Optimization for Speculative Execution in a MapReduce-like Cluster

Huanle XU, Wing Cheong LAU
Department of Information Engineering, The Chinese University of Hong Kong
{xh112, wclau}@ie.cuhk.edu.hk

***Abstract*—A parallel processing job can be delayed substantially as long as one of its many tasks is being assigned to an unreliable 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 or simply because extra idling resource is available. In this paper, we focus on the design of speculative execution schemes for a parallel processing cluster under different loading conditions. For the lightly loaded case, we analyze and propose two optimization-based schemes, namely, the Smart Cloning Algorithm (SCA) which is based on maximizing the job utility. We also derive the workload threshold under which SCA should be used for speculative execution. Our simulation results show SCA can reduce the total job *flowtime* by nearly 22% comparing to the speculative execution strategy of Microsoft Mantri. For the heavily loaded case, we propose the Enhanced Speculative Execution (ESE) algorithm which is an extension of the Microsoft Mantri scheme. We show that the ESE algorithm can beat the Mantri baseline scheme by 35% in terms of job *flowtime* while consuming the same amount of resource.**

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

I. Introduction

Empirical performance studies of large-scale computing clusters have indicated that the completion time of a job [4] is often significantly and unnecessarily prolonged by one or a few so-called "straggler" (or outlier) tasks, i.e. tasks which are unfortunately assigned to either a failing or overloaded computing node within a cluster which consists of hundreds of thousands of commodity servers. As such, recent parallel processing frameworks such as the MapReduce system or its many variants have adopted various preventive or reactive straggler-handling strategies under which the system may automatically launch extra (backup) copies of a task on alternative machines in a judicious manner. Unfortunately, most of the existing speculative execution schemes are based on simple heuristics. In particular, there are two main classes of speculative execution strategies, namely, the Cloning [2], [3] approach and Straggler-Detection-based one [4], [8], [10], [11], [15], [20]. Under the Cloning approach, extra copies of a task are scheduled in parallel with the initial task as long as the resource consumption of the task is expected to be low and there is system resource available. For the Straggler-Detection-based approach, the progress of each task is monitored by the system and backup copies are launched when a straggler is detected. 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 lightly-loaded and heavily-loaded regimes but at the expense of extra system instrumentation and performance overhead. The situation is particularly challenging when the progress of a large number of tasks have to be tracked.

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 metrics which are the total job *flowtime* and resource consumption. In particular, we have made the following technical contributions:

- After reviewing related work in Section II, we introduce the system model in Section III and derive the cut-off workload threshold between the lightly-loaded and heavily-loaded operating regimes of a computing cluster. Based on this workload threshold, the applicability of a speculative execution strategy can be analyzed for different operating regimes.
- In Section IV, we introduce a generalized cloning-based framework to optimize the job utility which is a combination of job *flowtime* and resource consumption for the lightly loaded cluster. We also present a specific Smart Cloning Algorithm (SCA) based on this framework. We run several trace-driven simulations to compare the performance of SCA with Microsoft Mantri's scheme and show that the SCA can reduce the average job completion time in the lightly loaded cluster.
- In Section V, we propose the Enhanced Speculative Execution (ESE) algorithm for a heavily loaded cluster by extending the speculative execution strategy of Microsoft Mantri [4]. We demonstrate that ESE can improve the job completion time of Mantri while consuming the same amount of resource. We also summarize our findings and conclude the paper in Section VI.

II. Related work

Several speculative execution strategies have been proposed in the literature for the MapReduce system and its variants or derivatives. The initial Google MapReduce system only begins to launch backup tasks when a job is close to completion. It has been shown that speculative execution can decrease the job execution time by 44% [10]. This approach is easy to implement, however, the task whose progress is normal may

also be chosen for making extra duplicates and thus makes the speculative execution a waste of resource.

The speculative execution strategies in the initial versions of Hadoop [1] and Microsoft Dryad [11] closely follow that of the Google MapReduce system. However, the research group from Berkeley presents a new strategy called LATE (*Longest Approximate Time to End*) [20] in the Hadoop-0.21 implementation. It monitors the progress rate of each task and estimates their remaining time. Tasks with progress rate below certain threshold (*slowTaskTherehold*) 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*speculativeCap*. Microsoft Mantri [4] proposes a new speculative execution strategy for Dryad in which the system estimates the remaining time to finish, t_{rem}, for each task and predicts the required execution time of a relaunched copy of the task, t_{new}. Once a computing node 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}. Specifically, a duplicate is scheduled if $\mathbb{P}(t_{rem} > 2 * t_{new}) > \delta$ is satisfied where the default value of $\delta = .25$. Hence, Mantri schedules a duplicate only if the total resource consumption is expected to decrease while it does not explore the tradeoff between the completion time (*flowtime*) and resource consumption.

To accurately and promptly identify stragglers, [8] proposes a Smart Speculative Execution strategy and [15] presents an Enhanced Self-Adaptive MapReduce Scheduling Algorithm respectively. The main ideas of [8] include: i) use exponentially weighted moving average to predict process speed and compute the remaining time of a task and ii) determine which task to backup based on the load of a cluster using a cost-benefit model. The limitation is that those work only focus on the optimization of task level rather than the job level.

[2] proposes to mitigate the straggler problem by cloning every small job and avoid the extra delay caused by the straggler monitoring/ detection process. When the majority of the jobs in the system are small, the cloned copies only consume a small amount of additional resource. As an extension from [2], it presents two different strategies for Deadline-bound and Error-bound Jobs in [3]. These cloning approaches are demonstrated to be efficient for some specific job types rather than arbitrary workload.

III. System Model

Assume a set of jobs $J = \{J_1, J_2, \cdots\}$ arriving at a computing cluster at a rate of λ jobs per unit time. Different jobs may run different applications and a particular job J_i which arrives at the cluster at time a_i consists of m_i tasks. This cluster has M computing nodes (machine) and each computing node can only hold one task at any time. For simplicity, we assume this cluster is homogeneous in the sense that all the nodes are identical. Further, we assume that the execution time (i.e. the time between the task is launched and the task is finished) of each task of J_i without any speculative execution follows the same distribution, i.e., $x_j^i \sim F_i(t) = Pr(x_j^i < t)$ for $1 \leq j \leq m_i$. We assume the execution time distribution information can be estimated for each job according to prior trace data of the application it runs and the size of the input data to be processed. Upon arrival, each job joins a queue in the master-node of the cluster, waiting to be scheduled for execution according to some priorities to be determined in the following sections.

Here, we define the job *flowtime* which is an important metric we capture as below.

Definition 1. *The flowtime of a job J is $flow(J_i) = f(J_i) - a(J_i)$, where $f(J_i)$ and $a(J_i)$ denote the finish (completion) time and arrive time respectively.*

If a task j runs t_j units of time on a computing node, then it consumes t_j units of resource on this node where. We ignore the resource consumption of an idle machine. For ease of description, we often interchange the two notations in this paper, namely, machine and computing node.

A. Speculative execution under different operating regimes

The cloning-based strategy for speculative execution schedules extra copies of a task in parallel with the initial task as long as the resource consumption of the task is expected to be low and there is system resource available. Only the result of one which finishes first among all the copies is used for the subsequent computation. Cloning does not incur any monitoring overhead. Nevertheless, cloning consumes a large amount of resource and can easily block the scheduling of subsequent jobs when the cluster workload is heavy.

On the other hand, the Straggler-Detection-based approach makes a speculative copy for the task after a straggler is detected. In this approach, the scheduler needs to monitor the progress for each task. However, the monitoring incurs extra system instrumentation and performance overhead as discussed in [5]. The situation is particularly challenging when the progress of a large number of tasks have to be tracked. To make things even worse, it's always difficult to detect a straggler for small jobs as they usually complete their work in a very short period [2].

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 lightly-loaded and heavily-loaded regimes. Hence, there exists a cutoff threshold to separate the cluster workload into these two operating regimes. When the workload is below this threshold, the cloning-based strategy can obtain a good performance in terms of job *flowtime*. Conversely, when the workload exceeds this threshold, only the Straggler-Detection-based strategy can help to improve the cluster performance.

B. Deriving the cutoff threshold for different regimes

In this subsection, we derive the cutoff workload threshold λ^U which allows us to separate our subsequent analysis into the lightly loaded vs. heavily loaded regimes. We assume that the random variables m_i and x_j^i are independent from each

other for all i and j. To simplify the analysis, we focus on the task delay in the cluster instead of job delay. Denote by λ_t the task arrival rate to the whole cluster. Thus, $\lambda_t = \mathbb{E}[m_i] \cdot \lambda$. We approximate the task arrival as a Poisson process with rate λ_t. Further, denote by λ_t^m the average task arrival rate to each single machine which is given by $\lambda_t^m = \frac{\lambda_t}{M}$. We model the task service process of each computing node (machine) as a $M/G/1$ queue. Applying the result of [9], we get the average delay of each task without speculative execution made in the following equation

$$W_t = \frac{\lambda_t^m \mathbb{E}[s^2]}{2(1-\lambda_t^m \mathbb{E}[s])} + \mathbb{E}[s] \tag{1}$$

where $\mathbb{E}[s]$ and $\mathbb{E}[s^2]$ are the first order and second order for the duration of all the tasks in the system without speculative execution implemented.

We proceed to derive the expression of the task delay for the cloning-based strategy. Different from [2] where the cloning is done for small jobs only, here we consider a more general cloning scheme in which the small jobs are not distinguished from the big ones. In this scheme, each task should at least make two copies. Otherwise, the task which does not have any extra copy may delay the completion of the entire job.

We illustrate an example in which $F_i(t)$ (for all i) follows the Pareto Distribution as follows:

$$F_i(t) = \begin{cases} 1-(\frac{\mu}{t})^{\alpha} & for\ t \geq \mu \\ 0 & otherwise \end{cases}$$

Assume r (≥ 2) copies are launched for a particular task v. Then the expected duration for task v is $\mathbb{E}[s^{'}] = \frac{\mu r \alpha}{r\alpha - 1}$. Thus, $\frac{\mathbb{E}[s^{'}]}{\mathbb{E}[s]} = \frac{\alpha-1}{\alpha-1/r} > \frac{\alpha-1}{\alpha}$. This also gives a lower-bound of the performance improvement of cloning regardless of the number of extra copies to be made for each task.

Denote by $\mathbb{E}[r]$ the average number of copies each task makes. Hence, $\mathbb{E}[r] \geq 2$. Further define $\mathbb{E}[s^c]$ and λ_t^c as the average task duration and equivalent task arrival rate to each machine respectively after the cloning is made.

The first constraint for cloning is that it must not overload the system, i.e. the long-term system utilization of the cluster should be less than 1. Thus, the following inequality holds:

$$\lambda_t^c \cdot \mathbb{E}[s^c] < 1 \tag{2}$$

By considering the constraint in Eq.(2), and the fact that $\lambda_t^c = \mathbb{E}[r] \cdot \lambda_t^m$, we have:

Theorem 1. *The condition $\lambda \cdot \mathbb{E}[m_i]\mathbb{E}[s] \cdot \frac{4(\alpha-1)}{2\alpha-1} < M$ is necessary to guarantee that the cloning does not overload the system.*

Proof. Denote by Ω the sample space of the discrete random variable r and $P(a)$ ($a \in \Omega$) by the probability that r takes the value a. Thus,

$$E[r] = \sum_{a\in\Omega} aP(a) \tag{3}$$

$$E[s^c] = \sum_{a\in\Omega} \frac{\mu a \alpha}{a\alpha - 1} P(a) \tag{4}$$

Hence, $E[r]E[s^c] = \sum_{a\in\Omega} aP(a) \cdot \sum_{a\in\Omega} \frac{\mu a\alpha}{a\alpha-1} P(a)$. Applying the Koshy-Schwarz inequality here,

$$\begin{aligned} \sum_{a\in\Omega} aP(a) \sum_{a\in\Omega} \frac{\mu a\alpha}{a\alpha-1} P(a) &\geq \left(\sum_{a\in\Omega} \sqrt{aP(a)} \sqrt{\frac{\mu a\alpha}{a\alpha-1} P(a)} \right)^2 \\ &= \left(\sum_{a\in\Omega} \sqrt{\frac{\mu a^2 \alpha}{a\alpha-1}} P(a) \right)^2 \end{aligned}$$

Further, it holds that

$$\left(\sum_{a\in\Omega} \sqrt{\frac{\mu a^2\alpha}{a\alpha-1}} P(a) \right)^2 \geq \frac{4\mu\alpha}{2\alpha-1} \left(\sum_{a\in\Omega} P(a) \right)^2 = \frac{4\mu\alpha}{2\alpha-1} \tag{5}$$

Then, $\lambda_t^c \cdot E[s^c] \geq \frac{4\mu\alpha}{2\alpha-1} \cdot \lambda_t^m$. Substitute Equation (2) into this inequality and the result immediately follows. Thus, it completes the proof. □

However, the efficiency of cloning is not guaranteed by Theorem 1. 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. Denote by W_t^c the average task delay when each task has two copies. For convenience, we define $\omega = \frac{\lambda \cdot \mathbb{E}[m_i]\mathbb{E}[s]}{M}$. Thus,

$$W_t^c = \mathbb{E}[s] \cdot \frac{\omega \cdot \frac{(\alpha-1)(1-4\alpha^2+4\alpha)}{\alpha(2\alpha-1)} + 2(\alpha-1)}{2\alpha - 1 - 4\omega(\alpha-1)} \tag{6}$$

and

$$W_t^c < W_t \tag{7}$$

Combine (1), (6) and (7), we can derive the upper bound ω^U for ω. Hence, the cutoff threshold is determined by the following equation:

$$\lambda^U = \frac{\omega^U M}{\mathbb{E}[m_i]\mathbb{E}[s]} \tag{8}$$

In the following sections, we continue to introduce the cloning-based strategy and Straggler-Detection-based approaches under two different workload regimes.

IV. Optimal Cloning in the lightly loaded regime

In the lightly loaded cluster, i.e., $\lambda < \lambda^U$, we first apply the generalized cloning-based scheme to improve the job performance.

We consider that time is slotted and the scheduling decisions are made at the beginning of each time slot. Assume $J_i \in J$ consisting of m_i tasks which are from the set $\Phi_i = \{\delta_1^i, \delta_2^i, \cdots, \delta_{m_i}^i\}$ and δ_j^i is scheduled at time slot w_j^i. Denote by w_i the scheduling time of job i. Hence, w_j^i and w_i satisfy the following constraints:

$$w_j^i \in \{0, 1, 2\cdots,\} \ \ and \ \ w_j^i \geq a_i \quad \forall i;\ j \tag{9}$$

$$w_i = \min\{w_1^i, w_2^i, \cdots, w_{m_i}^i\} \quad \forall i \tag{10}$$

Each task of Φ_i can maintain different number of duplicates as the tasks in the same job may be scheduled at different time slots depending on server availability. Denote by c_j^i the number

of copies made for task δ_j^i. Further let $t_{j,k}^i$ define the duration of the kth clone for task δ_j^i. We assume $t_{j,k}^i$ follows the same distribution as x_j^i and all the $t_{j,k}^i$ are i.i.d random variables for $1 \leq k \leq c_j^i$. Define t_j^i as the duration of task δ_j^i and t_i as the *flowtime* of job J_i respectively. Then the following two equations hold:

$$t_j^i = \min\{t_{j,1}^i, t_{j,2}^i, \cdots, t_{j,c_i}^i\} \qquad \forall i; 1 \leq j \leq m_i \tag{11}$$

$$t_i = \max\{t_1^i + w_1^i, t_2^i + w_2^i, \cdots, t_{m_i}^i + w_{m_i}^i\} - a_i \qquad \forall i \tag{12}$$

Equation (11) states that as soon as one copy of task δ_j^i finishes, the task completes. Equation (12) describes the job *flowtime*.

We define the utility for each job which is a function of job *flowtime* and resource consumption. The formulation (P1) is as follows:

$$\begin{aligned} \max_{c_j^i, w_j^i} \quad & \sum_{i=1} -\mathbb{E}[t_i] - \gamma \cdot \sum_{i=1} \sum_{j=1}^{m_i} c_j^i \cdot \mathbb{E}[t_j^i] \\ s.t. \quad & \sum_{w_i \leq l} \sum_{w_j^i + t_j^i > l} c_j^i \leq M \quad \forall l \\ & 1 \leq c_j^i \leq r \quad \forall i; 1 \leq j \leq m_i \\ & (9), (11), (12) \end{aligned}$$

In this formulation, $U_i = -\mathbb{E}[t_i] - \gamma \cdot \sum_{i=1} \sum_{j=1}^{m_i} c_j^i \cdot \mathbb{E}[t_j^i]$ is the utility of job J_i and γ is a constant number. Our objective is to maximize the total utility of all the jobs in the cluster. The first constraint states that the total number of tasks including all task copies at any time slot is no more than M and the second constraint states that each individual task can at most maintain r copies in the cluster.

Remark 1. *The utility defined in U_i is a combination of the job flowtime and resource consumption. User can tune the parameter γ to be very small if he (she) is more concerned about flowtime than resource consumption or vice versa.*

A. Solving P1 through approximation

P1 is an online stochastic optimization problem and the scheduling decisions should be made without knowing the information of future jobs. In Equation (12), the tasks of the same job can be scheduled in different time slots. Hence, it is not easy to express the job *flowtime* in terms of the distribution function and thus makes P1 difficult to solve.

Due to the fact that the cluster is lightly loaded, there is a large room for making clones for all the jobs most of the time. Thus, we solve another optimization problem (P2) as a relaxation for P1 when system resource is available. In P2, all the tasks of the same job are scheduled together and maintain the same number of copies. In this way, we can simplify the modeling of the job *flowtime*.

Define $\chi(l)$ as the job set which contains all unscheduled jobs at time slot l. Assume $\chi(l) = \{J_{l_1}, J_{l_2}, \cdots\}$. If there is enough idling servers to schedule the jobs in the cluster at the beginning of time slot l, i.e., $\sum_i m_{l_i} < N(l)$ where $N(l)$ is number of available machines, we solve P2 to determine the number of copies for each task in J_{l_i} as below:

$$\begin{aligned} \max_{c_{l_i}} \quad & \sum_{i=1} -\mathbb{E}[t_{l_i}] - \gamma \cdot \sum_{i=1} \sum_{j=1}^{m_{l_i}} c_{l_i} \cdot \mathbb{E}[t_j^{l_i}] \\ s.t. \quad & \sum_i m_{l_i} \cdot c_{l_i} \leq N(l) \\ & 1 \leq c_{l_i} \leq r \quad \forall i \\ & t_j^{l_i} = \min\{t_{j,1}^{l_i}, t_{j,2}^{l_i}, \cdots, t_{j,c_{l_i}}^{l_i}\} \qquad \forall i; 1 \leq j \leq m_{l_i} \\ & t_{l_i} = \max\{t_1^{l_i}, t_2^{l_i}, \cdots, t_{m_i}^{l_i}\} + l - a_{l_i} \qquad \forall i \end{aligned}$$

Here, c_{l_i} is the number of duplicates assigned to each task in job J_{l_i}. Solving P2 is much easier and it only depends on the current information. Define $H_j^{l_i}(t)$ as the cumulative distribution function of $t_j^{l_i}$ and we have:

$$H_j^{l_i}(t) = 1 - (1 - F_{l_i}(t))^{c_{l_i}} \tag{13}$$

Let $d_{l_i} \triangleq \max\{t_1^{l_i}, t_2^{l_i}, \cdots, t_{m_i}^{l_i}\}$ and the distribution function of d_{l_i} is given by:

$$W_{l_i}(t) = Pr(d_{l_i} < t) = \prod_{j=1}^{m_{l_i}} H_j^{l_i}(t) = (1 - (1 - F_{l_i}(t))^{c_{l_i}})^{m_{l_i}} \tag{14}$$

Further, $\mathbb{E}[t_{l_i}] = \mathbb{E}[d_{l_i}] + l - a_i$ and we get

$$\mathbb{E}[d_{l_i}] = \int_0^\infty t \cdot d(W_{l_i}(t)) = \int_0^\infty (1 - W_{l_i}(t)) dt. \tag{15}$$

Similarly, $\mathbb{E}[t_j^{l_i}] = \int_0^\infty (1 - H_j^{l_i}(t)) dt$, which yields:

$$\sum_{j=1}^{m_{l_i}} c_{l_i} \cdot \mathbb{E}[t_j^{l_i}] = m_{l_i} c_{l_i} \int_0^\infty (1 - F_i(t))^{c_{l_i}} dt \tag{16}$$

Lemma 1. *$c_{l_i} \int_0^\infty (1 - F_i(t))^{c_{l_i}} dt$ is a convex function of c_{l_i} when provided $\ln(1 - F_i(t))$ is a convex function of t.*

Due to space limited, we omit the proof here. In the same way, $\mathbb{E}[t_{l_i}]$ is also a convex function of c_{l_i} which decreases as c_{l_i} increases.

Observe that the first two constraints in P2 are linear. Hence, we can adopt the convex optimization technique to solve P2. The Lagrangian Dual problem of P2 is given by:

$$\min_{\nu, \xi_{l_i}, h_{l_i}} D(\nu, \xi_{l_i}, h_{l_i}) \tag{17}$$

$$D(\nu, \xi_{l_i}, h_{l_i}) = \max_{c_{l_i}} f(\nu, \xi_{l_i}, h_{l_i}, c_{l_i}) \tag{18}$$

$$\begin{aligned} f(\nu, \xi_{l_i}, h_{l_i}, c_{l_i}) = & -\sum_{i=1} \mathbb{E}[t_{l_i}] - \gamma \cdot \sum_{i=1} m_{l_i} c_{l_i} \cdot \mathbb{E}[t_j^{l_i}] \\ & - \nu(\sum_{i=1} m_{l_i} c_{l_i} - N(l)) - \sum_i \xi_{l_i}(c_{l_i} - r) \\ & - \sum_{i=1} h_{l_i} \cdot (1 - c_{l_i}) \end{aligned} \tag{19}$$

where ν, ξ_{l_i}, h_{l_i} are nonnegative multipliers. Applying the result of convex optimization, we conclude that there is no duality gap between P2 and D.

We adopt the gradient projection approach to get the optimal solution of D. Define the vector $c(l) = (c_{l_1}, c_{l_2}, \cdots)$ and the algorithm is outlined below:

- Initialize $c_{l_i}^0 = 1$ for all i, $\nu^0 = 0.1$, $\xi_{l_i}^0 = 0.1$, $h_{l_i}^0 = 0.1$.
- $c_{l_i}^{k+1} = \arg\max_{c_{l_i}} f(\nu^k, \xi_{l_i}^k, h_{l_i}^k, c_{l_i})$;
- $v^{k+1} = v^k + \eta_1[\sum_{i=1} m_{l_i} c_{l_i}^{k+1} - N(l)]_{v^k}^+$;
- $\xi_{l_i}^{k+1} = \xi_{l_i}^k + \eta_2[c_{l_i}^{k+1} - r]_{\xi_{l_i}^k}^+$;
- $h_{l_i}^{k+1} = h_{l_i}^k + \eta_3[1 - c_{l_i}^{k+1}]_{h_{l_i}^k}^+$;
- if $|c^{k+1}(l) - c^{k+1}(l)| < \epsilon$, the gradient algorithm terminates.

where $\{c, v, \xi, h\}_*^k$ denote the values of the corresponding parameters during the kth iteration of the algorithm.

Next, we proceed to prove the convergence of this algorithm by adopting the method of Lyapunov stability theory [12]. We first define the following Lyapunov function:

$$V(\nu, \xi_{l_i}, h_{l_i}) = \int_{\nu^*}^{\nu} \frac{(\xi - \nu^*)}{\eta_1} d\xi + \sum_i \int_{\xi_{l_i}^*}^{\xi_{l_i}} \frac{(\xi - \xi_{l_i}^*)}{\eta_2} d\xi + \sum_i \int_{h_{l_i}^*}^{h_{l_i}} \frac{(\xi - h_{l_i}^*)}{\eta_3} d\xi \tag{20}$$

where ν^*, $\xi_{l_i}^*$ and $h_{l_i}^*$ is the optimal solution of D. It can be shown that $V(\nu, \xi_{l_i}, h_{l_i})$ is positive definite.

Denote by $\dot{V}$ the derivative of $V(\nu, \xi_{l_i}, h_{l_i})$ with respect to time. With the following lemma, we get $\dot{V} \leq 0$.

Lemma 2. *The subgradient of $D(\nu, \xi_{l_i}, h_{l_i})$ at ν is given by:*

$$\frac{\partial D}{\partial \nu} = N(l) - \sum_{i=1} m_{l_i} \tilde{c}_{l_i}$$

Similarly, the subgradient of $D(\nu, \xi_{l_i}, h_{l_i})$ and ξ_{l_i}, h_{l_i} are:

$$\frac{\partial D}{\partial \xi_{l_i}} = r - \tilde{c}_{l_i}; \quad \frac{\partial D}{\partial h_{l_i}} = \tilde{c}_{l_i} - 1$$

respectively where $\tilde{c}_{l_i}$ minimizes $f(\nu, \xi_{l_i}, h_{l_i}, c_{l_i})$.

Theorem 2. *When the step size η_1, η_2, η_3 are positive, the above gradient projection algorithm can converge to the global optimal.*

Proof. First, the trajectories of V converge to the set $S = \{(\nu, \xi_{l_i}, h_{l_i}) | \dot{V}(\nu, \xi_{l_i}, h_{l_i}) = 0\}$ according to the Lasalle's Principle [12]. Then, based on the proof of $\dot{V} \leq 0$, for all $(\nu, \xi_{l_i}, h_{l_i}) \in S$, $D(\nu, \xi_{l_i}, h_{l_i}) = D^*$ which is the optimal value. Thus, all the elements in S must be global optimal solutions. Hence, we conclude that the gradient projection algorithm converges to the global optimal. □

Not only that the algorithm is guaranteed to converge to the global optimal, we can further tune its parameters to speed up the convergence in an actual implementation. Fig. 1 depicts the results of a matlab-based simulation experiment to demonstrate the fast convergence rate of the gradient projection algorithm. In this experiment, we assume that, in a particular time slot l, there are 4 jobs waiting to be scheduled, i.e., $\chi(l) = \{J_{l_1}, J_{l_2}, J_{l_3}, J_{l_4}\}$. The cluster has 100 available machines and the number of tasks for each job are 10, 20, 5 and 10 respectively. Assume $F_i(t)$ to follow the Pareto Distribution where $F_i(t) = 1 - (\frac{\mu_i}{t})^2$ for $t \geq \mu_i$ and $\mu_1 = 1, \mu_2 = 2, \mu_3 = 1, \mu_4 = 2$ and the number of copies for each task is given by $r = 8$. We tune the parameters η_1, η_2, η_3 to be 0.2, 0.3, 0.4 respectively. As shown in Fig. 1, this algorithm can converge very fast to the optimal solution.

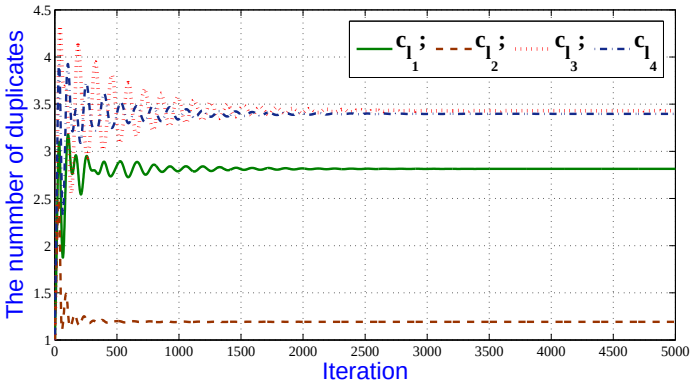

Fig. 1. The convergency performance of the gradient projection algorithm. The number of duplicates for the tasks in each job iterates and can converge to the optimal value. c_{l_i} represents the number of duplicates for each task in job l_i where $i = 1, 2, 3, 4$.

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;
end
if Σ_i m_{l_i} < N(l) then
    solve P2 and assign duplicates of the tasks in J_{l_i} based on the optimization result;
else
    for Job J_{l_i} in χ(l) do
        assign only one copy for each task of J_{l_i};
        update N(l);
        if N(l) == 0 then
            return;
        end
    end
end
return;
```

Algorithm 1: Smart Cloning Algorithm

B. The design of the Smart Cloning Algorithm (SCA)

Following the analysis in subsection IV-A, there exists a case where there is no space for cloning the jobs at the beginning of a particular time slot. In this scenario, it does not make sense to solve P2. Instead, we adopt a smallest remaining workload first scheme. It is well known in scheduling literature that the Shortest Remaining Processing Time (SRPT)

TABLE I
GOOGLE TRACE DATA STATISTICS

Total number of Jobs	6064
Trace duration (s)	35032
Average number of tasks per job	26.31
Minimum task duration (s)	12.8
Maximum task duration (s)	22919.3
Average task duration (s)	1179.7

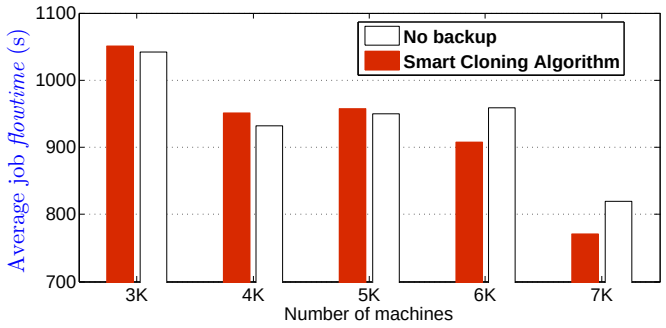

Fig. 2. Comparison of the average *flowtime* of the trace data running on different number of machines.

scheduler is optimal for overall flowtime on a single machine where there is one task per job. The SRPT-based approach has been adopted widely for the scheduling in a parallel system as presented in [13] and [21]. It is demonstrated to be more efficient for job completion than other scheduling algorithms in [6], [7], [16]–[18]. Based on P2 and SRPT, we propose the Smart Cloning Algorithm (SCA) below.

SCA consists of two separate parts. At the beginning of each time slot, we first schedule the remaining tasks of the 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 $\chi(l)$ according to the increasing order of the workload in each J_{l_i} and schedule the jobs based on this order and only one copy of each task is created. Notice that the resultant workload is the product of m_{l_i} and $\mathbb{E}[x_j^i]$. The corresponding pseudo-code is given in Algorithm 1.

C. Performance evaluation for SCA

In this section, we evaluate the performance of SCA via extensive simulations driven by Google cluster-usage traces [14]. The traces contain the information of job submission and completion time of Google services on a cluster of 12K servers. It also includes the number of tasks as well as the duration of each task. From the traces, we extract the statistics of more than 6000 jobs during a 12-hour period as shown in Table I. We already exclude those jobs which have specific constraints on machine attributes.

Baseline: We use the scheduling strategy of Microsoft Mantri as the baseline. This speculative execution scheme is demonstrated to be the best one among the straggler detection based schemes in the literature [4]. We use job *flowtime* and job resource consumption as the performance metrics for

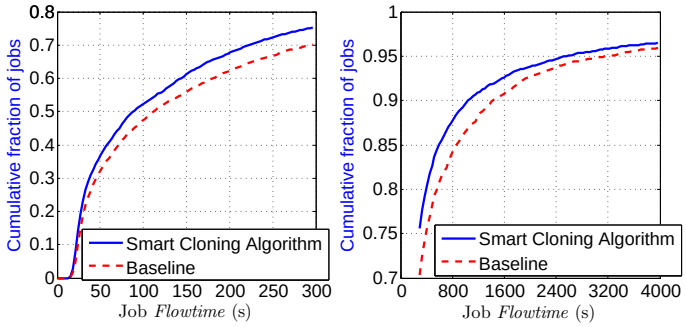

Fig. 3. The comparison between SCA and baseline for job *flowtime*. Left panel shows the CDF of Job *flowtime* with *Flowtime* less than 300 seconds while the right panel shows *Flowtime* more than 300 seconds.

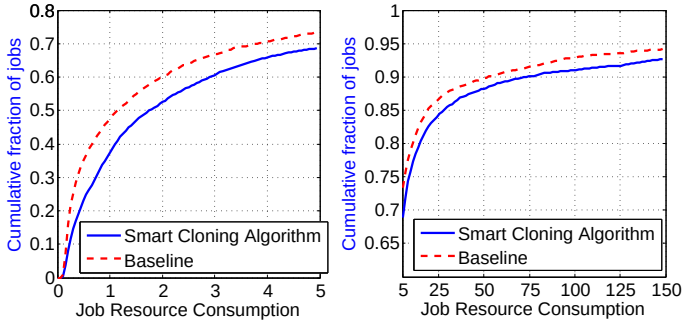

Fig. 4. The comparison between SCA and baseline in terms of resource consumption. Left panel shows the CDF of resource consumption with resource consumption less than 5 units while the right panel shows resource consumption more than 5 units.

comparison. The length of each time slot is 1 second and we scale down all the values of job resource consumption by 100 times for ease of presentation.

1) Characterizing the threshold for lightly and heavily loaded regime: Based on the analysis in Section III-B, we obtain the threshold for the lightly loaded and heavily loaded regimes. Rather than deriving the job arrival rate directly, we derive the number of machines under which Equation (7) is satisfied. For the given job trace illustrated in Table I, this theoretical threshold is around 6000. 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. 2 depicts the comparison result of the average job *flowtime* between SCA ($\gamma = 0$) and the pure SRPT-based scheduling strategy without task backup. It shows that, when the number of machines exceeds 6K, SCA performs better.

2) Impact of γ: We run the job trace on 12K machines and tune γ from zero to one under SCA. It shows that when γ is above 0.1, the cluster performance in terms of job *flowtime* and resource consumption under SCA is close to the baseline. However, when γ is below 0.1, the average job *flowtime* under SCA is much smaller than that under the baseline.

3) Comparison between SCA and baseline: Following the simulations above, we proceed to compare the performance of SCA and baseline under $\gamma = 0.05$. Fig. 3 and Fig. 4 depicts the cumulative density function (CDF) of job *flowtime* and resource consumption respectively. The average job *flowtime*

reduces by 22% under SCA compared to the baseline. It is worth noting that, under SCA, more than 75% of jobs can finish within 300 seconds. In contrast, about 70% of jobs can finish within 300 seconds under the baseline. However, the average job resource consumption in SCA is 20% larger than the baseline. The reason behind is that SCA launches duplicates in a greedy fashion and therefore needs to consume more resources for reducing the job *flowtimes*.

V. Design of Straggler-Detection-based algorithm for the heavily loaded regime

In the heavily loaded cluster, i.e., $\lambda \geq \lambda^U$, the cloning-based scheme cannot be applied and only the Straggler-Detection-based approach is efficient. In the literature, Microsoft Mantri [4] chooses to schedule a speculative copy if $\mathbb{P}(t_{rem} > 2 * t_{new}) > \delta$ is satisfied when there are available machines. We extend this scheme and propose the *Enhanced Speculative Execution* (ESE) algorithm. The scheduling decision is made in each time slot and we also leave some space for cloning the small jobs. Usually the small job represents for interactive applications like query and have very strict latency requirement. When the workload is heavy, the probability of making speculative copies for the tasks of small jobs is low in Mantri's scheduling algorithm. Cloning for small jobs only incurs a small amount of resource consumption while contributing a lot to the job *flowtime*.

A. The Enhanced Speculative Execution algorithm

The ESE algorithm also includes three scheduling levels. Explicitly, at the beginning of time slot l, the scheduler estimates the remaining time of each running task and puts the tasks whose remaining time satisfy a particular condition into the 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,

$$D(l) = \{\delta^i_j : c^i_j(l) = 1 \ \& \ t^i_{j,rem}(l) > \sigma_i \cdot \mathbb{E}[x^i_j]\}$$

All the tasks in $D(l)$ are sorted according to the decreasing order of $t^i_{j,rem}(l)$ and the scheduler assigns a duplicate of each task in $D(l)$ based on this order. In the same way as P3, we also need to find an appropriate value for σ_i that the expected resource consumption for a single task is minimized.

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

$N(l)$ is updated after the above scheduling. If there are still available machines and some jobs waiting to be scheduled, the scheduler then tries to do the cloning for small jobs and assign the number of copies which can maximize the difference between the job utility and total resource consumption. For big jobs, no cloning will be made. More precisely, denote by $\chi(l) = \{J_{l_1}, J_{l_2}, \cdots\}$ the job set which contains all the jobs have not been scheduled yet in the cluster. The jobs in $\chi(l)$ are sorted according to the increasing order of workloads. For small job J_{l_i} which satisfies $m_{l_i} < \eta \cdot \frac{N(l)}{|\chi(l)|}$ and $\mathbb{E}[x^{l_i}_j] < \xi$ in $\chi(l)$, the optimal number of copies cloned for task in J_{l_i} is determined by the following equation.

$$c^*_{l_i} = \arg\max_{c_{l_i}} U\left(\mathbb{E}[t_{l_i}], m_{l_i}\right) - \gamma \cdot \sum_{j=1}^{m_{l_i}} c_{l_i} \cdot \mathbb{E}[t^{l_i}_j] \qquad (21)$$

The parameters η and ξ can be tuned based on the cluster workload. Here, c_{l_i}, t_{l_i}, m_{l_i} and $t^{l_i}_j$ are the same as Section IV and $w^j_{m_{l_i}}$ is equal to l for all j. The corresponding pseudocode is given in Algorithm 2 in the below.

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 the 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;
    end
end
for the 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;
    end
end
for the job J_{l_i} in χ(l) do
    if m_{l_i} < η · N(l)/|χ(l)| & E[x^{l_i}_j] < ξ then
        Compute (c_{l_i})* base on Equation (21);
        Assign (c_{l_i})* duplicates for each task in J_{l_i};
        update N(l);
    else
        Assign one duplicate for each task in J_{l_i};
        update N(l);
    end
    if N(l) == 0 then
        return;
    end
end
return;
```

Algorithm 2: Enhanced Speculative Execution Algorithm

B. Approximate Analysis for σ_i in ESE Algorithm

σ_i has a great impact on the system performance. In the following analysis, we aim to find an appropriate σ_i in the ESE algorithm through minimizing the expected resource consumption for one particular task.

Define the random variable R^i_j as the resource that task δ^i_j consumes in the cluster. For convenience, denote by θ^i_j the

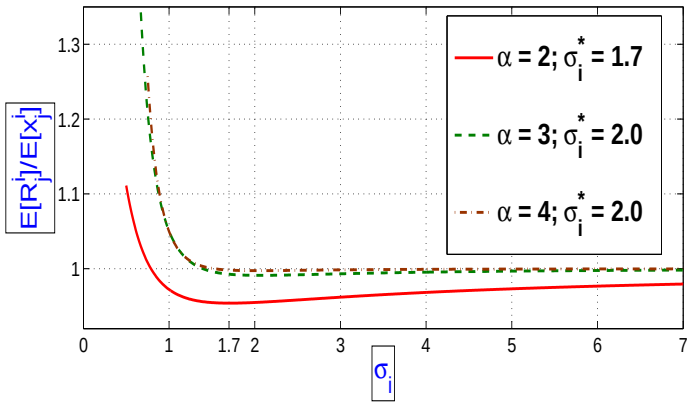

Fig. 5. The illustration of $E[R_j^i]/E[x_j^i]$ under different σ_i for pareto distribution when $\alpha = 2, 3, 4, 5$.

event that $x_j^i > \sigma_i \mathbb{E}[x_j^i]$. The expectation of R_j^i therefore can be expressed in the following equation:

$$\mathbb{E}[R_j^i] = \mathbb{E}\left[R_j^i \left|\theta_j^i\right.\right] Pr(\theta_j^i) + \mathbb{E}\left[R_j^i \left|\overline{\theta_j^i}\right.\right] Pr(\overline{\theta_j^i}) \quad (22)$$

Moreover, we have

$$\mathbb{E}\left[R_j^i \left|\overline{\theta_j^i}\right.\right] Pr(\overline{\theta_j^i}) = \int_0^{\sigma_i \mathbb{E}[x_j^i]} t d(F_i(t)) \quad (23)$$

We give the following definition which helps to derive the expression of $\mathbb{E}[R_j^i]$.

Definition 2. *The asktime of a running task i is the earliest time that the scheduler checks whether it should duplicate a new copy for task i on available machines or not.*

Assume the interval of each time slot is short enough, then the *asktime* of δ_j^i can be treated as uniformly distributed in the interval [0, x_j^i] due to the cluster is heavily loaded. At this *asktime*, the scheduler assigns a new copy for δ_j^i if $t_{rem} > \sigma_i \mathbb{E}[t_{new}]$ is satisfied. Hence,

$$\mathbb{E}\left[R_j^i \left|\theta_j^i\right.\right] Pr(\theta_j^i) = \int_{\sigma_i \mathbb{E}[x_j^i]}^{\infty} d(F_i(t))[\int_0^{t-\sigma_i \mathbb{E}[x_j^i]} \frac{1}{t}(x + 2\mathbb{E}[\min\{t-x, t_{new}\}])dx + \sigma_i \mathbb{E}[x_j^i]] \quad (24)$$

In Equation (24), $\mathbb{E}[min\{t-x, t_{new}\}]$ is given by:

$$\mathbb{E}[min\{t-x, t_{new}\}] = \int_0^{t-x} w d(F_i(w)) + (t-x)(1-F_i(t-x)) \quad (25)$$

Substitute Equation (23),(24) and (25) into (22), $\mathbb{E}[R_j^i]$ can be determined and it's only a function of σ_i. We obtain the optimal value of σ_i that minimizes the expected resource consumption via letting the derivative of $\mathbb{E}[R_j^i]$ be 0.

We illustrate the picture of $\mathbb{E}[R_j^i]$ in Fig. 5 under different σ_i for the pareto distribution. α is the heavy-tail order and σ_i^* is the optimal value. It indicates that when σ_i is close to 1.7 where α is 2, $\mathbb{E}[R_j^i]$ achieves the minimum value. We also compare the different optimal value for σ_i when the heavy-tail order changes. As shown in Fig. 5, σ_i^* increases along with α. Moreover, for all $\alpha \geq 3$, σ_i^* is very close to 2.0.

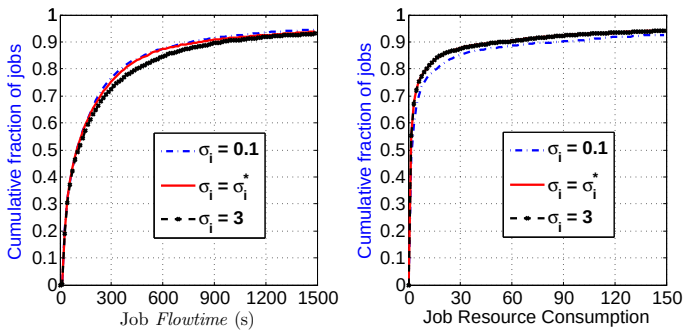

Fig. 6. The comparison between different σ_i in the ESE algorithm. The left panel and right panel depicts the comparison result of job *flowtime* and resource consumption respectively.

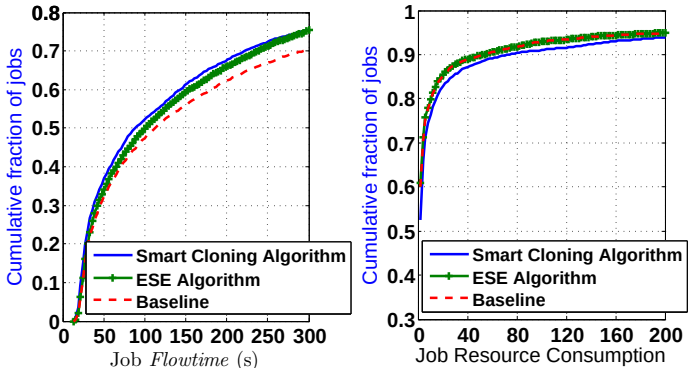

Fig. 7. Comparison between SCA and ESE algorithm in the lightly loaded cluster. The left panel and right panel depicts the comparison result of job *flowtime* and resource consumption respectively.

C. Performance Evaluation for ESE

In this section, we evaluate the performance of ESE via extensive simulations driven by the same trace in Section IV-C.

1) 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 resource consumption in the cluster with 12K servers. The optimal value for σ_i is obtained according to the theoretical analysis in Section V-B. As an representative, we illustrate the comparison result under σ_i being 0.1 and 3. It indicates in Fig. 6 that when σ_i takes the optimum, both of the job *flowtime* and resource consumption attains the minimum.

2) Comparison between SCA and ESE in the lightly loaded cluster: Both of the cloning and straggler detection approach can improve the system performance in the lightly loaded regime. However, it is still unknown which one is better. In this subsection, we compare the performance of SCA and ESE in the lightly loaded cluster, to be more specific, we rerun the job trace on 12K machines.

Fig. 7 depicts the comparison result between SCA and ESE for both the job *flowtime* and resource consumption. It indicates that the *flowtimes* of small jobs under the ESE algorithm are larger than that under SCA. There is no much difference between these two algorithms in terms of the flowtimes of big jobs. However, the average resource consumption of SCA is larger than that of ESE. As explained in Section III-A, ESE

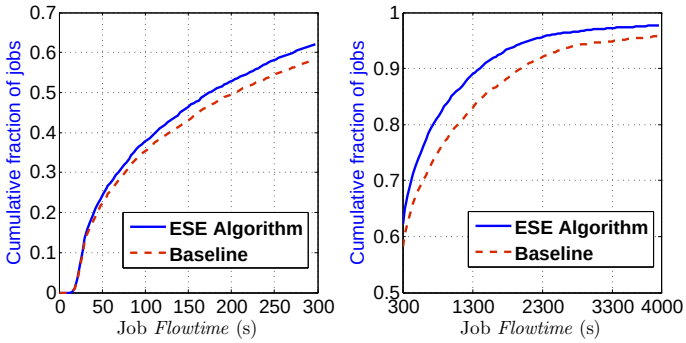

Fig. 8. Comparison of job *flowtime* between ESE algorithm and baseline in the heavily loaded cluster.

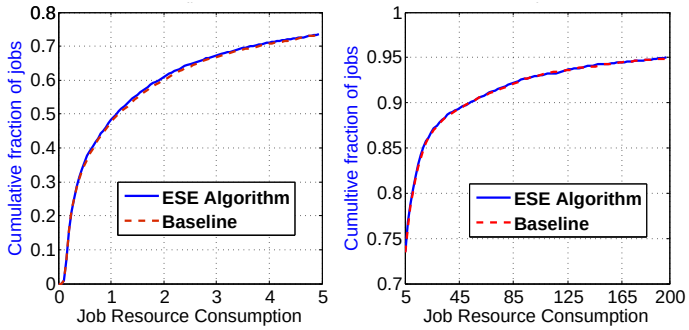

Fig. 9. The comparison of job resource consumption between ESE algorithm and baseline in the heavily loaded cluster.

algorithm incurs extra system instrumentation for monitoring the task progress. To make a fair comparison, such overhead should not be neglected.

3) The performance of ESE Algorithm in the heavily loaded regime: In this simulation, we rerun the same job trace on 6K machines. The results are illustrated in Fig. 8 and Fig. 9. Actually, the performance of ESE algorithm in the heavily loaded cluster is just slightly worse than it in the lightly loaded cluster. As shown in Fig. 8, about 63% jobs complete in 300 seconds under ESE while only 58% jobs complete in 300 seconds in the baseline. Moreover, the average job *flowtime* reduces by nearly 35% under the ESE algorithm compared with the baseline. Fig. 9 shows that the resource consumption under these two schemes are roughly the same. We conclude that the ESE algorithm can beat the baseline in both lightly loaded and heavily loaded regimes.

VI. Conclusions and Future Work

In this paper, we address the speculative execution issue in the parallel computing cluster and focus on two metrics which are job *flowtime* and resource consumption. We utilize the distribution information of task duration and build a theoretical framework for making speculative copies. We categorize the cluster into lightly loaded and heavily loaded cases and derive the cutoff threshold for these two operating regimes. Moreover, we propose the Smart Cloning Algorithm which is derived from optimizing a utility maximization problem when the cluster is lightly loaded. On the other hand, we design the Enhance Speculative Execution Algorithm in the heavily loaded cluster. The trace driven simulations demonstrate that our proposed algorithms achieve good performance in terms of both job *flowtime* and resource consumption.

In the future work, we will design speculative execution schemes for more sophisticated jobs which have task-dependency constraint. Like the MapReduce applications, any reduce task can only begin after the map tasks finish within a job. In addition, we consider to formulate an optimization problem for speculative execution in which only partial information about the job statistics are known *a priori*, i.e., the first and second moments of task duration. We have presented the details of these extensions in [19].

References

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

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

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

[4] 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 *USENIX OSDI*, Vancouver, Canada, October 2010.

[5] D. Breitgand, R. Cohen, A. Nahir, and D. Raz. On cost-aware monitoring for self-adaptive load sharing. *IEEE JSAC*, 28(1):70–83, January 2010.

[6] H. Chang, M. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee. Scheduling in MapReduce-like systems for fast completion time. In *Proceedings of IEEE Infocom*, pages 3074–3082, March 2011.

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

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

[9] R. B. Cooper. *Introduction to queueing theory*. The Macmillan Company, New York, 1972.

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

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

[12] Lefechetz, S., Lasalle, and J.P. *Stability by Lyapunov's direct method with applications*. Academic Press, New York, 1961.

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

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

[15] X. Sun, C. He, and Y. Lu. Esamr: An enhanced self-adaptive MapReduce scheduling algorithm. In *the 18th International Conference on Parallel and Distributed Systems (ICPADS)*, December 2012.

[16] J. Tan, X. Meng, and L. Zhang. Delay tails in MapReduce scheduling. In *Proceedings of SIGMETRICS*, pages 5–16, London, United Kingdom, June 2012.

[17] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang. Map task scheduling in MapReduce with data locality: Throughput and heavy-traffic optimality. In *Proceedings of IEEE Infocom*, Turin, Italy, April 2013.

[18] Q. Xie and Y. Lu. Degree-guided map-reduce task assignment with data locality constraint. In *Information Theory Proceedings (ISIT)*, pages 1–6, July 2012.

[19] H. Xu and W. C. Lau. Task-cloning algorithms in a MapReduce cluster with competitive performance bounds. In *CoRR abs/1501.02330*, January 2015.

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

[21] 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.