Speculative Execution for a Single Job in a MapReduce-like System

Huanle Xu
The Department of Information Engineering
The Chinese University of Hong Kong
Email: xh112@ie.cuhk.edu.hk

Wing Cheong Lau
The Department of Information Engineering
The Chinese University of Hong Kong
Email: wclau@ie.cuhk.edu.hk

***Abstract*—Parallel processing plays an important role for large-scale data analytics. It breaks a job into many small tasks which run parallel on multiple machines such as MapReduce framework. One fundamental challenge faced to such parallel processing is the straggling tasks as they can delay the completion of a job seriously.**

In this paper, we focus on the speculative execution issue which is used to deal with the straggling problem in the literature. We present a theoretical framework for the optimization of a single job which differs a lot from the previous heuristics-based work. More precisely, we propose two schemes when the number of parallel tasks the job consists of is smaller than cluster size. In the first scheme, no monitoring is needed and we can provide the job deadline guarantee with a high probability while achieve the optimal resource consumption level. The second scheme needs to monitor the task progress and makes the optimal number of duplicates when the straggling problem happens. On the other hand, when the number of tasks in a job is larger than the cluster size, we propose an Enhanced Speculative Execution (ESE) algorithm to make the optimal decision whenever a machine is available for a new scheduling. The simulation results show the ESE algorithm can reduce the job *flowtime* by 50% while consume fewer resources comparing to the strategy without backup.

***Index Terms*—MapReduce, speculative execution, resource optimization, theoretical analysis**

I. Introduction

The parallel processing framework plays an important role for large-scale data analytics and are widely applied in the cloud environment which consists of tens of thousands of machines. In Amazon EC2, the elastic MapReuce cluster is an important service which provides an easy way for users to write parallel programs and deploy it among a large number of machines.

The parallel processing framework breaks a computation into small tasks that run in parallel on multiple machines, and can scale easily to very large clusters. Typically, the stored data are organized in distributed file systems. During the computation, the entire input data is partitioned into several small chunks which are assigned to an individual machine for partial computation of results. Once a user submits a job to the cluster, the number of tasks can be defined by the user or the framework itself based on the input data size. Take MapReduce framework for example, the tasks are assigned to the machines (worker node) in the computing cluster by a master node which keeps track of the status of these tasks to manage the computation process. Each worker node has a task tracker that periodically sends status to the job tracker via heartbeats every few seconds. In response, a scheduler on the master node assigns tasks to each worker node. Hadoop is a very popular open source implementation of MapReduce and has bee deployed in industry widely [2]. Hadoop's default scheduler run jobs in FIFO order. When the scheduler receives a heartbeat indicating that a slot is free, it scans through jobs in order of priority and submit time to find one with an appropriate task. Due to the large number of jobs sharing limited resources, an efficient job scheduler is critical for achieving good fairness among jobs. Fair Scheduler [3], [6] as the de facto industry standard, is one of the most widely used schedulers in practice. There has been many different schemes proposed in the literature to optimize the scheduler in Hadoop [4-5], [7-11].

Usually, the parallel processing framework can automatically handle failures. If a node is available but is performing poorly, then the task running on it will take a long time to finish and is so called a straggler. Experience from an operational MapReduce cluster indicates that stragglers delay the job execution time and degrade the cluster throughput significantly. There mainly exists three categories of root causes for stragglers which are machine characteristics, network contention and input data skew as discussed in [13]. Speculative execution is a common approach for dealing with the straggler problem by simply backing up those slow running tasks. Explicitly, upon detecting a straggler, the scheduler launches a speculative copy on another machine with the expectation to finish the task faster. Google has noted that even the naive speculative execution approach can improve job response time by 44% [1] in the MapReduce framework.

Figure 1 shows a job that has seven tasks which are labelled from number 1 to 7 and there are two machines in the cluster. Normal tasks run for time t and $2t$ time units. However, task 2 which is suffering from a straggler has a runtime of $5t$ time units. The timeline at the top shows a baseline which ignores the straggler and finishes at $7t$ time units. The bottom timeline shows with duplicating the straggler on another machine can decrease the job finish time to $6t$ time units.

The research work for speculative execution in the literature mainly focus on two aspects: i) How to identify the stragglers early and accurately. ii) How to select the right candidate for backup to reduce the completion time. However, all the algorithms need to choose appropriate parameters and do not distinguish between the loading conditions in the cluster. Actually, when the cluster is lightly loaded, there is a large room to make duplicates for the tasks such that their work can finish earlier. In this way, we choose to characterize the loading of the cluster into different regimes. On the other hand, the trace data from the MapReduce cluster [13] shows that the duration of straggler tasks follows the heavy tail distribution and 10% of the stragglers take more than 5 times the median duration. We consider to utilize the distribution information of task duration for our speculative execution scheme design. What's more, none of the algorithms proposed in literature consider the tradeoffs between job finish time and resource consumption. Intuitively, if we make more copies of the running task, the job will finish earlier with the cost of consuming more resources. Hence, we also explore the tradeoffs between the job *flowtime* and the total resource consumption in this paper.

In our work, we focus on the speculative execution of one single job and categorize the cluster's loading into two cases, namely, lightly loaded and heavily loaded. We jointly optimize the job *flowtime* time with resource consumption and propose two different strategies in the lightly loaded cluster. Moreover, we propose the ESE algorithm for task backup when the cluster is heavily loaded. The simulation results show that our proposed ESE algorithms perform well comparing to the baseline algorithm.

The rest of this paper is organized as follows. Section II discusses the related work on speculative execution in parallel processing clusters. It provides a background on recent research work . Section III introduces some preliminaries and basic assumptions in the parallel processing clusters. Section IV describes our optimization problem while presents two different strategies when the cluster is lightly loaded. Section V presents our ESE algorithm design for the heavily loaded cluster. Section VI shows the simulation results for the ESE algorithm and Section VII concludes our work.

II. Related work for Speculative Execution

Several speculative execution strategies have been proposed in the literature in the MapReduce cluster. Google only begins to do backups when a MapReduce operation is close to complete and shows that speculative execution can decrease the job execution time by 44% [1]. Speculative execution is also implemented in Hadoop [2] and Microsoft Dryad [12] to improve the performance in their clusters. At the beginning, the speculative execution in Dryad is roughly the same as Google. However, the research group from Berkeley presents a new strategy called LATE (*Longest Approximate Time to End*) [14] in the Hadoop-0.21 implementation. It monitors the progress rate of each task and estimates their remaining time. Tasks with their progress rate below *slowTaskTherehold* are chosen as backup candidates and the one with the longest remaining time is given the highest priority when the number of backups in the cluster does not exceed *speculativeCap*.

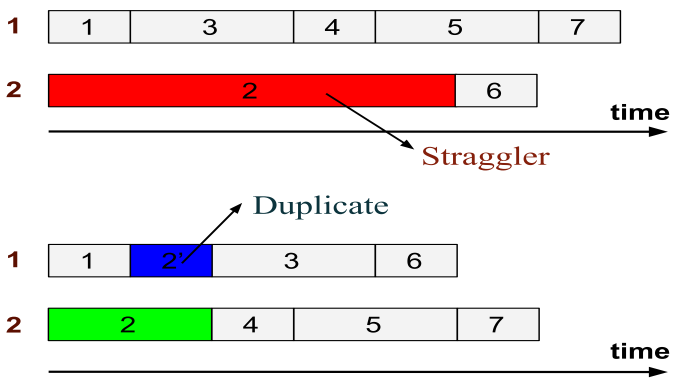

Fig. 1. Duplicate the task with an unusually long time on another machine to make it finish much faster.

Later, Microsoft Mantri [13] proposes a new speculative execution strategy for Dryad. Mantri estimates for each task the remaining time to finish, t_{rem}, and predicts the duration of a new copy of the task, t_{new}. Once a machine is idle, Mantri then makes a decision whether to do backup or not based on the statistics of t_{rem} and t_{new}. More precisely, a duplicate is scheduled if $\mathbb{P}(t_{rem} > 2 * t_{new}) > \delta$ is satisfied. By default, $\delta = .25$. Hence, Mantri schedules a duplicate only if the total resource consumption decreases. Moreover, Mantri allows killing a task if it really takes a long time to complete the remaining work.

To accurately and promptly identify stragglers, [15] proposes a Smart Speculative Execution strategy and [16] presents an Enhanced Self-Adaptive MapReduce Scheduling Algorithm respectively. The main contributions that [15] makes are twofold: i) Use exponentially weighted moving average to predict process speed and compute a task's remaining time; ii) Determine which task to backup based on the load of a cluster using a cost-benefit model. Recently, the same research group from Berkeley proposes a clone scheme to mitigate the straggler in MapReduce cluster [17]. In their scheme, the full *cloning* algorithm is proposed to avoid waiting for speculative execution. When the majority of the jobs are small, the clones only consume a small fraction of the resources.

III. Preliminaries and Problem Definition

Consider a parallel computing cluster which consists of M worker nodes and each node holds one slot, i.e., it can only support one task at any time. We apply the general FIFO scheduler to schedule the jobs that come into the cluster, namely, each time, the job with the highest priority is scheduled if there are available machines, other jobs which have the lower priority should wait in the queue. For simplicity, we assume the cluster is homogeneous in the sense that all the worker nodes are identical.

In this cluster, we assume t_i be the duration of task i (the time between the task is initialized and the task is finished) , follows a heavy-tail distribution, i.e., $t_i \sim F(\lambda_i, t)$, where F is a specific distribution and λ_i is a parameter related to the input data size for task i. Each job consists of several tasks which can run parallel on multiple machines.

If a task i runs t_i time on a worker node, then it will consume $c * t_i$ amount of resource on this node where c is a constant number. Actually, there exists a startup time for each task. And for simplicity, we do not consider it.

In this paper, we focus on the optimization of speculative execution for one single job. Without loss of generality, consider a job J which is submitted by a particular user to the cluster and consists of N tasks. For ease of presentation, we often exchange the two notations, namely, worker node and machine in the following descriptions.

Here, we define the job *flowtime* which is an important metric we capture in this paper.

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

In the next section, we will introduce our models and present the analysis under two different regimes: i) The cluster is lightly loaded, i.e., the total number of slots is larger than the number of tasks in job J. ii) The cluster is heavily loaded, i.e., the total number of slots is smaller than the number of tasks in job J. The intuition of distinguishing between the loading in the cluster is that the space for making speculative copies differs a lot under different conditions. In the lightly load case, the scheduler can make a large number of speculative copies so as to achieve a small *flowtime* for job J. While in the heavily loaded cluster, there is only a small number of available machines and the scheduler needs to make a tradeoff between scheduling duplicates and assigning new tasks at most time. Assigning too many duplicates can easily block the scheduling of other tasks and may make the *flowtime* of job J even longer.

IV. System optimization when the cluster is lightly loaded

When the cluster is lightly loaded, i.e $M > N$, all the tasks of Job J can be launched simultaneously in the cluster while some machines still remain idle. In this case, we can choose to do more backups for the tasks running in the cluster. Moreover, we assume that each task can at most maintain r copies in the cluster. The objective is to minimize the total resource consumption for job J while meeting the deadline constraint with a high probability. Denote by r_i the number of copies task i keeps and x_i^j the duration of jth copy. Further, we assume that all the x_i^j are i.i.d random variables and follow the same distribution as x_i for $1 \le j \le r_i$. Here we propose two different strategies: i) Pre-compute, make duplicate for each task at the very beginning. ii) Dynamically assign the duplicates (backups) based on the progress of the running tasks.

A. Pre-compute at the very beginning

We begin to make backups for tasks at the same time as job J begins to run in the cluster. Under such a strategy, there is no need to monitor the progress of each task any more and the user can define the deadline for job J. Our objective is to minimize the total resource consumption while satisfy the user's deadline with a high probability. The formulation yields

$$\min_{r_i} \quad \mathbb{E}\left[\sum_{i=1}^{N} r_i \cdot c \cdot t_i\right] \tag{1a}$$

$$s.t. \quad \sum_{i=1}^{N} r_i \le M \tag{1b}$$

$$r_i \le r \qquad \forall i \tag{1c}$$

$$t_i = \min\{x_i^1, x_i^2, \ldots, x_i^{r_i}\} \qquad \forall i \tag{1d}$$

$$Pr[\max\{t_{1,}t_2, \ldots, t_N\} > \frac{\rho}{\lambda}] \le \epsilon \tag{1e}$$

In the formulation, $\rho \ge 1$ is a number defined by the user to capture the deadline and $1/\lambda = \max\{1/\lambda_1, 1/\lambda_2, \ldots, 1/\lambda_N\}$. The first constraint states that the total number of copies of job J is no more than M and the third constraint states that as soon as one copy of task i finishes, the task completes. The last constraint states that $flow(J) = max\{t_{1,}t_2, \ldots, t_N\}$ should not hit the deadline $\frac{\rho}{\lambda}$ with a certain probability $1-\epsilon$. We proceed to derive the solution of this optimization problem in the following.

Define $F_i(x) = F(\lambda_i, x) = Pr\{x_i < x\}$ and $H_i(t)$ as the distribution of t_i. We have

$$H_i(t) = 1 - (1 - F_i(t))^{r_i} \tag{2}$$

and

$$Pr[\max\{t_{1,}t_2, \ldots, t_N\} > \frac{\rho}{\lambda}] = 1 - \prod_{i=1}^{N} H_i(\frac{\rho}{\lambda}). \tag{3}$$

Further,

$$\mathbb{E}[t_i] = \int_0^\infty t \cdot d(H_i(t)) = \int_0^\infty (1 - H_i(t))dt. \tag{4}$$

Thus

$$\mathbb{E}\left[\sum_{i=1}^{N} r_i \cdot c \cdot t_i\right] = c \cdot \sum_{i=1}^{N} \underbrace{r_i \int_0^\infty (1 - F_i(t))^{r_i} dt}_{\triangleq f_i(r_i)} \tag{5}$$

Lemma 1. *$r_i \cdot \int_0^\infty (1 - F_i(t))^{r_i} dt$ is a convex function of r_i on the condition that $\ln(1 - F_i(t))$ is convex.*

Proof. See Appendix A. □

For all the commonly used heavy-tail distributions, $\ln(1 - F_i(t))$ is convex. Moreover, the last constraint implies that

$$\sum_{i=1}^{N} \ln(H_i(\frac{\rho}{\lambda})) \ge \ln(1 - \epsilon). \tag{6}$$

Applying the inequality that $\ln(1-x) \le -x$ when $x \in [0, 1)$, we can relax the last constraint as follows:

$$\sum_{i=1}^{N} (1 - F_i(\frac{\rho}{\lambda}))^{r_i} \le \ln(\frac{1}{1-\epsilon}). \tag{7}$$

For ease of presentation, let $g_i(r_i)$ denote $(1-F_i(\frac{\rho}{\lambda}))^{r_i}$ for all i and C denote $\ln(\frac{1}{1-\epsilon})$. Observe that $g_i(r_i)$ is also a convex function of r_i for all i. If the condition in Lemma 1 is positive, the objective in Formulation (11) is also convex. Thus, we can adopt the convex optimization technique to solve the problem.

We write down the Dual problem as

$$\max_{l,k,h_i} D(l,k,h_i); \quad D(l,k,h_i) = \min_{r_i} L(l,k,h_i,r_i) \tag{8}$$

$$\begin{aligned} L(l,k,h_i,r_i) = &\sum_{i=1}^{N} f_i(r_i) + l \cdot (\sum_{i=1}^{N} r_i - M) \\ &+ \sum_{i=1}^{N} h_i \cdot (r_i - r) + k \cdot (\sum_{i=1}^{N} g_i(r_i) - C) \end{aligned} \tag{9}$$

In (8-9), h_i (for $1 \le i \le N$), l and k are non-negative Lagrangian multipliers. Further, we write down the KKT conditions as follows:

$$\begin{aligned} f'_i(r_i^*) + w_i + l + kg'_i(r_i^*) = 0 &= 0 \quad \forall i \\ l^* \cdot (\sum_{i=1}^{N} r_i^* - M) &= 0 \\ h_i^* \cdot (r_i^* - r) &= 0 \quad \forall i \\ k^* \cdot (\sum_{i=1}^{N} g_i(r_i^*) - C) &= 0 \end{aligned}$$

In the above Equations, r_i^*, h_i^* (for $1 \le i \le N$), l^* and k^* are the optimal solutions to the primal and dual problem. Observe that when constraints (1b) & (1c) are inactive, l^* and h_i^* (for all i) are equal to zero. Thus, the solutions to the KKT equations can be easily computed. Based on this observation we have the following theorem.

Theorem 1. *When the constraints (1b) & (1c) are inactive and $F_i(t)$ is a pareto distribution for all i, the optimal solution for P1 is determined by the following equation.*

$$r_i^* = max\left\{ \left[\frac{1}{\alpha} \cdot log_{\frac{\rho}{\lambda\mu_i}} \left(\frac{\ln \frac{\rho}{\lambda\mu_i} \cdot \sum_{i=1}^{N} \frac{1}{\ln \frac{\rho}{\lambda\mu_i}}}{\ln \frac{1}{1-\epsilon}} \right) \right], 1 \right\} \tag{10}$$

where α is the heavy-tail order and μ_i is the mean of the task duration.

Proof. Let h_i and l be zero in the KKT equations and the result immediately follows. □

Based on this theorem, we can apply r^* in (10) to test whether the constraints (1b) & (1c) can be satisfied. If the answer is positive, then r_i^* must be the optimal number of copies each task should make. Otherwise, we adopt the primal-dual approach to solve P1 iteratively. We first illustrate the framework of the projection gradient algorithm and then show the design details.

Define the vector $r = (r_1, r_2, \cdots)$. The gradient projection algorithm is presented in the below.

- Initialize $r_i^0 = 1$ for all i, $l^0 = 0$, $h_i^0 = 0$ for all i, $k^0 = 0$.
- $r_i^{n+1} = r_i^n + \eta_1^i[f'_i(r_i^n) + l^n + h_i^n + kg'_i(r_i^n)]^+_{r_i^n} \quad \forall i;$
- $l^{n+1} = l^n + \eta_2[\sum_{i=1}^{N} r_i^{n+1} - M]^+_{l^n};$
- $h_i^{n+1} = h_i^n + \eta_3^i[r_i^{n+1} - r]^+_{h_i^n} \quad \forall i;$
- $k^{n+1} = k^n + \eta_4[\sum_{i=1}^{N} g_i(r_i^{n+1}) - C]^+_{k^n};$
- if $|r^{n+1} - r^n| < \epsilon$, the gradient projection algorithm terminates.

r_i^{n+1} is the value of r_i in the $(n+1)th$ iteration. Similarly, l^{n+1}, h_i^{n+1} and k^{n+1} denote the values of multipliers in the $(n+1)th$ iteration. Further, the stepsizes of the algorithm are determined by the following equations.

$$\begin{aligned} \eta_1^i &= \frac{1}{2\mu_i\alpha} \quad \forall i; \quad \eta_2 = \frac{1}{M} \\ \eta_3^i &= \frac{1}{r} \quad \forall i; \quad \eta_4 = 1 \end{aligned}$$

We approximate the last constraint as a linear inequality, thus the convergence of this algorithm is guaranteed according to the standard technique using Lyapunov function in [18].

B. Assign the duplicates based on the task progress

In this case, we only begin to make backups for task i if we detect an straggler which means the progress rate of i falls behind the average. To be more specific, a task is declared to be a straggler if its estimated remaining time to finish (t_{rem}) is greater than $\sigma \cdot \mathbb{E}[t_{new}]$ where t_{new} is the expected execution time of a new copy. To model the monitoring progress, we assume that we can detect the straggler for task i after it runs sx_i time in the cluster. And s is a constant number assumed to be known. Similarly, we assume that all the x_i^j are i.i.d random variables for $1 \le j \le r_i$. The objective is to minimize the total resource consumption which yields the following formulation.

$$\min_{r_i,\sigma} \quad \mathbb{E}\left[\sum_{i=1}^{N} r_i \cdot c \cdot t_i - \sum_{i=1}^{N} (r_i - 1) \cdot c \cdot sx_i^1\right] \tag{11a}$$

$$s.t. \quad \sum_{i=1}^{N} r_i \le M \tag{11b}$$

$$1 \le r_i \le r \quad \forall i \tag{11c}$$

$$r_i = 1 \quad if \ (1-s) \cdot x_i^1 < \frac{\sigma}{\lambda_i} \quad \forall i \tag{11d}$$

$$t_i = \min\{(1-s)x_i^1, x_i^2, \ldots, x_i^{r_i}\} + sx_i^1 \tag{11e}$$

In this formulation, x_i^1 is the running time of the first copy for task i. The differences between these two models are that here we choose to make backups for those tasks which "really" need help and do not have the deadline constraint. Moreover, the optimization variables are σ and r_i.

Define $y_i = min\{x_i^2, x_i^3, \ldots, x_i^{r_i}\}$ and $l_i = min\{(1-s)x_i^1, y_i\}$. Thus,

$$\mathbb{E}\left[\sum_{i=1}^{N} r_i \cdot ct_i - (r_i - 1) \cdot csx_i^1\right] = c \cdot \sum_{i=1}^{N} \mathbb{E}\left[r_i \cdot l_i + s \cdot x_i^1\right] \tag{12}$$

and

$$\begin{aligned}\mathbb{E}\left[r_i \cdot l_i + s \cdot x_i^1\right] &= \mathbb{E}\left[r_i \cdot l_i + s \cdot x_i^1 \middle| (1-s)x_i^1 \geq \frac{\sigma}{\lambda_i}\right] \\ &\cdot Pr\left\{(1-s)x_i^1 \geq \frac{\sigma}{\lambda_i}\right\} \\ &+ \mathbb{E}\left[r_i \cdot l_i + s \cdot x_i^1 \middle| (1-s)x_i^1 < \frac{\sigma}{\lambda_i}\right] \\ &\cdot Pr\left\{(1-s)x_i^1 < \frac{\sigma}{\lambda_i}\right\}\end{aligned} \tag{13}$$

We first find the optimal value for r_i under the fixed σ. Due to the constraints imposed in the formulation, we know that on the condition $(1-s)x_i^1 < \frac{\sigma}{\lambda_i}$, r_i is just 1. It implies $\mathbb{E}\left[r_i \cdot l_i + s \cdot x_i^1 \middle| (1-s)x_i^1 < \frac{\sigma}{\lambda_i}\right] = \mathbb{E}\left[x_i^1 \middle| (1-s)x_i^1 < \frac{\sigma}{\lambda_i}\right]$. This term is a constant when σ is determined, thus, we only need to consider the the case when $(1-s)x_i^1 \geq \frac{\sigma}{\lambda_i}$ and find an optimal value for r_i to minimize $\mathbb{E}\left[r_i \cdot l_i + s \cdot x_i^1 \middle| (1-s)x_i^1 \geq \frac{\sigma}{\lambda_i}\right]$.

As an illustrative example, we adopt the Pareto distribution, i.e., $Pr\{y_i \leq y\} = 1 - (\frac{\mu_i}{y})^{\alpha(r_i-1)}$. Thus, $\mathbb{E}\left[r_i \cdot l_i + s \cdot x_i^1 \middle| (1-s)x_i^1 \geq \frac{\sigma}{\lambda_i}\right] = r_i \cdot \mathbb{E}\left[l_i \middle| (1-s)x_i^1 \geq \frac{\sigma}{\lambda_i}\right] + s \cdot \mathbb{E}\left[x_i^1 \middle| (1-s)x_i^1 \geq \frac{\sigma}{\lambda_i}\right]$. Observe that the second term in the summation at the right side is a constant and does not relate to r_i. For the first term, we have

$$\begin{aligned}\mathbb{E}[l_i|(1-s)x_i^1 \geq \frac{\sigma}{\lambda_i}] &= Pr\{y_i \leq \frac{\sigma}{\lambda_i}\} \cdot \mathbb{E}[y_i|y_i \leq \frac{\sigma}{\lambda_i}] \\ &+ Pr\{y_i > \frac{\sigma}{\lambda_i}\} \cdot \mathbb{E}[l_i|l_i > \frac{\sigma}{\lambda_i}]\end{aligned} \tag{14}$$

Applying the equality that $\frac{1}{\lambda_i} = \frac{\mu_i \alpha}{\alpha - 1}$,

$$\begin{aligned}r_i \cdot \mathbb{E}\left[l_i \middle| (1-s)x_i^1 \geq \frac{\sigma}{\lambda_i}\right] &= \sigma r_i \mu_i \cdot (\frac{\alpha-1}{\sigma\alpha})^{\alpha(r_i-1)} \\ &\cdot \frac{\alpha^2}{1-\alpha} \frac{(\frac{\alpha-1}{\sigma\alpha})^{\alpha(r_i-1)}}{(\alpha r_i - 1)(\alpha r_i - \alpha - 1)} \\ &+ r_i \mu_i \cdot \frac{\alpha(r_i-1)}{\alpha r_i - \alpha - 1}\end{aligned} \tag{15}$$

Define $\tau_i(r_i) = r_i \cdot \mathbb{E}\left[l_i \middle| (1-s)x_i^1 \geq \frac{\sigma}{\lambda_i}\right]$, we derive the following lemma:

Lemma 2. *$\tau_i(r_i)$ is an increasing function of r_i when $r_i \in [2, \infty)$ for all $\alpha, \sigma > 1$. Moreover, we can choose an appropriate σ such that $\tau_i(2) < \tau_i(1)$.*

Proof. It holds that

$$\begin{aligned}\tau_i(r_i) &= r_i \cdot \mathbb{E}\left[l_i \middle| (1-s)x_i^1 \geq \frac{\sigma}{\lambda_i}\right] \\ &= \mu_i \cdot [\frac{\alpha r_i(r_i-1)}{\alpha r_i - \alpha - 1} - \frac{\alpha^2}{\alpha - 1} \frac{\sigma r_i}{(\alpha r_i - 1)(\alpha r_i - \alpha - 1)} \\ &\cdot (\frac{\alpha-1}{\sigma\alpha})^{\alpha(r_i-1)}]\end{aligned}$$

and $\frac{\alpha r_i(r_i-1)}{\alpha r_i - \alpha - 1}$ is an increasing function when $r_i \geq 2$. Moreover, $\frac{\sigma r_i}{(\alpha r_i - 1)(\alpha r_i - \alpha - 1)}$ and $\frac{\alpha-1}{\sigma\alpha})^{\alpha(r_i-1)}$ are both decreasing functions when $r_i \geq 2$. Thus $\tau_i(r_i)$ is an increasing function.

We continue to prove that $\tau_i(2) < \tau_i(1)$ for some σ.

$$\frac{\tau_i(2)}{\tau_i(1)} = \frac{2(\alpha-1)}{\sigma\alpha} - \frac{2}{(2\alpha-1)}(\frac{\alpha-1}{\sigma\alpha})^{\alpha}.$$

Obviously, just take $\sigma > 2$, we get $\frac{\tau_i(2)}{\tau_i(1)} < 1$. Thus, we complete the proof. □

Applying this lemma, we observe that the optimal value for r_i is 2 for all $1 \leq i \leq N$ which means only one speculative copy should be made once a straggler is detected. Substituting this result into Equation 13, the optimal value for σ is determined by

$$\sigma^* = \arg\min_{\sigma} \left\{\sigma^{-\alpha+1} \cdot [\frac{2(\alpha-1)}{\sigma\alpha} - \frac{2}{2\alpha-1} \cdot (\frac{\alpha-1}{\sigma\alpha})^{\alpha} - 1]\right\} \tag{16}$$

Hence, σ^* is the solution to the following equation:

$$(\frac{\alpha-1}{\alpha})^{\alpha} \cdot (1 + \frac{\alpha+1}{(2\alpha-1)(\alpha-1)})\sigma^{-\alpha-1} - \sigma^{-2} - 1 = 0 \tag{17}$$

An immediate result from Equation 17 is that the optimal value for r_i and σ do not relate to $\frac{1}{\lambda_i}$ and they depend on the order of the heavy-tail distribution only.

V. Algorithm design when the cluster is heavily loaded

In this case, $M < N$ and it indicates that the beginning the tasks of job J cannot be launched simultaneously in the cluster. As introduced in Section II A.2, Microsoft Mantri [13] chooses to kill and restart a new copy if $t_{rem} > \mathbb{E}[t_{new}] + \delta$ when the cluster is heavily loaded. We extend this scheme and propose the *Enhanced Speculative Execution* (ESE) algorithm. Explicitly, when a slot is idle and sends heartbeat to the Job Tracker, the Job Tracker will duplicate a task who satisfies $t_{rem} > \sigma \cdot \mathbb{E}[t_{new}]$ to launch in this slot. If multiple tasks satisfy this condition, the task with the longest remaining time will be launched. However, if no task satisfies this condition, the Job Tracker will launch a new task which has not been scheduled yet. The corresponding pseudocode is given in Algorithm 1.

Our objective is to find an appropriate σ in the ESE algorithm to minimize the job *flowtime*. However, analyzing the *flowtime* directly in the heavily loaded cluster is difficult. Instead, we choose to minimize the expected resource consumption for one particular task. In this way, the average duration of each task is also optimized.

Define the random variable R_i as the resource that task i consumes in the cluster. It's quite difficult to deal with the heavy-tail distribution and thus we consider to analyze the case where the duration of task i in job J is exponentially distributed with parameter λ_i as an approximation. For convenience, we take $c = 1$ and $\lambda_i = 1$. Thus, the expectation of

```
1  while slots are available do
2      if tasks are waiting for slots then
3          if there exists some running tasks who satisfy $t_{rem} > \sigma \cdot \mathbb{E}[t_{new}]$ then
4              choose a task with the longest remaining time to be launched in this slot.
5          else
6              start the waiting task that has the largest data to read.
7          end
8      else
9          choose to do backup follows an optimal solution in model 2.
10     end
11 end
12 return;
```

Algorithm 1: Enhanced Speculative Execution Algorithm

R_i can be expressed as

$$\mathbb{E}[R_i] = \mathbb{E}\left[x_i \middle| x_i > \frac{\sigma}{\lambda_i}\right] Pr\left\{x_i > \frac{\sigma}{\lambda_i}\right\} + \mathbb{E}\left[x_i \middle| x_i < \frac{\sigma}{\lambda_i}\right] Pr\left\{x_i < \frac{\sigma}{\lambda_i}\right\} \tag{18}$$

And

$$\mathbb{E}\left[x_i \middle| x_i < \frac{\sigma}{\lambda_i}\right] = \frac{1}{\lambda_i}\left(1 - \frac{\sigma e^{-\sigma}}{1 - e^{-\sigma}}\right) \tag{19}$$

$$Pr\{x_i < \frac{\sigma}{\lambda_i}\} = 1 - e^{-\sigma} \tag{20}$$

We define the notation of *asktime* which helps to derive the expression of $\mathbb{E}[R_i]$.

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.*

We assume that a task's *asktime* is uniformly distributed in its life time as the cluster is heavily loaded. Thus,

$$\mathbb{E}[x_i | x_i > \frac{\sigma}{\lambda_i}] Pr\{x_i > \frac{\sigma}{\lambda_i}\} = \int_{\frac{\sigma}{\lambda_i}}^{\infty} \lambda_i e^{-\lambda_i t} [\int_0^{t-\frac{\sigma}{\lambda_i}} \frac{1}{t}(x + 2 \cdot \mathbb{E}[min\{t - x, t_{new}\}]dx) + \frac{1}{t}\int_{t-\frac{\sigma}{\lambda_i}}^{t} t dx]dt \tag{21}$$

Further, we get

$$\mathbb{E}[min\{t - x, t_{new}\}] = \mathbb{E}[t_{new} | t_{new} < t - x] Pr\{t_{new} < t - x\} + (t - x) Pr\{t_{new} > t - x\} \tag{22}$$

$$\mathbb{E}[t_{new} | t_{new} < t - x] = \frac{1}{\lambda_i} - \frac{(t - x) e^{-\lambda_i (t-x)}}{1 - e^{-\lambda_i (t-x)}} \tag{23}$$

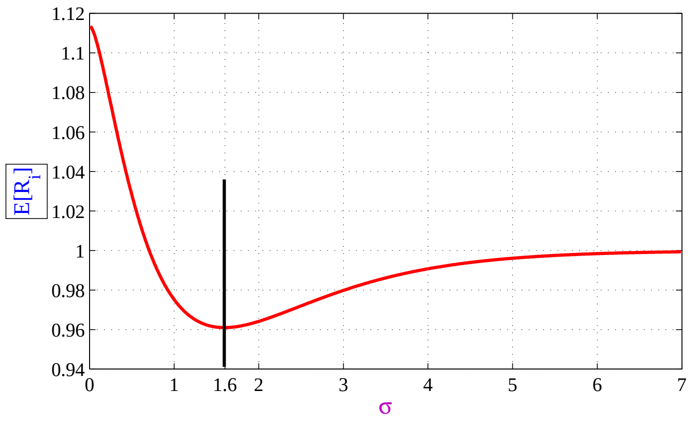

Fig. 2. Approximate analysis: the expected resource consumption for a single task under different σ with $\lambda_i = 1$.

Combining Equation 22 and 23, we obtain

$$\mathbb{E}[min\{t - x, t_{new}\}] = \frac{1 - e^{-\lambda_i (t-x)}}{\lambda_i} \tag{24}$$

Substituting Equation 19 20, 21 and 24 into 18,

$$\mathbb{E}[R_i] = \frac{1}{\lambda_i}\{\int_{\sigma}^{\infty} [2 - \frac{2\sigma}{t} + \frac{t}{2} + \frac{\sigma^2}{2t} + \frac{2}{t} \cdot (e^{-t} - e^{-\sigma})] e^{-t} dt\} + \frac{1}{\lambda_i} - \frac{\sigma + \sigma \cdot e^{-\sigma}}{\lambda_i} \tag{25}$$

The optimal value of σ that minimizes the expected resource consumption can be derived through letting the derivative of $\mathbb{E}[R_i]$ be 0. We illustrate the picture of $\mathbb{E}[R_i]$ in Fig. 2 under different σ for the exponential distribution. It indicates that when σ is close to 1.6, $\mathbb{E}[R_i]$ achieves the minimum value.

VI. Performance Evaluation

We have obtained the close-form solutions when the cluster is lightly loaded. To evaluate the performance of our proposed ESE algorithm, we run a simulation to show the resource consumption and job *flowtime* in the case where the cluster is heavily loaded.

A. Simulation of a Single job

In the simulation, there is only one job which arrives at time 0 and consists of $N = 10000$ tasks. The cluster has $M = 100$ worker nodes. In this situation, the job completion time is equivalent to *flowtime*. For convenience, we take $c = 1$. Moreover, we adopt two different distributions for the task duration in the following:

I. The duration of all the tasks follows the same Pareto distribution which is given as:

$$Pr\{x_i < x\} = \begin{cases} 1 - (\frac{1}{2x})^2 & for\ x \geq \frac{1}{2} \\ 0 & otherwise \end{cases}$$

II. The duration of all the tasks follows the same exponential distribution with λ.

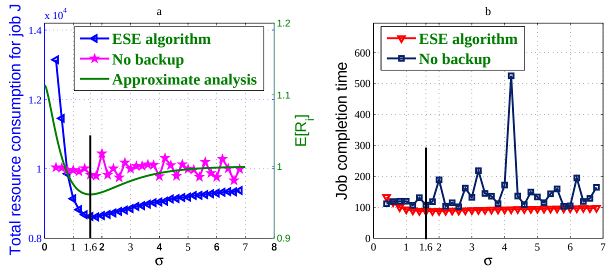

Fig. 3. Simulation result for heavy-tail distribution. Subfigure a shows the comparison of resource consumption under different σ between ESE algorithm and the naive method without backup. The blue line represents the ESE algorithm, the dark green line describes the result of the approximate analysis while the pink red line represents the naive method without backup. Subfigure b shows the comparison of job completion time.

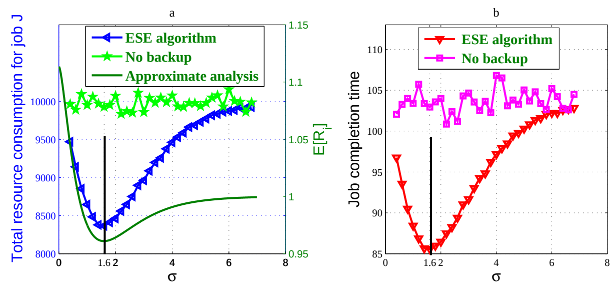

Fig. 4. Simulation result for exponential distribution. Panel a shows the comparison of resource consumption under different σ between ESE algorithm and the naive method without backup. Panel b shows the the comparison of job completion time.

For each $\sigma \in (0, 6)$ we run 50 simulations and take the average under two different distributions. We adopt the naive scheme in which speculative execution is not implemented as the comparison to the ESE algorithm. We capture two metrics which are the total resource consumption and the job completion time. The simulation results are shown in Figure 3 and 4. Figure 3 describes the resource consumption and job completion time under heavy-tail distribution while Figure 4 describes these two metrics under exponential distribution.

It shows that when σ is close to 1.6 for $\alpha = 2$, both the total amount of resource consumption and job completion time will achieve the minimum value for those two different distributions. Observe that the simulation results match well with the approximate analysis in Section V. Figure 3(b) indicates that the job completion time will be as long as 500 seconds in the worst case under heavy-tail distribution if speculative execution is not made. The reason behind is that some stragglers prolong the job completion time a lot which is also an important observation in [4]. However, with our proposed algorithm implemented, this metric will get a significant reduction by 50% when σ is chosen to be 1.6.

B. Performance evaluation for different α

To evaluate the impact of α on the job *flowtime* and resource consumption when the ESE algorithm is implemented, we test different α under the heavy-tail distribution for a single job.

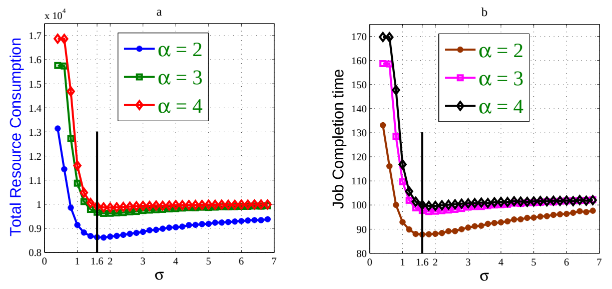

Fig. 5. Simulation result for different α under pareto distribution. Panel a shows the the resource consumption under different σ for $\alpha = 2, 3, 4$. Panel b shows the the job completion time.

We apply the pareto distribution and take $\alpha = 2, 3, 4$ while the mean of the task duration are all one. The other parameters are the same as the first simulation. The result is shown in Figure 5. It indicates that the optimal value for σ is close to 1.6 when $\alpha \in [2, 4]$. Moreover, as α increases, the performance improvement of the ESE algorithm degrades. When α is close to 4, there is no significant difference of the performance between the ESE algorithm and the naive scheduling scheme without backup strategy.

VII. Conclusion and Future Work

In this paper we address the resource management issue in the MapReduce Cluster and focus on how to put forward a good scheme for task speculative execution. The literature work only proposes several simple heuristic algorithms and we try to apply theoretical analysis to get a good solution. We present our work based on two different regimes. Precisely, we propose two different strategies when the cluster is lightly loaded and design the ESE algorithm while the cluster is heavily loaded. This is the first work so far to adopt the theoretical analysis for resource optimization in terms of task backup. Our main contributions are threefold:

- We build two models when the workload of the cluster is light and obtain two close results respectively.
- We extend form Microsoft's work and present the *Enhanced Speculative Execution Algorithm* for task backup when the cluster is heavily loaded. Moreover, we adopt the approximate analysis to find an optimal value for the parameter σ in the algorithm.
- We run several simulations under two different distributions and show that the speculative execution strategy in our ESE Algorithm makes a great improvement and the theoretical analysis matches well with it.

In the future work, we will use the trace data from the MapReduce Cluster to get a good measurement for the order of the heavy-tail distribution. Such measurement can help us to solve the optimization problem presented in Section III. Then we attempt to consider the data locality issue. As mentioned in the introduction, the data locality problem plays an important role for the cluster performance. If the duplicate of a running task is not launched in the *local machine*, the performance of speculative execution can degrade a lot. There also exist some tradeoffs between the data locality and task response time. We want to combine this issue with task backup strategy to achieve a more efficient resource management scheme in the MapReduce Cluster.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", in *USENIX OSDI*, 2004.

[2] Apache Hadoop. http://hadoop.apache.org.

[3] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *EuroSys* 2010, April 2010.

[4] 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*, pp. 3074-3082, March 2011.

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

[6] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters," tech. rep., University of California, Berkeley, April 2009.

[7] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlos, "On scheduling in map-reduce and flow-shops," in *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures*, SPAA, pp. 289-298, June 2011.

[8] Jian Tan, Xiaoqiao Meng, Li Zhang, "Delay Tails in MapReduce Scheduling," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, SIGMETRICS' 12, pages 5-16, New York, NY, USA, 2012.

[9] W. Wang, K. Zhu, Lei Ying, J. Tan and L. Zhang. "Map Task Scheduling in MapReduce with Data Locality: Throughput and Heavy-Traffic Optimality," in *Proceedings INFOCOM* 2013, Turin, Italy, April, 2013.

[10] Yousi Zheng, Ness Shroff, Prasun Sinha, "A New Analytical Technique for Designing Provably Efficient MapReduce Schedulers," in *Proceedings INFOCOM* 2013, Turin, Italy, April, 2013.

[11] Qiaomin Xie, Yi Lu, "Degree-guided map-reduce task assignment with data locality constraint," in *Information Theory Proceedings (ISIT)*, 2012 IEEE, pages 1-6, July 2012.

[12] 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 2007*, ser. EuroSys '07, 2007.

[13] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, E. Harris, "Reining in the Outliers in MapReduce Clusters using Mantri," in *USENIX OSDI*, Vancouver, Canada, Oct 2010.

[14] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proceeding of the 8th USENIX conference on Operating systems design and implementation*, ser. OSDI '08, 2008.

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

[16] Xiaoyu Sun ; Chen He ; Ying Lu, "ESAMR: An Enhanced Self-Adaptive MapReduce Scheduling Algorithm," in *the 18th International Conference on Parallel and Distributed Systems (ICPADS)*, 2012 IEEE.

[17] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica, "Effective Straggler Mitigation: Attack of the Clones," in *10th USENIX Symposium on Networked Systems Design and Implementation,* ser. NSDI '13, 2013.

[18] Minghua Chen, "Advanced Topics in Computer Networks: Lecture 5," in Fall 2013.

Appendix

A. Proof of Lemma Lemma 1

Define $l(r_i) = r_i \cdot \int_0^\infty (1 - F_i(t))^{r_i} dt$. We show that the second derivative of $l(r_i)$ is positive on the condition that $\ln(1 - F_i(t))$ is strictly convex.

$$\frac{d^2 l}{dr_i^2} = \int_0^\infty (2\ln(1-F_i(t)) + r_i(\ln(1-F_i(t)))^2)(1-F_i(t))^{r_i} dt$$

Let $h(t) = \ln(1 - F_i(t))^{r_i}$, we get

$$\begin{aligned} r_i \cdot \frac{d^2 l}{dr_i^2} &= \int_0^\infty (2h(t) + h^2(t)) \cdot e^{h(t)} dt \\ &= \int_0^\infty \frac{(2h(t) + h^2(t))}{h'(t)} d(e^{h(t)}) \\ &= \frac{h^2(t) + 2h(t)}{h'(t)} e^{h(t)} \Big|_0^\infty \\ &- \int_0^\infty (2h(t) + 2 - \frac{h''(t)(h^2(t) + 2h(t))}{(h'(t))^2}) e^{h(t)} dt \end{aligned} \tag{26}$$

As $F_i(t)$ is a CDF function with $F_i(0) = 0$ and $F_i(\infty) = 1$. It's straightforward to get

$$\frac{h^2(t) + 2h(t)}{h'(t)} e^{h(t)} \Big|_0^\infty = 0. \tag{27}$$

Similarly,

$$\begin{aligned} \int_0^\infty (h(t)+1) e^{(h(t))} dt &= \int_0^\infty \frac{(h(t)+1)}{h'(t)} d(e^{h(t)}) \\ &= \frac{h(t)+1}{h'(t)} e^{h(t)} \Big|_0^\infty - \int_0^\infty e^{h(t)} dt \\ &+ \int_0^\infty \frac{h''(t)(h(t)+1)}{(h'(t))^2} \cdot e^{h(t)} dt \end{aligned} \tag{28}$$

And,

$$\int_0^\infty e^{(h(t))} dt = \int_0^\infty \frac{d(e^{h(t)})}{h'(t)} = \frac{e^{h(t)}}{h'(t)}\Big|_0^\infty + \int_0^\infty \frac{h''(t)}{(h'(t))^2} e^{h(t)} dt \tag{29}$$

Thus,

$$\int_0^\infty (h(t)+1) e^{(h(t))} dt = \frac{h(t) e^{h(t)}}{h'(t)}\Big|_0^\infty + \int_0^\infty \frac{h''(t) \cdot h(t)}{(h'(t))^2} e^{h(t)} dt \tag{30}$$

Further, $\lim_{t\to\infty} \frac{h(t)e^{h(t)}}{h'(t)}\Big|_0^\infty = 0$, thus,

$$\frac{h(t)e^{h(t)}}{h'(t)}\Big|_0^\infty = 0.$$

Finally, substituting Equation (26),(27),(28),(29),(30) into $\frac{d^2 l}{dr_i^2}$ yields

$$r_i \cdot \frac{d^2 l}{dr_i^2} = \int_0^\infty \frac{h''(t) \cdot h^2(t)}{(h'(t))^2} e^{h(t)} dt.$$

On the condition that $\ln(1 - F_i(t)) = \frac{h(t)}{r_i}$ is a strictly convex function of t, $h''(t)$ is positive. This implies $\int_0^\infty \frac{h''(t) \cdot h^2(t)}{(h'(t))^2} e^{h(t)} dt > 0$. Thus, $\frac{d^2 l}{dr_i^2} > 0$ and $l(r_i)$ is a convex function of r_i.

Hence, we complete the proof.