j] = \mathbb{E}\left[R_i^j \middle| \pi_i^j\right] \Pr(\pi_i^j) + \mathbb{E}\left[R_i^j \middle| \overline{\pi_i^j}\right] \Pr(\overline{\pi_i^j}).$$

If the event π_i^j does not happen, no extra copy will be made for task δ_i^j, thus, we have that,

$$\mathbb{E}\left[R_i^j \middle| \overline{\pi_i^j}\right] \Pr(\overline{\pi_i^j}) = \int_0^{\sigma_i \mathbb{E}[X_i^j]} x d(F_i(x)),$$

which depends on σ_i only.

In the contrast, if Event π_i^j happens, there is a possibility that the first copy of δ_i^j is identified as a straggler. By conditioning on $T_i^{j,1}$, we have that,

$$\mathbb{E}\left[R_i^j \middle| \pi_i^j\right] \Pr(\pi_i^j) = \int_{\sigma_i \mathbb{E}[X_i^j]}^\infty \mathbb{E}\left[R_i^j \middle| T_i^{j,1} = t\right] d(F_i(t)). \qquad (25)$$

Whether an extra copy is made or not depends on when there are available machines. To characterize this moment, we give the following definition:

Definition 1. *The* asktime, *ϕ_i^j, of a running task δ_i^j, is the earliest time that the scheduler checks whether it should make an extra copy for task δ_i^j on available machines or not.*

We introduce $\xi_i^j(x)$ to characterize the cumulative distribution of ϕ_i^j, i.e., $\xi_i^j(x)$ is given by,

$$\xi_i^j(x) = \Pr(\phi_i^j \leq x), \quad \forall x \in [0, T_i^{j,1}].$$

Thus, it follows that,

$$\mathbb{E}\left[R_i^j \middle| T_i^{j,1} = t\right] = \int_0^t \mathbb{E}\left[R_i^j \middle| \phi_i^j = x, T_i^{j,1} = t\right] d\xi_i^j(x). \quad (26)$$

Substitute (26) into (25) yields:

$$\mathbb{E}\left[R_i^j \middle| \pi_i^j\right] \Pr(\pi_i^j) = \int_{\sigma_i \mathbb{E}[X_i^j]}^{\infty} dF_i(t) \int_0^t \mathbb{E}\left[R_i^j \middle| \phi_i^j = x, T_i^{j,1} = t\right] d\xi_i^j(x). \quad (27)$$

Based on the detection policy, an extra copy is made for δ_i^j iff $(T_i^{j,1} - \phi_i^j) > \sigma_i \mathbb{E}[X_i^j]$, i.e., $\phi_i^j < (T_i^{j,1} - \sigma_i \mathbb{E}[X_i^j])$. Therefore, when $\phi_i^j \geq (T_i^{j,1} - \sigma_i \mathbb{E}[X_i^j])$, no duplicate will be made and we have that,

$$\mathbb{E}\left[R_i^j \middle| \phi_i^j = x, T_i^{j,1} = t\right] = t. \quad (28)$$

By contrast, when $\phi_i^j < (T_i^{j,1} - \sigma_i \mathbb{E}[X_i^j])$, an extra copy is made and it follows that,

$$\mathbb{E}\left[R_i^j \middle| \phi_i^j = x, T_i^{j,1} = t\right] = x + 2\mathbb{E}\left[\min[t - x, T_i^{j,2}]\right], \quad (29)$$

where $\mathbb{E}\left[\min[t - x, T_i^{j,2}]\right]$ characterizes the remaining processing time for task δ_i^j when it has run for x units of time.

However, it is extremely difficult to characterize the exact distribution of the asktime as it depends on the full state of the whole system. Since the cluster is heavily loaded and all the scheduling intervals share the same length, we use the uniform distribution to approximate $\xi_i^j(x)$, i.e., the distribution function of ϕ_i^j. Thus, $\xi_i^j(x)$ is given by,

$$\xi_i^j(x) = \frac{x}{T_i^{j,1}}, \quad \forall x \in [0, T_i^{j,1}]. \quad (30)$$

Substitute (28), (29) and (30) into (27) yields:

$$\mathbb{E}\left[R_i^j \middle| \pi_i^j\right] \Pr(\pi_i^j) = \int_{\sigma_i \mathbb{E}[X_i^j]}^{\infty} dF_i(t) \int_0^{t - \sigma_i \mathbb{E}[X_i^j]} \frac{\left(x + 2\mathbb{E}\left[\min[t - x, T_i^{j,2}]\right]\right)}{t} dx + \int_{\sigma_i \mathbb{E}[X_i^j]}^{\infty} dF_i(t) \int_{t - \sigma_i \mathbb{E}[X_i^j]}^{t} dx.$$

Towards this end, $\mathbb{E}[R_i^j]$ is expressed by a function that depends only on σ_i. We then obtain the optimal value of σ_i by setting the derivative of $\mathbb{E}[R_i^j]$ to zero.

6.2.3 Pareto Distribution as One Illustrative Example

In this part, we use the Pareto distribution given in (10) as one example to illustrate the optimal solutions to P3. Thus, we have that,

$$\int_0^{\infty} \bar{F}_i^2(x) dx = \frac{2\mu_i \alpha}{2\alpha - 1},$$

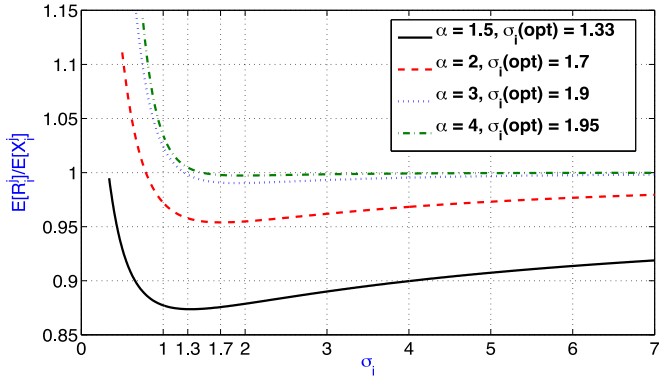


Fig. 2. The illustration of $E[R_i^j]/E[X_i^j]$ with different σ_i under Pareto Distribution for $\alpha = 1.5, 2, 3, 4$

which is above $\frac{2}{3}\mathbb{E}[X_i^j] = \frac{2\mu_i \alpha}{3(\alpha - 1)}$ when $\alpha \geq 2$. Therefore, the Pareto distribution satisfies the condition in Theorem 2 when $\alpha \geq 2$.

We depict the result of $\mathbb{E}[R_i^j]$ in Fig. 2 under different σ_i where $\sigma_i(opt)$ denotes the optimal value. Observe that when σ_i is close to 1.7 and α is 2, $\mathbb{E}[R_i^j]$ attains its minimum, which is less than $\mathbb{E}[X_i^j]$. Therefore, the optimal solution for c_i^j is exactly two when $\alpha \geq 2$. We also compare different optimal values for σ_i when the heavy-tail order changes. As shown in Fig. 2, $\sigma_i(opt)$ increases along α. Moreover, for all $\alpha \geq 3$, $\sigma_i(opt)$ is very close to 2.0.

6.3 Design Details of ESE

In this section, we design a detection algorithm called Enhanced Speculative Execution, which is based on the optimization results of P3. The corresponding pseudocode is given in Algorithm 2.

Algorithm 2. Enhanced Speculative Execution

Input: The jobs in the cluster associated with their running status at time slot l;
Output: Scheduling decisions for time slot l.

```
Count N(l), the number of idle machines at time slot l
and update D(l), R(l), χ(l).
for task δ_i^j in in D(l) do
    Assign a duplicate of δ_i^j on a random idle machine;
    N(l) -= 1;
    if N(l) == 0 then
        return;
for job J_i in R(l) do
    Assign the unscheduled tasks of J_i on idle machines;
    update N(l);
    if N(l) == 0 then
        return;
for job J_{l_i} in χ(l) do
    Assign one copy for each task in J_{l_i};
    update N(l);
    if N(l) == 0 then
        return;
return;
```

Our ESE Algorithm includes three scheduling levels. At the beginning of time slot l, the scheduler estimates the remaining time of each running task and puts the tasks whose remaining time satisfies the constraint of P3 in the

TABLE 2
Google Trace Data Statistics

Trace duration (seconds)	102,767
Average number of tasks per job	123.8
Minimum task duration (seconds)	13.5
Maximum task duration (seconds)	22,919.3
Average task duration (seconds)	1,246.7

backup candidate set, $D(l)$. Define $t_i^{j,rem}(l)$ as the remaining time of task δ_i^j at the beginning of time slot l. Then, $D(l) = \{\delta_i^j : c_i^j(l) = 1 \ \& \ t_i^{j,rem}(l) > \sigma_i \cdot \mathbb{E}[X_i^j]\}$ where σ_i is obtained via minimizing $\mathbb{E}[R_i^j]$ in (6.2.2). All tasks in $D(l)$ are sorted according to the decreasing order of $t_i^{j,rem}(l)$ and the scheduler launches an extra copy for each task in $D(l)$ based on this order.

The scheduler then schedules the remaining tasks of the jobs which have already been scheduled but have not left the cluster yet. Denote by $R(l)$ the set of unfinished jobs at time slot l and the jobs are sorted based on remaining workloads. Upon scheduling, the jobs which have smaller remaining workload are given the higher priorities.

The number of available machines, i.e., $N(l)$ is updated after the aforementioned scheduling and the scheduler proceeds to allocate machines to these unscheduled jobs. To be specific, let $\chi(l) = \{J_{l_1}, J_{l_2}, \ldots, J_{l_K}\}$ denote all the jobs that have not been scheduled yet where the jobs are sorted based on their non-decreasing order of workloads. The scheduler launches one copy for each task in J_{l_i} if there are available machines where $1 \leq i \leq K$.

Remark 5. At the beginning of each time slot, the ESE Algorithm needs to scan all unfinished tasks in the cluster. Therefore, the complexity of ESE is bounded by $O((|R(l)| + |\chi(l)|) \cdot \max_{i=1,2,\ldots,N} m_i)$.

7 Performance Evaluation

In this section, we evaluate the performance of SCA and ESE via extensive simulations driven by real-world cluster job traces from Google [30]. The traces contain the information of job submission and the completion time of Google services on a cluster with 11,000 servers. It also includes the number of tasks in each job as well as the duration of each task. From the traces, we extract the statistics of jobs during a 28-hour period as shown in Table 2. We also exclude the unfinished jobs as well as those which have specific constraints on machine attributes. All the experiments are conducted on a PC with a 2.6 GHz i5 Intel Core.

Simulation methodology. Based on the trace data, we estimate the task service time for each job, which is fitted into a Pareto distribution using the average task duration with some fixed Pareto-tail order. We adopt this estimation to run our optimization programs, which determine when to make redundant copies along with how many copies to launch. In addition, we set the service time of a redundant copy to be the same as that of a task which is randomly chosen from all the tasks in the same job.

Baseline. We use the scheduling strategy of Microsoft Mantri as the baseline as it has been shown to be better than straggler mitigation schemes in Hadoop [2], Dryad [20], MapReduce [16] and LATE [41] (Refer to Section 6.1 in [7]).

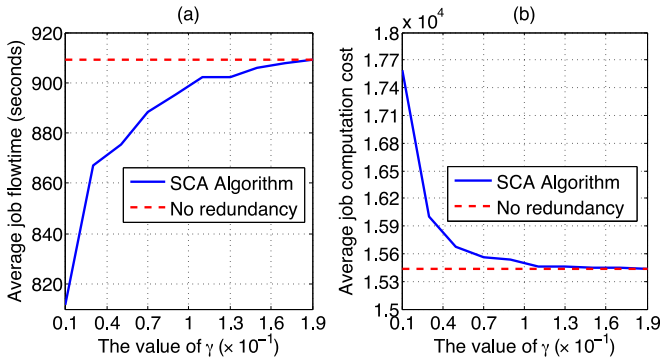


Fig. 3. The average job flowtime and job computation cost under the SCA algorithm with different γ.

Performance metric. We use the overall distribution of the job flowtime and the computation cost as the performance metrics for comparison. We also compare the average job flowtime and average job computation cost between different algorithms.

7.1 Performance Evaluation for SCA

In this section, we run the job traces on 11,000 machines and tune the parameters in P2, i.e., γ and ξ to evaluate the performance of SCA. We also investigate studies on the impact of scheduling intervals as well as the estimation accuracy.

7.1.1 Study the Impact of Different Parameters in SCA

As illustrated in Fig. 3, the average job flowtime increases in γ under SCA. When γ is below 0.1 and $\xi = 8$, the average job flowtime under SCA is much smaller than that under the PSRPT Algorithm (i.e., the smallest-remaining-workload-first-based scheduling strategy without task redundancy made). In particular, when γ is set to 0.01, the average job flowtime under SCA is 11 percent less than that under PSRPT. By contrast, when γ is above 0.1, the average job flowtime and computation cost under SCA are close to those under the PSRPT Algorithm. To yield a large reduction in the job flowtime for SCA when comparing to the Mantri baseline (as discussed in Section 7.1.5), we set γ to 0.01 in SCA.

Fig. 4 depicts the evaluation result of SCA with different ξ where $\gamma = 0.01$. The average job flowtime decreases in ξ.

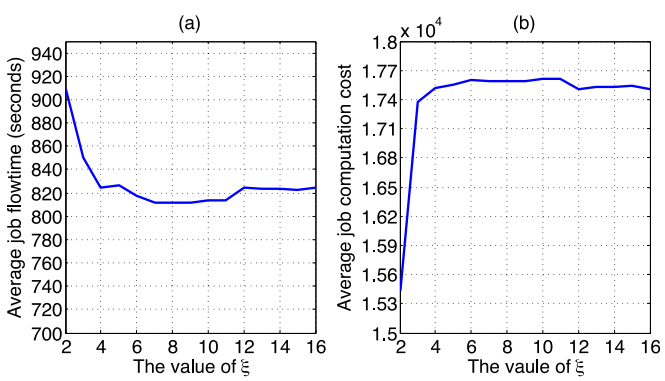


Fig. 4. The average job flowtime and job computation cost under the SCA algorithm with different ξ.

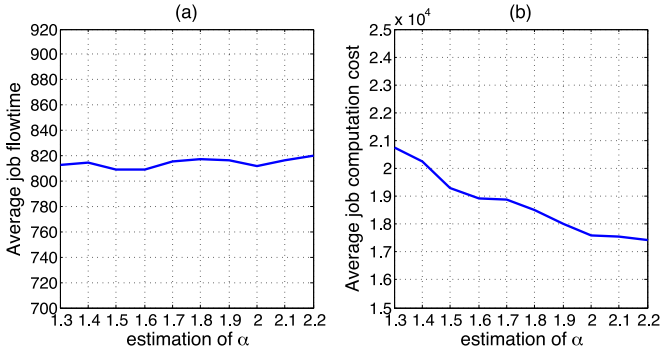


Fig. 5. The average job flowtime and job computation cost under the SCA algorithm with different estimated value of α.

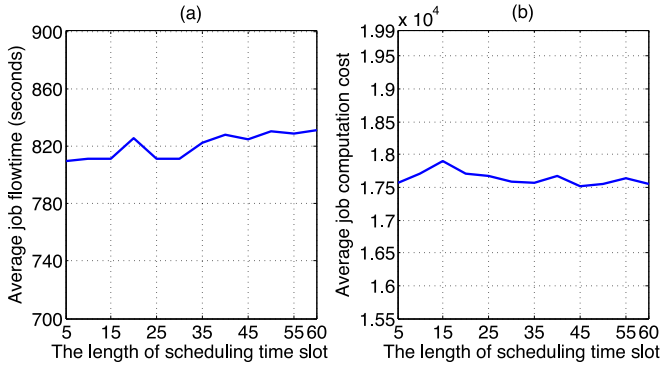


Fig. 6. The average job flowtime and job computation cost under SCA with different lengths of a scheduling interval.

However, when ξ is above three and further increases, the average job flowtime does not change much and converge to the same value, i.e., 810 seconds. Similarly, the average job computation cost converges to 1.75×10^4. In the following experiments, we set ξ to 8 in SCA.

7.1.2 The Impact of Estimation Accuracy

In this part, we characterize the impact of the estimation accuracy of task service time on job performance via tuning α, i.e, the heavy-tail order. We adopt different α to estimate the distribution of task service time and run the optimization program, P2 with $\gamma = 0.01$ and $\xi = 8$. As shown in Fig. 5, when α is between 1.3 and 2.2, the average job flowtime shares the same value, i.e., around 810 seconds. However, the average job computation cost decreases in α as a larger α results in a smaller number of duplicates scheduled. Therefore, we conclude that the job flowtime is not sensitive to the estimation accuracy of task service time and a larger estimated heavy-tail order yields a smaller job computation cost. In the sequel, we shall adopt $\alpha = 2$ for the estimation of task service time.

7.1.3 Characterizing the Length of Scheduling Intervals

In this part, we evaluate the cluster performance under SCA via tuning the length of a scheduling interval. As one may expect, on the one hand, with short scheduling intervals, SCA makes fast scheduling decisions and thus may reduce the job delay. On the other hand, short scheduling intervals usually cause large system overheads and make the scheduler not scalable.

We illustrate the evaluation result in Fig. 6, showing that both the average job flowtime and job computation cost are not sensitive to the length of scheduling intervals. As the job arrival rate is roughly $1/28$ job per second, we choose the length of each scheduling interval to be 30 seconds for SCA. Such a scheduling interval yields fast job response as well as small scheduling overheads since SCA runs a small-scale optimization problem which takes only 300 milliseconds on average at the beginning of each scheduling interval.

In addition, we also compare the computation time of SCA under different scheduling intervals, i.e., from 10 seconds to 60 seconds, to show how SCA scales with the number of tasks to be scheduled. We depict the comparison results in Table 3 where the second column represents the average number of tasks to be scheduled. As shown in Table 3, SCA is scalable since the computation time grows slower than the linear increase in the number of tasks.

7.1.4 Characterizing the Threshold for the Lightly and Heavily Loaded Regimes

We obtain the threshold to distinguish between the lightly loaded and heavily loaded regimes based on the analysis in Section 4. Rather than deriving the threshold for the job arrival rate directly, we derive the number of machines under which (14) is satisfied for the job traces, i.e., M is given by,

$$M = \max\left[\frac{4\alpha\lambda\overline{\mu \cdot m}}{2\alpha - 1}, \frac{\lambda\overline{m}}{\lambda_t^*}\right], \tag{31}$$

where λ_t^* is given in (14). For the given job trace illustrated in Table 2, we fist compute the mean of the total time spent by the system serving one job, which is 5,317 seconds. We then compute the theoretical threshold based on (31), which is around 6,000 machines. As a demonstration, we maintain the same job trace and let it run on different number of machines to capture the average job flowtime. Fig. 7 depicts the comparison result of the average job flowtime between SCA and PSRPT. It shows that, when the number of machines exceeds 6,000, SCA performs better in terms of the average job flowtime. Therefore, we conclude that our queuing model and approximation results in Section 4 are effective for deriving the cut-off threshold between different operating regimes.

7.1.5 Comparison between SCA and the Baseline

With the parameters set above, we proceed to compare the performance of SCA and the baseline in a cluster with 11K machines. In this cluster, the system load, i.e., the ratio between the arrival rate and the system processing capacity

TABLE 3
Computation Time of SCA Under Different Scheduling Intervals

scheduling interval (seconds)	number of tasks	computation time (milliseconds)
10	43	130
20	86	220
30	129	300
40	172	370
50	215	380
60	258	436

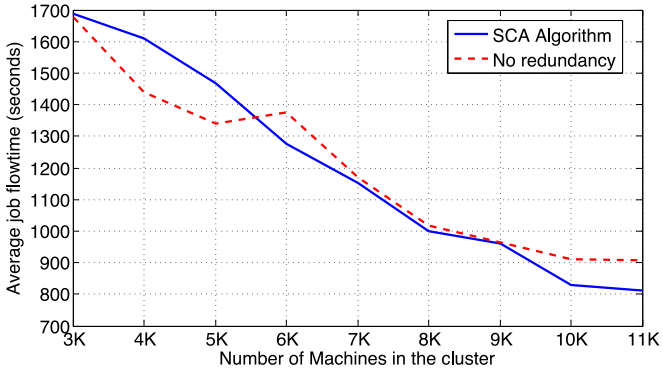


Fig. 7. The average job flowtime for the trace data running on different number of machines.

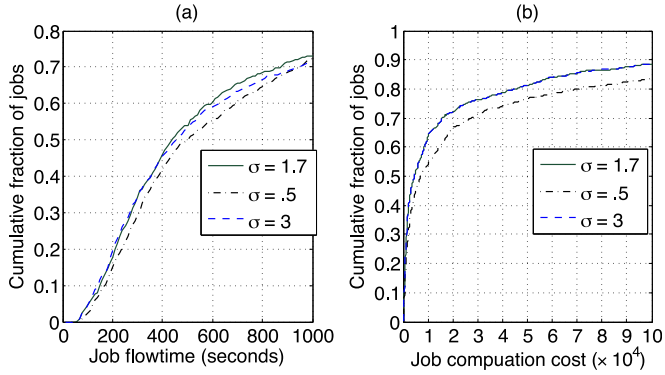


Fig. 9. Performance comparison between different σ_i under ESE in a heavily loaded cluster.

is $\frac{5317}{11000} = 0.48$. Our simulation results show that, the average job flowtime is 811 seconds under SCA while it is 860 seconds under Mantri, i.e., SCA reduces the job flowtime by nearly 6 percent. Fig. 8 depicts both the cumulative distribution function of job flowtime and computation cost. It is worth noting that, under SCA, more than 59 percent of jobs can finish within 300 seconds. By contrast, only about 55 percent of jobs can complete within 300 seconds under Mantri. In addition, Fig. 8b shows that, nearly 84 percent of jobs consume less than 10,000 units of computation cost under SCA. However, about 88 percent of jobs consume less than 10,000 units of computation cost under Mantri. The reason behind is that SCA launches duplicates in a greedy fashion and therefore needs to consume more resources for reducing the job flowtimes.

Comment 1. *Although many cluster schedulers reviewed in Section 2.1 have conducted formal analyses and optimized the job flowtime, they did not support speculative execution and assumed that task service times are deterministic numbers. Therefore, it is difficult to compare these works with ours.*

7.2 Performance Evaluation for ESE

In this section, we use the Google cluster traces to evaluate the performance of the ESE Algorithm in both the lightly-loaded and heavily-loaded regimes. We adopt $\alpha = 2$ for the estimation of task service time and set the length of each scheduling interval to 30 seconds.

7.2.1 Characterize the Impact of σ_i

To evaluate the impact of σ_i, we run several simulations under the optimal value and other constant values for σ_i to compare the average job flowtime and the computation cost in a heavily-loaded cluster with 5,000 servers, which corresponds to a cluster with system load that is close to one. The optimal value for σ_i is obtained according to the theoretical analysis in Section 6.2.2. As a representative case, we illustrate the comparison result for σ_i = 0.5 or 3. As shown in Fig. 9, when σ_i takes the optimum value, i.e., 1.7, both the job flowtime and the computation cost attains their minimum.

7.2.2 The Performance of ESE in the Heavily Loaded Regime

In this simulation, we evaluate the performance of ESE in the heavily-loaded regime by running the job traces on 5,000 machines. Since ESE consists of two parts, i.e., the SRPT-based task scheduling and the enhanced straggler detection approach designed based on the optimization results in Section 6.2, we evaluate these two parts in separate.

We first combine the same scheduling algorithm as Mantri and our optimized straggler detection approach as one scheme to run the job traces. The results show that the average job flowtime is 4,260 seconds under this scheme. By contrast, the average job flowtime under Mantri baseline is 4,640 seconds. Therefore, our optimized detection approach leads to a 9 percent reduction in this performance metric. We then run the job traces under ESE and the results show that the average job flowtime is 1,282 seconds. Thus, the SRPT-based scheduling algorithm leads to another 64 percent reduction in this performance metric.

To make a comprehensive comparison between ESE and Mantri, we also illustrate the CDF of both the job flowtime and computation cost. As shown in Fig. 10a, about

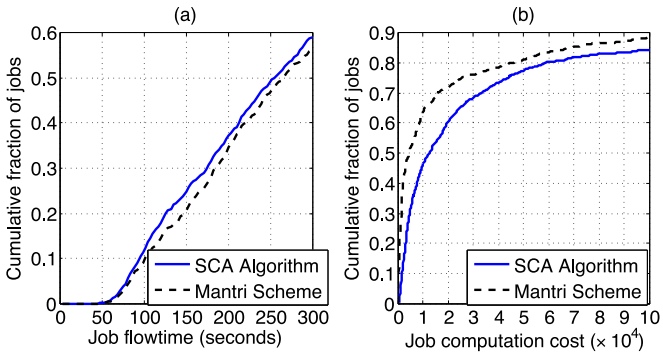


Fig. 8. The comparison between SCA and the baseline for the job flowtime.

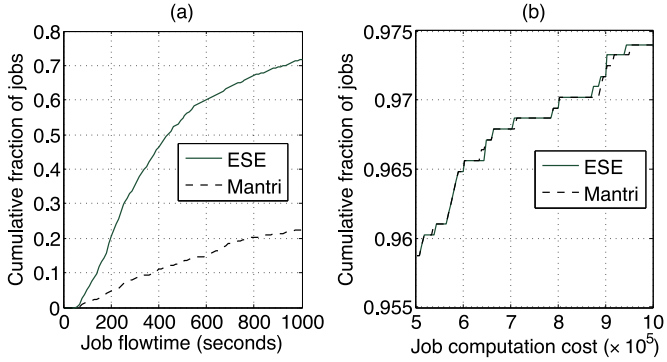


Fig. 10. Performance comparison between the ESE algorithm and the baseline in the heavily loaded regime.

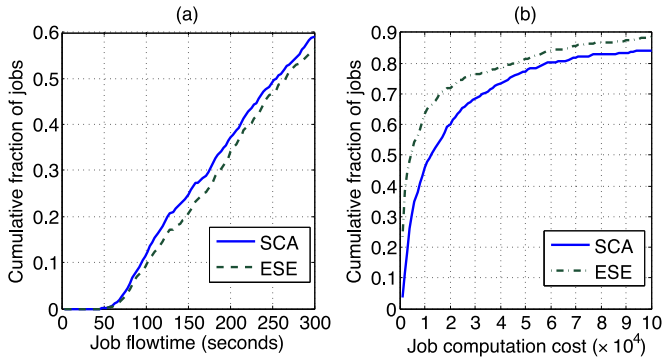


Fig. 11. Performance comparison between SCA and ESE in the lightly loaded regime.

73 percent of jobs complete within 1,000 seconds under ESE while only 22 percent of jobs complete within 1,000 seconds under Mantri. However, the average job computation cost under ESE is 1 percent less than that under Mantri. And the computation cost for small jobs are roughly the same under these two schemes. Thus, we only depict the CDF for big jobs in Fig. 10b, showing that the computation costs for big jobs under ESE are slightly smaller than those under Mantri.

Although ESE is optimized for straggler detection, only a small portion of tasks are identified to be stragglers, which have a very limited impact on the overall job computation costs. The same argument also holds for Mantri. To better demonstrate this point, we also run the same job traces on PSRPT (introduced in Section 7.1.1) without speculative execution. The results show that the average job computation cost under PSRPT is very close to that under both ESE and Mantri. Therefore, the detection approach does not contribute much to reducing the computation cost. Nevertheless, a single straggler can delay the whole job completion significantly and thus different detection approaches may vary substantially in terms of reducing the job flowtime. Due to these reasons, our proposed ESE outperforms Mantri baseline in terms of job flowtime while using nearly the same amount of computation costs.

7.2.3 Comparison between SCA and ESE in a Lightly-Loaded Cluster

As demonstrated in the aforementioned simulations, when the cluster has 5,000 machines, the average job flowtime under SCA is much larger than that under ESE (i.e., 1,500 seconds versus 1,282 seconds). In this part, we compare the performance of SCA and ESE in the lightly loaded regime by rerunning the job traces in a cluster with 11,000 servers.

Fig. 11 depicts the comparison result between SCA and ESE for both the job flowtime and the computation cost. It shows that 56 percent of jobs complete within 300 seconds under ESE while 59 percent of jobs complete within 300 seconds under SCA. Moreover, the average job flowtime under SCA is 3 percent less than that under ESE (811 seconds versus 837 seconds). However, the average computation cost under SCA is 14 percent higher than that under ESE.

Based on the evaluation results, one can see that, SCA is much more efficient for reducing the flowtime of small jobs, when comparing to ESE. The reason behind is that ESE launches a replica for a task after it runs for some time and may be too late for small tasks/jobs. Another advantage of SCA over ESE is that SCA does not need to monitor the task progress and estimate the remaining running time. To make a fair comparison between SCA and ESE, the system monitoring and estimation overhead should not be neglected in practice.

8 Discussions and Implementation Issues

In this section, we discuss how to implement SCA and ESE under the commonly-deployed big data resource management system, i.e., Hadoop YARN [36]. Under YARN, resource is allocated in terms of containers which specify the size of both CPU and Memory. As such, the resource constraint in P1 needs to be split into the memory constraint and the CPU constraint separately, i.e., the total CPU and Memory allocated at any time does not exceed their corresponding capacities. Nevertheless, the approximate solutions proposed in the two different operating regimes are still applicable.

Job initialization. Once a client submits a job to the cluster, the Resource Manager (RM) will allocate a container for its corresponding Application Master (AM) and spawn it on a node. After that, the AM will submit its resource requests in terms of containers along with the statistics of the service time (i.e., the distribution information) for each container.

Job execution. Except for allocating all the containers in the cluster, RM is also responsible for estimating the job arrival rate, which is based on the previous job arrivals in a fixed time interval. With the estimated job arrival rate, RM then determines the current operating regime, i.e., the lightly loaded or the heavily loaded case. If the current operating regime belongs to the lightly loaded one, RM allocates containers following the SCA Algorithm. Based on the received container leases from the Node Manager (NM), the AM determines the number of cloned copies for each task and sends the container launch context (CLC) for each copy to the corresponding NM. When the AM is notified the completion of a copy for a task from the RM, it requests the containers which are running other cloned copies for this task to be killed. In contrast, if the cluster is operating under the heavily-loaded regime, the RM adopts the ESE Algorithm to allocate containers to the active AMs. In this case, the AM will keep track of the progress for its running containers and estimate the remaining service time. When a running task is identified to be a straggler, the AM will request the RM to allocate another container for this task.

9 Conclusions

This work is an attempt to combine job scheduling and speculative execution for the design of redundancy algorithms in big data processing clusters. More importantly, we focus on two key performance metrics which are the average job flowtime and the overall system computation costs. By utilizing the distribution information of the task service time, we build an optimization framework to maximize the overall system utility. We then design two approximation algorithms to tackle this optimization problem, i.e., the SCA Algorithm and ESE Algorithm, corresponding to the cloning-based and detection-based approaches respectively. To differentiate the applicability of these two algorithms, we also categorize the cluster into the lightly loaded

and heavily loaded cases and derive the cutoff threshold for these two operating regimes.

REFERENCES

[1] B. C. Arnold, "Pareto Distributions Second Edition," CRC Press, 2015.

[2] Apache. 2013. [Online]. Available: http://hadoop.apache.org

[3] Capacity Scheduler. 2013. [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html

[4] Fair Scheduler. 2013. [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html

[5] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proc. 10th USENIX Conf. Netw. Syst. Design Implementation*, Apr. 2013, pp. 185–198.

[6] G. Ananthanarayanan, M. C.-C. Hung, X. Ren, and I. Stoica, "GRASS: Trimming stragglers in approximation analytics," in *Proc. 11th USENIX Conf. Netw. Syst. Design Implementation*, Apr. 2014, pp. 289–302.

[7] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoic, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in MapReduce clusters using mantri," in *Proc. 9th USENIX Conf. Operating Syst. Des. Implementation*, Oct. 2010, pp. 265–278.

[8] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA, USA: Athena Scientific, 1999.

[9] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 4, pp. 1–122, Jan. 2011.

[10] D. Breitgand, R. Cohen, A. Nahir, and D. Raz, "On cost-aware monitoring for self-adaptive load sharing," *IEEE J. Sel. Areas Commun.*, vol. 28, no. 1, pp. 70–83, Jan. 2010.

[11] 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 *Proc. IEEE Conf. Comput. Commun.*, Mar. 2011, pp. 3074–3082.

[12] F. Chen, M. Kodialam, and T. Lakshman, "Joint scheduling of processing and shuffle phases in MapReduce systems," in *Proc. IEEE Conf. Comput. Commun.*, Mar. 2012, pp. 1143–1151.

[13] Q. Chen, C. Liu, and Z. Xiao, "Improving MapReduce performance using smart speculative execution strategy," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 954–967, Apr. 2014.

[14] 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 *Proc. IEEE Conf. Comput. Commun.*, Apr. 2014, pp. 1042–1050.

[15] R. B. Cooper, *Introduction to Queuing Theory*. New York, NY, USA: Macmillan, 1972.

[16] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Symp. Operating Syst. Design Implementation*, Dec. 2004, pp. 107–113.

[17] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. Hyytia, "Reducing latency via redundant requests: Exact analysis," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Modeling Comput. Syst.*, Jun. 2015, pp. 347–360.

[18] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge, U.K.: Cambridge Univ. Press, 2013.

[19] M. Hong and Z.-Q. Luo. (2012). On the linear convergence of the alternating direction method of multipliers, http://arxiv.org/pdf/1208.3922v3.pdf

[20] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. 2nd ACM SIGOPS/EuroSys Eur. Conf. Comput. Syst.*, Mar. 2007, pp. 59–72.

[21] G. Joshi, E. Soljanin, and G. Wornell, "Efficient replication of queued tasks to reduce latency in cloud systems," in *Proc. 53rd Annu. Allerton Conf. Commun., Control, Comput.*, Oct. 2015, pp. 107–114.

[22] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker, "Complexity of machine scheduling problems," *Ann. Discr. Math.* vol. 1, pp. 343–362, 1977.

[23] M. Lin, L. Zhang, A. Wierman, and J. Tan, "Joint optimization of overlapping phases in MapReduce," in *Proc. IFIP Perform. Conf.*, Sep. 2013, pp. 720–735.

[24] T. Lin, S. Ma, and S. Zhang, "On the sublinear convergence rate of multi-block ADMM," *J. Oper. Res. Soc. China*, vol. 3, pp. 251–274, 2015.

[25] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlos, "On scheduling in map-reduce and flow-shops," in *Proc. 23rd Annu. ACM Symp. Parallelism Algorithms Architectures*, Jun. 2011, pp. 289–298.

[26] Z. Qiu and J. Perez, "Evaluating the effectiveness of replication for Tail-Tolerance," in *Proc. 15th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2015, pp. 443–452.

[27] Z. Qiu and J. F. Pérez, "Assessing the impact of concurrent replication with canceling in parallel jobs," in *Proc. IEEE 22nd Int. Symp. Modelling, Anal. Simul. Comput. Telecommun. Syst.*, Sep. 2014, pp. 31–40.

[28] Z. Qiu and J. F. Pérez, "Enhancing reliability and response times via replication in computing clusters," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2015, pp. 1355–1363.

[29] Z. Qiu, J. Prez, and P. Harrison, "Variability-aware request replication for latency curtailment," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2016, http://www.zhanqiu.co.uk/uploads/2/1/0/9/21098362/infocom2016_latencyreduction.pdf

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

[31] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu, "Hopper: Decentralized speculation-aware cluster scheduling at scale," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 379–392.

[32] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu, "Speculation-aware cluster scheduling," *ACM SIGMETRICS Performance Eval. Rev.*, vol. 43, pp. 42–44, 2015.

[33] N. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" in *Proc. 51st Annu. Allerton Conf. Commun., Control, Comput.*, Oct. 2013, pp. 731–738.

[34] X. Sun, C. He, and Y. Lu, "Esamr: An enhanced self-adaptive MapReduce scheduling algorithm," in *Proc. IEEE 18th Int. Conf. Parallel Distrib. Syst.*, Dec. 2012, pp. 148–155.

[35] J. Tan, X. Meng, and L. Zhang. "Delay tails in MapReduce scheduling," in *Proc. 12th ACM SIGMETRICS/PERFORMANCE Joint Int. Conf. Meas. Modeling Comput. Syst.*, Jun. 2012, pp. 5–16.

[36] V. K. Vavilapalli, A. C. Murthy, C. Douglas, and M. S. Agarwal, "Apache Hadoop YARN: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput.*, Oct. 2013, Art. no. 5.

[37] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol.*, 2013, pp. 283–294.

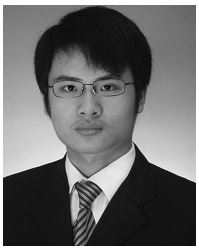
[38] H. Xu and W. C. Lau, "Task-cloning algorithms in a MapReduce cluster with competitive performance bounds," in *Proc. 35th IEEE Int. Conf. Distrib. Comput. Syst.*, Jun. 2015, pp. 339–348.

[39] H. Xu and W. C. Lau, "Optimization for speculative execution in big data processing clusters," 2016. [Online]. Available: http://home.ie.cuhk.edu.hk/~xh112/papers/Job_scheduling_technical_report.pdf

[40] Y. Yuan, D. Wang, and J. Liu, "Joint scheduling of MapReduce jobs with servers: Performance bounds and experiments," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2014, pp. 2175–2183.

[41] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. 8th Symp. Operating Syst. Des. Implementation*, Dec. 2008, pp. 29–42.

[42] Y. Zheng, N. B. Shroff, and P. Sinha, "A new analytical technique for designing provably efficient MapReduce schedulers," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2013, pp. 1600–1608.



Huanle Xu received the BSc(Eng) degree from the Department of Information Engineering, Shanghai Jiao Tong University in 2012. He is currently working toward the PhD degree in the Department of Information Engineering at The Chinese University of Hong Kong. His primary research interests include job scheduling and resource allocation in cloud computing, decentralized social networks, parallel graph algorithms, and machine learning. He is a student member of the IEEE.

Wing Cheong Lau received the BSc(Eng) degree from The University of Hong Kong and the MS and PhD degrees in electrical and computer engineering from the University of Texas at Austin. He is currently an associate professor in the Department of Information Engineering and the director of the Mobile Technologies Center at the Chinese University of Hong Kong. From 1997 to 2004, he was a member of the technical staff within the Performance Analysis Department at Bell Laboratories in Holmdel, New Jersey, where he conducted research in networking systems design and performance analysis. He joined Qualcomm, San Diego, California, in 2004 as a senior staff member conducting research on Mobility Management Protocols for the Next Generation Wireless Packet Data Networks. He also contributed actively to the standardization of such protocols in the Internet Engineering Task Force (IETF) and 3GPP2. He is/has been a technical program committee member of various international conferences, including ACM Sigmetrics, MobiHoc, IEEE Infocom, SECON, ICC, Globecom, WCNC, VTC, and ITC. He also served as the guest editor for the Special Issue on *High-Speed Network Security* of the *IEEE Journal of Selected Areas in Communications*. Wing holds 17 US patents with a few more pending. His research findings have been published in more than 90 scientific papers in leading international journals and conferences. His recent research interests include online social network privacy and vulnerabilities, resource allocation for big data processing systems, decentralized social networks, authenticated 2D barcodes and their Applications. He is a senior member of the IEEE and a member of ACM and Tau Beta Pi.