Efficient Straggler Replication in Large-Scale Parallel Computing

DA WANG, Argus Investment Management
GAURI JOSHI, Carnegie Mellon University
GREGORY W. WORNELL, Massachusetts Institute of Technology

In a cloud computing job with many parallel tasks, the tasks on the slowest machines (straggling tasks) become the bottleneck in the job completion. Computing frameworks such as MapReduce and Spark tackle this by replicating the straggling tasks and waiting for any one copy to finish. Despite being adopted in practice, there is little analysis of how replication affects the latency and the cost of additional computing resources. In this article, we provide a framework to analyze this latency-cost tradeoff and find the best replication strategy by answering design questions, such as (1) when to replicate straggling tasks, (2) how many replicas to launch, and (3) whether to kill the original copy or not. Our analysis reveals that for certain execution time distributions, a small amount of task replication can drastically reduce both latency and the cost of computing resources. We also propose an algorithm to estimate the latency and cost based on the empirical distribution of task execution time. Evaluations using samples in the Google Cluster Trace suggest further latency and cost reduction compared to the existing replication strategy used in MapReduce.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems—*Reliability, availability, and serviceability*

General Terms: Performance, Reliability, Algorithms

Additional Key Words and Phrases: Straggler replication, task scheduling

ACM Reference format:
Da Wang, Gauri Joshi, and Gregory W. Wornell. 2019. Efficient Straggler Replication in Large-Scale Parallel Computing. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 4, 2, Article 7 (April 2019), 23 pages.
https://doi.org/10.1145/3310336

1 INTRODUCTION

In cloud computing, large-scale sharing of computing resources provides users with great flexibility and scalability. Computing frameworks such as MapReduce (Dean and Ghemawat 2008) and Apache Spark (Zaharia et al. 2010) are developed to harness these benefits. These frameworks

This work was supported in part by the NSF under grant CCF-1319828, the AFOSR under grant FA9550-11-1-0183, the Wellington and Irene Loh Fund Fellowship, the Schlumberger Foundation Faculty for the Future Fellowship, and the Claude E. Shannon Research Assistantship.
Authors' addresses: D. Wang, Argus Investment Management, 411 Fifth Avenue Suite 702, New York, NY 10016; email: dawang@alum.mit.edu; G. Joshi, Department of Electrical and Computer Engineering, Carnegie Mellon University, 4720 Forbes Avenue CIC 4105, Pittsburgh, PA 15213; email: gaurij@andrew.cmu.edu; G. W. Wornell, Signals, Information and Algorithms Laboratory, Massachusetts Institute of Technology, 77 Massachusetts Avenue Building 36-677, Cambridge, MA 02139; email: gww@mit.edu.

2376-3639/2019/04-ART7 $15.00
https://doi.org/10.1145/3310336

employ massive parallelization by dividing a large job into many tasks that can be executed in parallel on different machines. These frameworks can be used to run optimization and machine learning algorithms that can be easily divided into independent parallel tasks, such as alternating direction method of multipliers (ADMM) (Boyd et al. 2011) and Markov chain Monte Carlo (MCMC) (Neiswanger et al. 2013).

The execution time of a task on a machine is subject to stochastic variations due to co-hosting, virtualization, and other hardware and network variations (Dean and Barroso 2013). Thus, a key challenge in executing a job that consists of a large number of parallel tasks is the latency in waiting for the slowest tasks, or the "stragglers," to finish. As pointed out in Dean and Barroso (2013, Table 1), the latency of executing many parallel tasks could be significantly larger (140ms) than the median latency of a single task (1ms).

In this work, we provide a mathematical framework to analyze how replication of straggling tasks affects the latency and the cost of computing resources, and propose better scheduling policy designs.

1.1 Related Prior Work

The idea of replicating tasks in parallel computing has been recognized by system designers (Ghare and Leutenegger 2005) and was first adopted at a large scale via the "backup tasks" in MapReduce (Dean and Ghemawat 2008). A line of systems work (Ananthanarayanan et al. 2013; Chen et al. 2014; Ousterhout et al. 2013; Xu and Lau 2014; Zaharia et al. 2012) and references therein further developed this idea. For example, Apache Spark implements "speculative execution" to allow relaunching slow running tasks. Distributed machine learning systems also suffer for the problem of waiting for straggling learners to return gradients (Dean et al. 2012), and redundancy strategies have been shown to be effective in reducing latency (Chen et al. 2016; Dutta et al. 2018).

Although task replication has been studied in systems literature and also adopted in practice, there is not much work on mathematical analysis of replication strategies. Replication strategies are analyzed in Wang et al. (2014), mainly for the single task case. In this work, we consider task replication for a job consisting of a large number of tasks, which corresponds more closely to today's large-scale cloud computing frameworks. A shorter version of this work appeared in Wang et al. (2015).

The use of redundancy to reduce latency has also attracted attention in other contexts, such as cloud storage and networking (Gardner et al. 2015; Joshi et al. 2014, 2015; Shah et al. 2014; Sun et al. 2015; Vulimiri et al. 2013). Most of these works that consider queueing focus on the case of one task. Waiting for many tasks is harder to analyze, as indicated by fork-join queue analysis.

There is also an emerging body of work on using replication or erasure coding to mitigate stragglers in linear computations, such as matrix-vector multiplication (Dutta et al. 2016; Lee et al. 2016; Mallick et al. 2018) and matrix-matrix multiplication (Yang et al. 2017; Yu et al. 2017), and machine learning (Ferdinand and Draper 2016; Tandon et al. 2017). Our work is for general (possibly nonlinear) computations for which coding techniques cannot be directly applied, and we have to resort to simpler task replication strategies.

1.2 Our Contributions

In this work, we propose a framework to analyze strategies for replicating straggling tasks of a large computing job. In particular, we consider three parameters of a straggler replication strategy: (1) the fraction of tasks declared as stragglers, (2) number of replicas for each straggling tasks, and (3) whether the original copy should be killed or kept running. We characterize how these parameters impact the trade-off between latency and computing cost. Our characterizations allow us to identify regimes with the surprising property that replicating a small fraction of tasks drastically

reduces latency while saving computing cost. These insights allow one to apply optimization to search for scheduling policies based on one's sensitivity to computing latency and computing cost.

The rest of the article is organized as follows. In Section 2, we introduce notation, formulate the problem, and define performance metrics used in the article. In Section 3, we provide an analysis of single-fork task replication policies and defer all proofs to the Appendix. Then in Section 4, we describe an algorithm that finds a good scheduling policy for execution time distributions that are not analytically tractable (e.g., empirical distributions from real-world traces). In Section 5, we conclude with a discussion of the implications and future perspectives.

2 PROBLEM FORMULATION

2.1 Notation

Lowercase letters (e.g., x) denote a particular value of the corresponding random variable, which is denoted in uppercase letters (e.g., X). We denote the cumulative distribution function (c.d.f.) of X by $F_X(x)$. Its complement, the tail distribution, is denoted by $\bar{F}_X(x) \triangleq 1 - F_X(x)$. We denote the upper end point of F_X by

$$\omega(F_X) \triangleq \sup\{x : F_X(x) < 1\}. \tag{1}$$

For independent and identically distributed (i.i.d.) random variables $X_1, X_2, \ldots, X_n$, we define $X_{j:n}$ as the j-th order statistic (i.e., the j-th smallest of the n random variables).

2.2 System Model

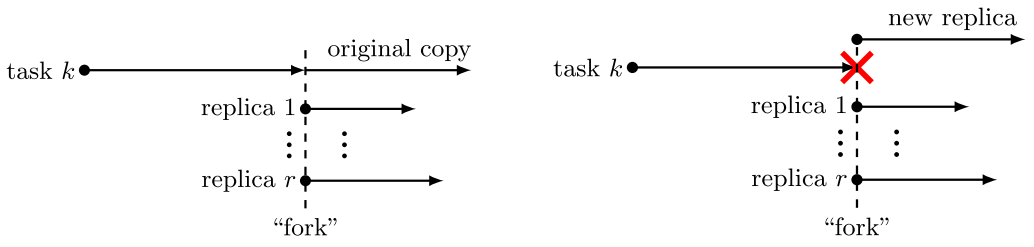
We consider a job consisting of n *parallel tasks*, where n is large[1] and each task is assigned to a different machine, and the job is said to be complete when all tasks are executed. Cloud service providers such as Amazon Web Services (AWS) typically allow users to rent a large number of servers, and thus it is possible to run a large number of parallel tasks. We do not consider queueing of tasks at these servers. Analyzing a system with queueing of parallel tasks requires a fork-join queueing analysis (Flatto and Hahn 1984; Joshi et al. 2012, 2014; Nelson and Tantawi 1988; Varki et al. 2008), which is known to be very hard even for the simple two server model.

Many systems works (Ananthanarayanan et al. 2010, 2013; Dean and Barroso 2013; Ousterhout et al. 2013) have observed that the same task can take drastically different execution time of different servers. The execution time of tasks run at a server can be random for two reasons: (1) variation in the server speed due to several factors such as virtualization, outages, and competing jobs, and (2) variation in the length of the tasks.

We use the probability distribution F_X to model the execution time due to the server and assume that this execution time distribution is i.i.d.across machines. The *identical* assumption of F_X implies that servers and tasks are homogeneous. In most embarrassingly parallel computations, the job is divided into tasks of equal size, and thus we assume that there is no variability in task lengths. The *independent* assumption of F_X could be satisfied when machine response times fluctuate independently over time, or when each new task (or new replica) is assigned to a new machine that is not previously used to run tasks of the current job. Note that we treat the variability that F_X captures as an exogenous factor from a user's perspective—in general, a user renting machines from a cloud computing service has little or no control over other jobs that share the resources.[2]

[1] Analysis of real-world trace data shows that it is common for a job to contain hundreds or even thousands of tasks (Reiss et al. 2012).

[2] A system designer may be able to influence this variability by adjusting the resource sharing among different jobs, which is another interesting direction that is beyond the scope of this work.



(a) Keep the original copy (π_{keep}) (b) Kill the original copy (π_{kill})

Fig. 1. Single-fork policy illustration.

2.3 Scheduling Policy

A *scheduling policy* or *scheduler* assigns one or more replicas of each task to different machines, possibly at different time instants. In this work, we assume the scheduler receives instantaneous feedback notifying it when a machine finishes its assigned task and there is *no intermediate feedback* indicating the status of processing of a task. We focus our attention on a set of policies called *single-fork policies*, defined as follows.

Definition 1 (Single-fork scheduling policy). A single-fork scheduling policy $\pi(p, r)$ launches all n tasks at time 0. It waits until $(1-p)n$ tasks finish. For each of the remaining pn straggling tasks, it chooses one of the following two actions:

- **replicate and keep the original copy ($\pi_{\text{keep}}(p, r)$):** launch r new replicas;
- **replicate and kill the original copy ($\pi_{\text{kill}}(p, r)$):** kill the original copy and launch $r + 1$ new replicas.

When the earliest replica of a task finishes, all the other remaining replicas of the same task are terminated.

Note that in both scenarios, there are a total of $r + 1$ replicas running after the forking point. Figure 1 illustrates these two cases of keeping or killing the original copy of a task. For simplicity of notation, we assume that p is such that pn is an integer. We note that $p = 0$ corresponds to running n tasks in parallel and waiting for all to finish, which is the baseline case without any replication or killing any original tasks.

Remark 1 (Backup tasks in MapReduce and Spark). The idea of "backup tasks" in Google's MapReduce (Dean and Ghemawat 2008) and "speculative execution" in Apache Spark (Zaharia et al. 2010) corresponds to a single-fork policy with $r = 1$ and π_{keep}. The value of p is tuned dynamically and hence not specified in Dean and Ghemawat (2008). The `spark.speculation.quantile` configuration in Apache Spark corresponds to p in the single-fork policy.

Although we focus on single-fork policies in this article, the analysis can be generalized to multi-fork policies, where new replicas of straggling tasks are launched at multiple times during the execution of the job (Wang 2014, Section 6.4). Forking multiple times can achieve a better latency-cost trade-off, but could be undesirable in practice due to additional delay and complexity in obtaining new and killing existing replicas.

2.4 Performance Metrics

We now define the latency and cost metrics used to compare straggler replication policies and understand when and how replication is useful.

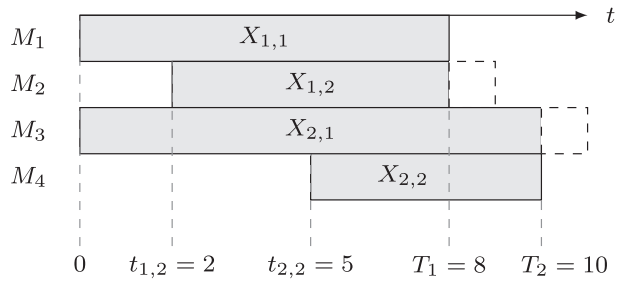


Fig. 2. Illustration of T and C for a job with two tasks, which are originally run on machines M_1 and M_3. Replicas of the tasks are added on machines M_2 and M_4 at times 2 and 5 respectively. The latency, or the time to complete the job, is $T = \max(8, 10) = 10$, and the computing cost is $C = (8 + 6 + 10 + 5)/2 = 14.5$.

Definition 2 (Expected Latency). Given a scheduling policy, the expected latency $\mathbb{E}[T]$ is the expected value of T, the time taken for at least one replica of each of the n tasks to finish. It can be expressed as

$$\mathbb{E}[T] = \mathbb{E}\left[\max_{i\in\{1,2,\ldots,n\}} T_i\right], \tag{2}$$

where T_i is the time when at least one replica of task i finishes. More specifically, suppose the scheduler launches r replicas of each of the n tasks at times $t_{i,j}$ for $j = 0, 1, 2, \ldots r$, then

$$T_i = \min_{0\le j\le r}(t_{i,j} + X_{i,j}), \tag{3}$$

where $X_{i,j}$ are i.i.d., drawn from the execution time distribution F_X.

Definition 3 (Expected Cost). The expected computing cost $\mathbb{E}[C]$ is the sum of the running times of all machines, normalized by n, the number of tasks in the job. The running time is the time from when the task is launched on a machine until it finishes or is killed by the scheduler. More specifically, suppose the scheduler launches r replicas of each of the n tasks at times $t_{i,j}$ for $j = 0, 1, 2, \ldots r$, then

$$C \triangleq \frac{1}{n}\sum_{i=1}^{n}\sum_{j=0}^{r}(T_i - t_{i,j})^+, \tag{4}$$

where T_i is given in (3) and $(x)^+ = \max(0, x)$.

Infrastructure as a Service (IaaS) providers such as AWS, Microsoft Azure, and Google Cloud Platform charge users by the time and the number of machines used. Then the money spent by a user to rent the machines is proportional to our cost metric $\mathbb{E}[C]$.

Figure 2 illustrates the execution of a job with two tasks, as well as evaluation of the corresponding latency T and cost C. Given two tasks, we launch two replicas of task 1 $t_{1,1} = 0$ and $t_{1,2} = 2$, and two replicas of task 2 at $t_{2,1} = 0$ and $t_{2,2} = 5$. The task execution times are $X_{1,1} = 8$, $X_{1,2} = 7$, $X_{2,1} = 11$, and $X_{2,2} = 5$. Machine M_1 finishes the task first at time $t = 8$, $T_1 = 8$ and the second replica running on M_2 is terminated before it finishes executing. Similarly, machine M_4 finishes task 2 at time $T_2 = 10$, and the replica running on M_3 is terminated. Thus, the latency of the job is $T = \max\{T_1, T_2\} = 10$. The cost is the sum of all running times normalized by n (i.e., $C = (8 + 6 + 10 + 5)/2 = 14.5$).

3 SINGLE-FORK POLICY ANALYSIS

In this section, we analyze the trade-off between the performance metrics $\mathbb{E}[T]$ and $\mathbb{E}[C]$ for the single-fork policy defined in Definition 1. The choice of the best single-fork policy depends on the tail of F_X, as we demonstrate for the shifted exponential and Pareto distributions. All proofs are deferred to the Appendix.

3.1 Performance Characterization

Theorem 1 (Single-fork Latency and Cost). *For a computing job with n tasks, and task execution time distribution F_X, the latency and cost metrics satisfies*

$$\mathbb{E}[T] = F_X^{-1}(1-p) + \mathbb{E}\left[T^{(2)}\right] + O(1/n), \tag{5}$$

$$\mathbb{E}[C] = \int_0^{1-p} F_X^{-1}(h)dh + pF_X^{-1}(1-p) + \mathbb{E}\left[C^{(2)}\right] + O(1/n), \tag{6}$$

where $T^{(2)}$ and $C^{(2)}$ are the latency and cost incurred after the forking point when we launch replicas of the straggling tasks. Their expected values are

$$\mathbb{E}\left[T^{(2)}\right] = \begin{cases} \mathbb{E}[Z_{pn:pn}] & \text{for } \pi_{\text{kill}}(p,r), \\ \mathbb{E}_{T^{(1)}}\left[\mathbb{E}\left[W_{pn:pn}|T^{(1)}\right]\right] & \text{for } \pi_{\text{keep}}(p,r), \end{cases} \tag{7}$$

$$\mathbb{E}\left[C^{(2)}\right] = \begin{cases} (r+1)p \cdot \mathbb{E}[Z] & \text{for } \pi_{\text{kill}}(p,r), \\ (r+1)p \cdot \mathbb{E}_{T^{(1)}}\left[\mathbb{E}\left[W|T^{(1)}\right]\right] & \text{for } \pi_{\text{keep}}(p,r), \end{cases} \tag{8}$$

where $Z = X_{1:r+1}$, $T^{(1)} = X_{(1-p)n:n}$ and $\bar{F}_{W|T^{(1)}=t_1}(w) = \frac{\bar{F}_X(w+t_1)}{\bar{F}_X(t_1)}\bar{F}_X(w)^r$. The behavior of $Z_{pn:pn}$ and $W_{pn:pn}$ for large n is given by the extreme value theorem (see Theorem 6).

The proof of Theorem 1 can be found in Appendix A.2. A key observation from Theorem 1 is that the execution time before forking, $F_X^{-1}(1-p)$, is a quantity *independent with respect to n* and monotonically non-increasing with p, whereas the latency after forking, $\mathbb{E}[T^{(2)}]$, is monotonically non-decreasing with pn. In certain regimes, increasing p (and with proper choice of r), the time reduction in the first stage outweighs the time increase in the second stage, reducing the overall execution latency.

Using Theorem 1, we can determine the single-fork policy parameters p and r that give the best latency-cost trade-off for a given execution time distribution F_X. To decide whether to kill or to keep the original copy of the straggling task, we are essentially comparing the additional time needed for the original copy to finish and the completion time for a new copy. In Lemma 1, we identify when keeping the original task is better than killing the original task and vice versa.

Lemma 1 (Kill or Keep Original Task). *For a given $0 < p \leq 1$, keeping the original task gives lower latency and cost than killing the original task if*

$$\frac{\Pr(X > x + t)}{\Pr(X > t)} \leq \Pr(X > x) \quad \text{for all } x, t \geq 0. \tag{9}$$

Conversely, if the inequality in (9) is reversed for all $x, t \geq 0$, then killing the original task is better.

The proof is given in Appendix A.2. The class of distributions satisfying (9) are called *new-longer-than-used* distributions (Kochar and Wiens 1987), and distributions that satisfy the reverse inequality are called *new-shorter-than-used* distributions. An example of a new-longer-than-used distribution is the shifted exponential distribution for which we analyze the latency-cost trade-off in Section 3.2.

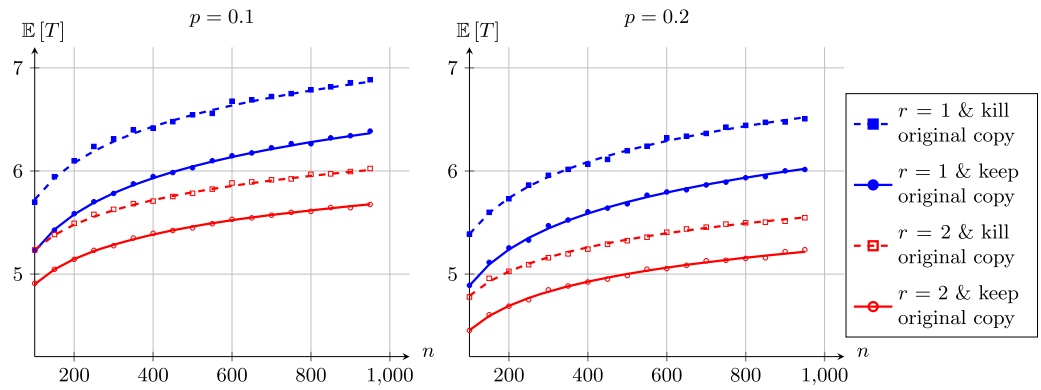


Fig. 3. Comparison of the expected latency $\mathbb{E}[T]$ obtained from simulation (points) and analytical calculations (lines) for the shifted exponential distribution ShiftedExp $(1, 1)$.

3.2 Single-Fork Scheduling With Analytical Execution Time Distributions

In this section, we evaluate the latency-cost trade-off in Theorem 1 for two execution time distributions: shifted exponential and Pareto. The shifted exponential distribution has an exponential tail, whereas Pareto distribution has a heavy tail.

3.2.1 Shifted Exponential Execution Time. Consider that the task execution time distribution F_X is a *shifted exponential distribution* ShiftedExp(Δ, μ). Its tail distribution function is given by

$$\Pr(X > x) = \begin{cases} e^{-\mu(x-\Delta)} & \text{for } x \geq \Delta, \\ 1 & \text{otherwise.} \end{cases} \tag{10}$$

The shifted exponential distribution has an exponentially decaying tail. It is lower bounded by a constant Δ, aiming to capture the delay due to machine start-up or task initialization. Due to this constant Δ, the shifted exponential distribution satisfies (9) for any $0 < p \leq 1$. Thus, it is always better to keep the original straggling task and launch additional replicas if necessary.

Theorem 2. *For a computing job with n tasks, if the execution time distribution of tasks are i.i.d. ShiftedExp(Δ, μ), then the latency and cost metrics satisfy*

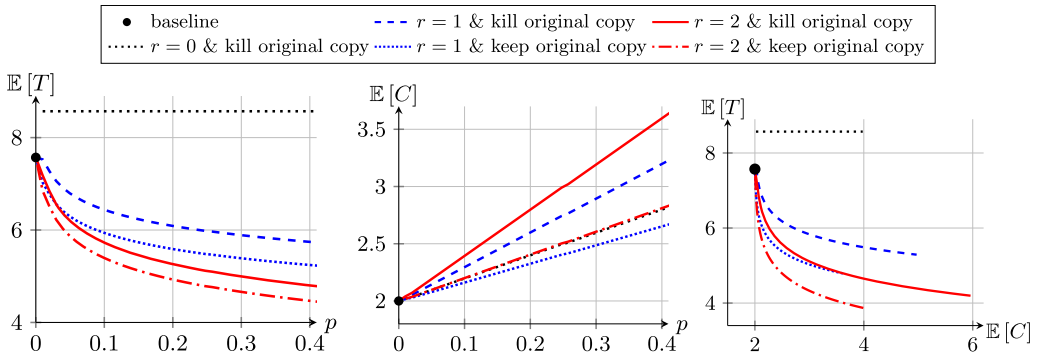
$$\mathbb{E}[T] = \begin{cases} \frac{2r+1}{r+1}\Delta + \frac{1}{(r+1)\mu}(\ln n - r\ln p + \gamma_{\text{EM}}) + O(1/n) & \text{for } \pi_{\text{keep}}(p, r) \\ 2\Delta + \frac{1}{(r+1)\mu}(\ln n - r\ln p + \gamma_{\text{EM}}) + O(1/n) & \text{for } \pi_{\text{kill}}(p, r) \end{cases}, \tag{11}$$

$$\mathbb{E}[C] = \begin{cases} \Delta + \frac{1}{\mu} + p\left[\Delta + r\frac{(1-e^{-\mu\Delta})}{\mu}\right] + O(1/n) & \text{for } \pi_{\text{keep}}(p, r) \\ \Delta + \frac{1}{\mu} + p(r+2)\Delta + O(1/n) & \text{for } \pi_{\text{kill}}(p, r) \end{cases}, \tag{12}$$

where γ_{EM} is the Euler-Mascheroni constant,

$$\gamma \triangleq \int_1^\infty \left(\frac{1}{\lfloor x \rfloor} - \frac{1}{x}\right) dx \approx 0.577. \tag{13}$$

The proof is given in Appendix A.3. Figure 3 compares the latency obtained from Monte Carlo simulation and analytical calculations for the shifted exponential distribution, indicating that the latency obtained from analytical calculation is very close to the simulated performance for $n \geq 100$, especially for the case with killing the original task. From Theorem 2, we observe that given r and whether we kill or keep the original task, replicating earlier (larger p) gives an $\Theta(\ln p)$ decrease in latency and a linear increase in the cost. This is also illustrated in Figure 4(a) and (b) for execution



(a) Expected latency $\mathbb{E}[T]$ (b) Expected cost $\mathbb{E}[C]$ (c) Trade-off between $\mathbb{E}[T]$ and $\mathbb{E}[C]$

Fig. 4. Characterization for ShiftedExp $(1, 1)$ and $n = 400$, by varying p in the range of $[0.05, 0.95]$.

time distribution ShiftedExp $(1, 1)$ and $n = 400$. Figure 4(c) illustrates the latency-cost trade-off. For the special case of $\Delta = 0$ by Theorem 2, the cost $\mathbb{E}[C] = 1/\mu$, which is independent of p and r. But latency always decreases with r and p. This suggests that we can achieve arbitrarily low latency without any increase in cost. However, in practice, the minimum time to complete a task is strictly positive—that is $\Delta > 0$.

3.2.2 Pareto Execution Time. The tail distribution function of the Pareto distribution Pareto(α, x_m) is

$$\Pr(X > x) \triangleq \begin{cases} \left(\frac{x_m}{x}\right)^{\alpha} & x \geq x_m, \\ 1 & \text{otherwise.} \end{cases} \tag{14}$$

The Pareto distribution has a heavy tail that decays polynomially. It has been observed to fit task execution time distributions in data centers (Dean and Barroso 2013; Reiss et al. 2012). The mean Pareto(α, x_m) is bounded only for $\alpha > 1$, and hence we focus on the cases where $\alpha > 1$.

Theorem 3. *For a computing job with n tasks, if the execution time distribution of tasks are i.i.d. Pareto(α, x_m), then for large n, the latency and cost metrics are*

$$\mathbb{E}[T] = x_m p^{-1/\alpha} + \Gamma\left(1 - \frac{1}{(r+1)\alpha}\right)\tilde{a}_{pn} + o(1), \tag{15}$$

$$\mathbb{E}[C] = \begin{cases} x_m \frac{\alpha}{\alpha-1} - x_m \frac{p^{1-1/\alpha}}{\alpha-1} + (r+1)p \int_0^{\infty} \left(\frac{x_m}{w}\right)^{\alpha r} \left(\frac{F_X^{-1}(1-p)}{w+F_X^{-1}(1-p)}\right)^{\alpha} dw + O(1/n) & \text{for } \pi_{\text{keep}}(p, r) \\ x_m \frac{\alpha}{\alpha-1} - x_m \frac{p^{1-1/\alpha}}{\alpha-1} + \frac{p x_m (r+1)^2 \alpha}{(r+1)\alpha - 1} + O(1/n) & \text{for } \pi_{\text{kill}}(p, r), \end{cases} \tag{16}$$

where the value of $\tilde{a}_{pn}$ depends on whether we choose to keep or kill the original task, and is given by

$$\tilde{a}_{pn} = \begin{cases} \mathbb{E}_{T^{(1)}}\left[\bar{F}^{-1}_{W|T^{(1)}}\left(\frac{1}{pn}\right) \middle| T^{(1)} = t_1\right] & \text{for } \pi_{\text{keep}}(p, r), \\ (pn)^{\frac{1}{(r+1)\alpha}} & \text{for } \pi_{\text{kill}}(p, r), \end{cases} \tag{17}$$

where $\bar{F}_{W|T^{(1)}=t_1}(w) = (\frac{x_m}{w})^{\alpha r}(\frac{t_1}{w+t_1})^{\alpha}$.

The proof is given in Appendix A.4. For the π_{keep} case, it is difficult to numerically evaluate the expectation over $T^{(1)}$ in (17). We evaluate an asymptotic lower bound on the expected latency for the π_{keep} case in Lemma 2.

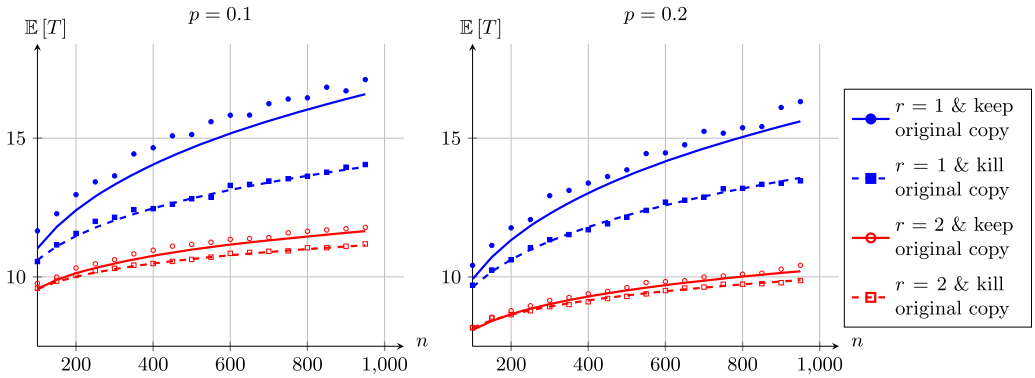
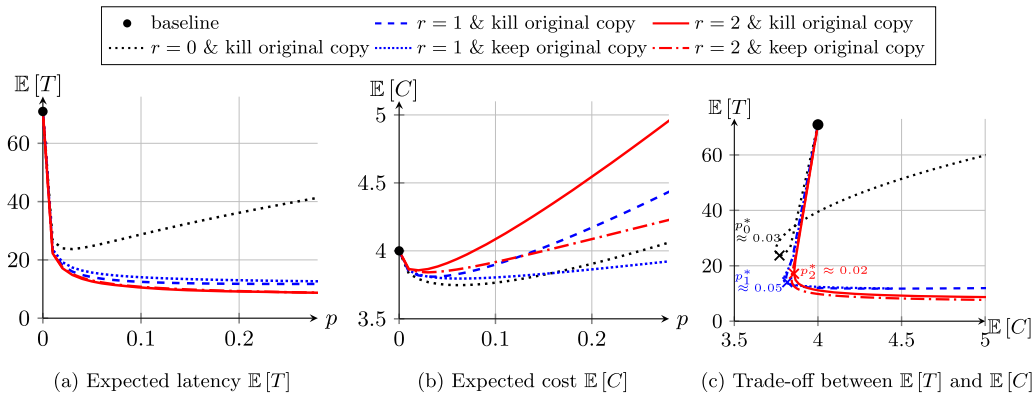


Fig. 5. Comparison of the expected latency $\mathbb{E}[T]$ obtained from simulation (points) and analytical calculations (lines) for the Pareto distribution Pareto $(2, 2)$. For the π_{keep} case, the analytical plot of $\mathbb{E}[T]$ is the asymptotic lower bound given by Lemma 2.



(a) Expected latency $\mathbb{E}[T]$ (b) Expected cost $\mathbb{E}[C]$ (c) Trade-off between $\mathbb{E}[T]$ and $\mathbb{E}[C]$

Fig. 6. Characterization for Pareto $(2, 2)$ and $n = 400$, by varying p in the range of $[0.05, 0.95]$.

Lemma 2 (Asymptotic Lower Bound on $\mathbb{E}[T]$ for the π_{keep} Case). *For the π_{keep} case, the expected latency can be lower bounded as follows for large n:*

$$\mathbb{E}[T] \geq x_m p^{-1/\alpha} + \Gamma\left(1 - \frac{1}{(r+1)\alpha}\right) \bar{F}^{-1}_{\tilde{W}}\left(\frac{1}{pn}\right) + o(1), \tag{18}$$

where $\bar{F}_{\tilde{W}}(w) = \left(\frac{x_m}{w}\right)^{\alpha r} \left(\frac{F_X^{-1}(1-p)}{w + F_X^{-1}(1-p)}\right)^{\alpha}$.

The proof follows from Fatou's lemma and is given in Appendix A.4. Similar to Figure 3, Figure 5 compares the latency obtained from simulation and analytical calculations for the Pareto distribution, which again demonstrates the effectiveness of the asymptotic analysis and bound given by Theorem 3 and Lemma 2.

In Figure 6(a) and (b), we plot the expected latency and cost as p varies, for different values of r. For the π_{keep} case, we plot the asymptotic lower bound on $\mathbb{E}[T]$ given by Lemma 2. The black dot is the baseline case ($p = 0$), where no replication is used and we simply wait for the original copies of all n tasks to finish. Note that $r = 0$ and keeping the original copy is also equivalent to the baseline case, and thus not plotted in the figures. The diminishing return of increasing r in terms of latency reduction is clearly demonstrated. In addition, we observe that a small amount of

replication (small p and r) can reduce latency significantly in comparison with the baseline case. But as p increases further, the latency may increase (as observed for $r = 0$) because of the second term in (5).

Intuition suggests that replicating earlier (larger p) and more (higher r) will increase the cost $\mathbb{E}[C]$. But Figure 6(a) and (b) show that this is not necessarily true. Since we kill replicas of task when one of its replicas finish, there could in fact be a saving in the computing cost. However, this benefit diminishes as p and r increase above a certain threshold.

Figure 6(c) shows the latency versus the computing cost for different values of r, with p varying along each curve. Depending upon the latency requirement and limit on the cost, one can choose an appropriate operating point on this trade-off curve. This plot again demonstrates the non-intuitive phenomenon that it is possible to reduce latency (from 70 to about 15 for $r = 1$ and $r = 2$ cases) and computing cost simultaneously.

4 EMPIRICAL EXECUTION TIME DISTRIBUTIONS

In practice, it may be difficult to fit the empirical behavior of the task execution time to a well-characterized distribution, thus making the latency-cost analysis using the framework presented in Section 3 difficult. In this section, we propose an algorithm to estimate the latency and cost from the empirical distribution of task execution time. This enables users to evaluate the latency-cost trade-off of various replication strategies using execution trace directly instead of a fitted execution time distribution. Applying our algorithm to the Google Cluster Trace data (Reiss et al. 2011), we show that it is possible to improve upon the performance of the default replication policy in MapReduce-style frameworks.

4.1 Latency and Cost Estimation

To estimate the latency and cost from empirical execution time samples, we apply the bootstrapping method (Efron and Tibshirani 1986) that uses the empirical distribution as an approximation of the true distribution. We present the algorithm for performance characterization in Algorithm 1.

By the central value theorem (see Theorem 4 in Appendix A.1), the standard deviation of the error in estimating $\mathbb{E}[C]$ and $\tilde{T}_1$, first term in $\mathbb{E}[T]$, converges to zero as $O(1/\sqrt{mn})$, where m is the number of times the sampling procedure is repeated. And generally $\tilde{T}_2$, the maximum order statistic term in $\mathbb{E}[T]$, converges to zero as $O(1/\sqrt{m})$. Thus, the estimation of $\tilde{C}$ is more robust than that of $\tilde{T}$. Nonetheless, with large enough m, we can make the estimation errors of both metrics small enough.

4.2 Demonstration Using Google Cluster Trace

The Google Cluster Trace data (Reiss et al. 2011) gives timestamps of events such as SCHEDULE, EVICT, FINISH, FAIL, and KILL for each of the tasks of computing jobs that are run on Google's cluster machines. In this section, we apply Algorithm 1 to two jobs in the Google Cluster Trace and study the latency-cost trade-offs for these real-world task service distributions.

In our demonstration, we only consider tasks with SCHEDULE and FINISH times, as we would like to obtain samples that represent a normal execution (not killed or evicted). In a few rare cases, a task is associated with multiple SCHEDULE and FINISH events due to duplicate execution. For these, we choose to keep the first occurrences in each event category.

We choose two jobs (Job IDs 6252284914 and 6252315810) with different numbers of tasks. For each task in a job, we obtain the task execution time by calculating the time difference between SCHEDULE and FINISH. The normalized histograms of the task execution times of the two jobs are shown in Figure 7(a) and Figure 7(b), respectively. Both the distributions have straggling tasks whose execution time is significantly longer than average. We then apply these execution time

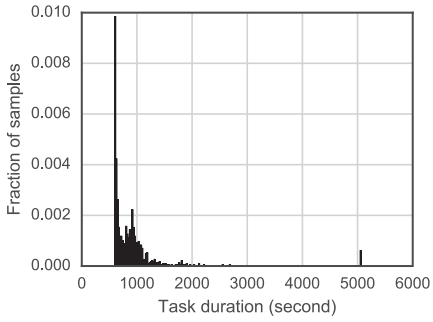
ALGORITHM 1: Latency and cost estimation

INPUT: $\mathbf{x} = [x_1, x_2, \ldots, x_N]$, N task execution duration samples (no replication, no original task killing)
Compute the empirical c.d.f. $\hat{F}_X(x)$ from $\mathbf{x}$
Computing the empirical c.d.f. $\hat{F}_Z(z)$ of $Z = X_{1:r+1}$
for $i = 1, 2, \ldots m$ **do**
  Draw n samples $\hat{\mathbf{x}} = [\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n]$ from $\hat{F}_X$
  Sort $\hat{\mathbf{x}}$ in ascending order: $[\hat{x}_{(1)}, \hat{x}_{(2)}, \ldots, \hat{x}_{(n)}]$
  $k \leftarrow n(1-p)$; $k' \leftarrow np$
  $\tilde{T}_1^{(i)} \leftarrow \hat{x}_{(k)}$ (the k-th smallest sample in $\hat{\mathbf{x}}$)
  $\tilde{C}_1^{(i)} \leftarrow \sum_{j=1}^{k} \hat{x}_{(j)}$
  if $\pi = \pi_{\text{kill}}$ **then**
    Draw k' samples $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{k'}]$ from $\hat{F}_Z(z)$
  else
    Compute empirical CDF of W as given in Theorem 1
    Draw k' samples $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{k'}]$ from $\hat{F}_W(w)$
  end if
  $\tilde{T}_2^{(i)} \leftarrow \max_{1 \le j \le k'} \hat{y}_j$
  $Y_{sum}^{(i)} \leftarrow \sum_{j=1}^{k'} \hat{y}_j$
  $\tilde{C}_2^{(i)} \leftarrow pn\tilde{T}_1^{(i)} + (r+1)Y_{sum}^{(i)}$
  $\tilde{T}^{(i)} \leftarrow \tilde{T}_1^{(i)} + \tilde{T}_2^{(i)}$
  $\tilde{C}^{(i)} \leftarrow \frac{1}{n}\left[\tilde{C}_1^{(i)} + \tilde{C}_2^{(i)}\right]$
end for
$\tilde{T} \leftarrow$ mean of $\tilde{T}^{(i)}$ for $i = 1, 2, \ldots m$
$\tilde{C} \leftarrow$ mean of $\tilde{C}^{(i)}$ for $i = 1, 2, \ldots m$
OUTPUT: $[\tilde{T}, \tilde{C}]$

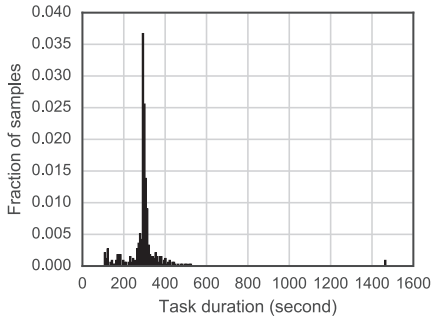
samples as inputs to Algorithm 1 with $m = 1{,}000$. By varying the value of r ($r \in \{1, 2, 3\}$) and p ($0 \le p \le 0.5$), we plot the $\mathbb{E}[T]$-$\mathbb{E}[C]$ trade-offs for all three jobs in Figures 8 and 9. Comparing the latency-cost performance of replication strategies for Job 1 and Job 2 provides helpful insights into when straggler replication helps and when it is too expensive.

For the two Google cluster jobs (Jobs 1 and 2), we observe that a small amount of replication (small p) reduces both $\mathbb{E}[T]$ significantly as compared to the baseline case ($p = 0$), demonstrating the effectiveness of replication for real-world execution time distributions. The decrease is sharper when we kill and relaunch the original copy (π_{kill}) as compared to keeping the original copy (π_{keep}). However, the expected cost with π_{kill} is larger than with π_{keep}. Similarly, adding more replicas (larger r) reduces latency but results in a higher expected cost per task.

Recall that the back-up tasks option in MapReduce uses $r = 1$ and keeps the original task. Figures 8 and 9 demonstrate that it may be more desirable to improve the performance trade-off by using more replicas, such as in Job 1, where a higher r could lead to lower latency $\mathbb{E}[T]$ with a slightly higher cost $\mathbb{E}[C]$. For Job 2, the trade-off improvement via using a higher r is less significant, as Figure 9 indicates. We conjecture that this is due to the tail in Figure 7(a) being heavier than that in Figure 7(b). Observe that for Job 1, the fraction of tasks that take longer than the most likely duration (~600 seconds) is larger (a significant fraction can take around ~1,000 seconds, in addition to the small fraction of tasks taking ~5,000 seconds). However, for Job 2, most jobs

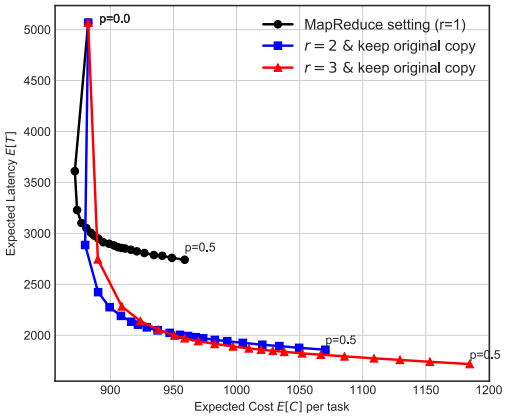


(a) Job 1: Google cluster Job 6252284914 (1026 tasks)

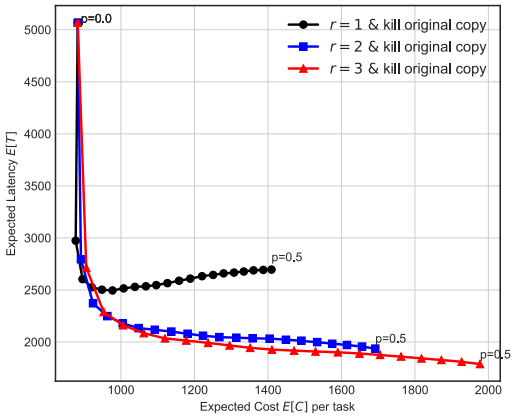


(b) Job 2: Google cluster Job 6252315810 (488 tasks)

Fig. 7. Normalized histogram of the task execution times.



(a) Trade-off with original copy kept π_{keep}



(b) Trade-off with original copy killed π_{kill}

Fig. 8. The $\mathbb{E}[T]$-$\mathbb{E}[C]$ trade-off for Job 1 (ID 6252284914) with 1,026 tasks. Each pair of adjacent dots corresponds to change in p by 0.025.

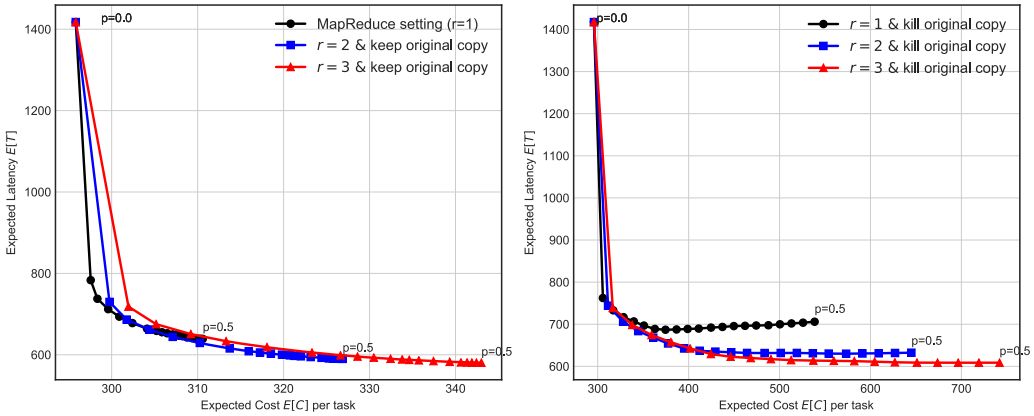
finish within 300 seconds, with only a small probability of straggling and taking ~1,450 seconds to finish.

4.3 Scheduling Policy Selection

With the trade-off between latency $\mathbb{E}[T]$ and computing cost $\mathbb{E}[C]$ provided in Algorithm 1, a user can formulate an optimization problem to choose the best scheduling policy based on one's sensitivity to latency and computing cost. In addition, one can incorporate additional constraints, such as $r_{\max}$, the maximum number of copies to replicate, due to the communication overhead of issuing and canceling tasks.

For example, a latency-sensitive user may choose to define the optimal scheduling policy via the following constrained optimization problem:

$$\begin{aligned} &\text{minimize} && \mathbb{E}[T(\pi)], \\ &\text{subject to} && \mathbb{E}[C(\pi)] \leq (1+\mu)\mathbb{E}[C(\pi_0)], \\ & && r \leq r_{\max}, \end{aligned} \tag{19}$$



(a) Trade-off with original copy kept π_{keep} (b) Trade-off with original copy killed π_{kill}

Fig. 9. The $\mathbb{E}[T]$-$\mathbb{E}[C]$ trade-off for Job 2 (ID 6252315810) with 488 tasks. Each pair of adjacent dots corresponds to change in p by 0.025.

Table 1. Scheduling Policy Obtained via Latency-Sensitive Optimization in (19) and Cost-Sensitive Optimization in (20)

	Baseline		Latency Sensitive With $\mu = 0.1$					Cost Sensitive With $\lambda = 5$				
Job	$\mathbb{E}[T]$	$\mathbb{E}[C]$	p^*	r^*	Keep/Kill	$\mathbb{E}[T]$	$\mathbb{E}[C]$	p^*	r^*	Keep/Kill	$\mathbb{E}[T]$	$\mathbb{E}[C]$
Job 1	5,068	882	0.177	3	Keep	1,939	970	0.142	2	Keep	2,111	920
Job 2	1,418	296	0.400	2	Keep	591	325	0.106	2	Keep	639	306

where π_0 is the baseline scheduling policy without replication, μ the allowed additional computing cost, and $r_{\max}$ the maximum allowed number of copies for a task. However, a cost-sensitive user may choose to define the optimal scheduling policy via the following optimization problem:

$$\begin{aligned} &\text{minimize} && \mathbb{E}[T(\pi)] + \lambda \mathbb{E}[C(\pi)], \\ &\text{subject to} && r \le r_{\max}, \end{aligned} \tag{20}$$

where λ indicates the relative importance of computing cost, because $\mathbb{E}[C]$ is approximately proportional to the cost of cloud computing instances. Although it is difficult to determine closed-form optimal solutions to (19) and (20), we observe that constrained optimization methods such as the constrained optimization by linear approximation (COBYLA) method (Powell 2007) are effective in searching for the optimal solution due to the low dimensionality of the search space. In Table 1, we present the scheduling policies obtained via these two different optimization formulations.

5 CONCLUDING REMARKS

5.1 Main Implications

Replication of the slowest tasks of a computing job (straggling tasks) has been observed to be highly effective in practice to speed up job completion. In this article, we provide a theoretical framework to understand the effect of straggler replication on the job completion latency and the additional computing time spent on running the replicas. Our latency-cost analysis gives the insight that the scaling of job completion latency with the number of tasks depends on the tail of the per-task execution time. We identify regimes where replicating a small fraction of stragglers

can drastically reduce latency and computing cost simultaneously. With the guidance from this asymptotic analysis, we propose a bootstrapping-based algorithm to estimate the latency and cost from empirical traces of execution time. The effectiveness of this algorithm is demonstrated on the Google Cluster Trace data, where we show that careful choice of the replication strategy can improve the latency-cost trade-off as compared to the default option in MapReduce.

5.2 Future Directions

Generalizations of this straggler replication model include considering heterogeneous servers, dependencies between tasks (some tasks need to complete to begin others), and taking into account queueing delay of tasks as considered in Gardner et al. (2015, 2016), Joshi (2016), and Joshi et al. (2015) for the single task case. Another direction is to analyze approximate computing, where we need only a subset of the tasks of a job to complete, which is a relevant model for information retrieval and machine learning jobs. This idea is developed in the context of coded distributed storage in Joshi et al. (2014) and Shah et al. (2014). We also aim to develop an algorithm that learns the task execution time distribution F_X online, and we use it to decide when and how many replicas to launch. This has an exploration-exploitation trade-off, similar to the multi-arm bandit problems studied in reinforcement learning (Sutton and Barto 1998).

More broadly, our analysis framework can be applied to other systems with stochastically varying components. For example, in crowdsourcing, each worker may take a variable amount of time to complete a task (Wang 2014).

A APPENDIX

A.1 Results From Order Statistics

In this section, we present results from order statistics that are used in our analysis.

Theorem 4 (Central Value Theorem (Theorem 10.3 in David and Nagaraja (2003))). *Given* $X_1, X_2, \ldots, X_n \overset{i.i.d.}{\sim} F_X$, *if* $0 < p < 1$ *and* $0 < f(x_p) < \infty$, *where* $x_p = F_X^{-1}(p)$, *then for* $k = np + o(\sqrt{n})$, *the k-th order statistic is asymptotically normal,*

$$\sqrt{n}(X_{k:n} - x_p) \xrightarrow{d} N\left(0, \frac{p(1-p)}{f^2(x_p)}\right),$$

where $f(\cdot)$ *is the p.d.f. that corresponds to* F_X *and* $\xrightarrow{d}$ *denotes convergence in distribution as* $n \to \infty$.

Extreme value theory (EVT) is an asymptotic theory of extremes (i.e., minima and maxima). It shows that if a distribution belongs to one of three families of distributions (specified in Theorem 5), then its maxima can be well characterized asymptotically by Theorem 6, which is also referred to as the Fisher-Tippett-Gnedenko theorem.

Theorem 5 (Domains of Attraction). *A distribution function* F_X *has one of the following domains of attraction if it satisfies the conditions of the extreme value distribution* $G(x)$ *if and only if*

(1) $F_X \in \mathrm{DA}(\Lambda)$ *if and only if there exists* $\eta(x) > 0$ *such that*

$$\lim_{x \to \omega(F)^-} \frac{\bar{F}(x + t\eta(x))}{\bar{F}(x)} = e^{-t},$$

(2) $F_X \in \mathrm{DA}(\Phi_\xi)$ *if and only if* $\omega(F) = \infty$ *and*

$$\lim_{x \to \infty} \frac{\bar{F}(tx)}{\bar{F}(x)} = t^{-\xi}, \quad t > 0,$$

(3) $F_X \in \mathrm{DA}(\Psi_\xi)$ *if and only if* $\omega(F) < \infty$ *and*

$$\lim_{x \to 0^+} \frac{\bar{F}(\omega(F) - tx)}{\bar{F}(\omega(F) - x)} = t^\xi, \quad t > 0,$$

where $\omega(F) = \sup\{x : F_X(x) < 1\}$*, the upper end point of the distribution* F_X*.*

Intuitively, $F \in \mathrm{DA}(\Lambda)$ corresponds to the case that $\bar{F}$ has an exponentially decaying tail, $F \in \mathrm{DA}(\Phi_\xi)$ corresponds to the case that $\bar{F}$ has heavy tail (e.g., polynomially decaying), and $F \in \mathrm{DA}(\Psi_\xi)$ corresponds to the case that $\bar{F}$ has a short tail with finite upper bound.

Theorem 6 (Extreme Value Theorem (Theorem 1.1.3 in de Haan and Ferreira (2006))). *Given* $X_1, \ldots, X_n \overset{i.i.d.}{\sim} F$*, if there exist sequences of constants* $a_n > 0$ *and* $b_n \in \mathbb{R}$ *such that*

$$\mathbb{P}[(X_{n:n} - b_n)/a_n \le x] \to G(x) \tag{21}$$

as $n \to \infty$ *and* $G(\cdot)$ *is a non-degenerate distribution. The extreme value distribution* $G(x)$ *and the values of* a_n *and* b_n *depend on the domain of attraction (and hence the tail behavior) of* F_X *given by Theorem 5.*

(1) *For* $F_X \in \mathrm{DA}(\Lambda)$*,*

$$a_n = \eta(F^{-1}(1 - 1/n)), \tag{22}$$
$$b_n = F^{-1}(1 - 1/n), \tag{23}$$
$$G(x) = \Lambda(x) = \exp\{-\exp(-x)\}, \tag{24}$$

where $\Lambda(x)$ *is called the* Gumbel *distribution.*

(2) *For* $F_X \in \mathrm{DA}(\Phi_\xi)$*,*

$$a_n = F^{-1}(1 - 1/n), \tag{25}$$
$$b_n = 0, \tag{26}$$
$$G(x) = \Phi_\xi(x) = \begin{cases} 0 & x \le 0 \\ \exp\{-x^{-\xi}\} & x > 0 \end{cases}. \tag{27}$$

where $\Phi_\xi(x)$ *is called the* Fréchet *distribution.*

(3) *For* $F_X \in \mathrm{DA}(\Psi_\xi)$*,*

$$a_n = \omega(F) - F^{-1}(1 - 1/n), \tag{28}$$
$$b_n = \omega(F), \tag{29}$$
$$G(x) = \Psi_\xi(x) = \begin{cases} \exp\{-(-x)^\xi\} & x < 0, \\ 1 & x \ge 0, \end{cases} \tag{30}$$

where $\Psi_\xi(x)$ *is called the* reversed-Weibull *distribution.*

Based on Theorem 6, we can derive the expected value of extreme values, as shown in Lemma 3.

Lemma 3 (Expected Extreme Values). *For* $F_X \in \mathrm{DA}(G)$*,*

$$\mathbb{E}[X_{n:n}] = a_n \mathbb{E}[G] + b_n + o(1), \tag{31}$$

where $G \in \{\Lambda, \Phi_\xi, \Psi_\xi\}$ *and*

$$\mathbb{E}[\Lambda] = \gamma_{\mathrm{EM}},$$
$$\mathbb{E}[\Phi_\xi] = \begin{cases} \Gamma(1 - 1/\xi) & \xi > 1 \\ +\infty & \text{otherwise}, \end{cases}$$
$$\mathbb{E}[\Psi_\xi] = -\Gamma(1 + 1/\xi),$$

where γ_{EM} is the Euler-Mascheroni constant and $\Gamma(\cdot)$ is the Gamma function—that is,

$$\Gamma(t) \triangleq \int_0^\infty x^{t-1}e^{-x}\,dx.$$

Proof. Theorem 6 shows that the extreme values of i.i.d. random variables converges *in distribution* to the extreme value distribution $G(x)$. Although, in general, convergence in distribution does not imply convergence in expectation, it holds for extreme values of i.i.d. random variables, as shown in Pickands (1968). Therefore, the expected value of extreme values equals the expectation of its corresponding extreme value distribution $\mathbb{E}[G]$ with an asymptotically varnishing error term $o(1)$, and $\mathbb{E}[G]$ can be computed straightforwardly. □

In addition to the expectation of extreme values, the central quantiles can be characterized by the following lemma, which is adapted from Equations (10.2.1) and (10.2.2) in David and Nagaraja (2003).

Lemma 4 (Expansion of Quantiles). *Given $X_1, X_2, \ldots, X_n \overset{i.i.d.}{\sim} F_X$, if F_X has finite mean and a bounded second derivative in a neighborhood of x_p with $f(x_p) > 0$, where $0 < p < 1$, $x_p = F_X^{-1}(p)$, then for any $k = np + O(1/n)$, the following approximation holds:*

$$X_{k:n} = x_p - \frac{\frac{1}{n}\sum_{i=1}^n B_i - p}{f(x_p)} + R_n,$$

where $\{B_i, i = 1, 2, \ldots, n\}$ are i.i.d. Bernoulli random variables with mean p, and almost surely

$$R_n = O\left(n^{-\frac{3}{4}}(\log n)^{\frac{1}{2}}(\log\log n)^{\frac{1}{4}}\right)$$

as $n \to \infty$.

Equation (10.2.6) in David and Nagaraja (2003) also provides a sharper bound on the expectation of quantiles, as shown in Lemma 5.

Lemma 5 (Expectation of Quantiles). *Given $X_1, X_2, \ldots, X_n \overset{i.i.d.}{\sim} F_X$, if F_X has finite mean and a bounded second derivative in a neighborhood of x_p with $f(x_p) > 0$, where $0 < p < 1$, $x_p = F_X^{-1}(p)$, then for any $k = np + O(1/n)$, the following approximation holds:*

$$\mathbb{E}[X_{k:n}] = x_p + O(1/n).$$

A.2 Proofs of Single-Fork Analysis

Proof of Theorem 1. The expected latency $\mathbb{E}[T]$ can be divided into two parts: before and after replication:

$$\mathbb{E}[T] = \mathbb{E}\left[T^{(1)}\right] + \mathbb{E}\left[T^{(2)}\right].$$

The time before forking $T^{(1)}$ is the time until $(1-p)n$ of the n tasks launched at time 0 finish. Thus, its expected value $\mathbb{E}[T^{(1)}]$ is the expectation of the $(1-p)n$-th order statistic $X_{(1-p)n:n}$ of n i.i.d. random variables with distribution F_X. By Lemma 4,

$$\begin{aligned}\mathbb{E}\left[T^{(1)}\right] &= \mathbb{E}\left[X_{(1-p)n:n}\right],\\ &= F_X^{-1}(1-p) + O\left(\frac{1}{n}\right).\end{aligned} \tag{32}$$

The expected time after forking is given by

$$\mathbb{E}\left[T^{(2)}\right] = \mathbb{E}_{T^{(1)}}\left[\mathbb{E}\left[\max(Y_1, Y_2, \ldots, Y_{pn})\middle| T^{(1)} = t_1\right]\right], \tag{33}$$

where Y_i is the residual time of the i-th straggling task given $T^{(1)} = t_1$. We find $\mathbb{E}[T^{(2)}]$ separately for the two cases: π_{kill} and π_{keep}.

For π_{kill}, where we kill the original copy of the straggling task, the residual execution times are independent of $T^{(1)}$, and they are the minimum of $r+1$ i.i.d. random variables with distribution F_X. We denote this by the random variable $Z = X_{1:r+1}$. Hence,

$$\mathbb{E}\left[T^{(2)}\right] = \mathbb{E}[Z_{pn:pn}] \quad \text{for } \pi_{\text{kill}}. \tag{34}$$

For π_{keep}, there is one original replica and r new replicas of each of the straggling tasks. Let the residual time be denoted by W. Its tail distribution is given by

$$\Pr(W > w) = \Pr(X_1 > w + t_1 | X_1 > t_1) \cdot \Pr(\min(X_2, \ldots X_{r+1}) > w), \tag{35}$$

$$\bar{F}_W(w) = \frac{\bar{F}_X(w+t_1)}{\bar{F}_X(t_1)} \bar{F}_X(w)^r. \tag{36}$$

Hence, the expected time after forking is given by

$$\mathbb{E}\left[T^{(2)}\right] = \mathbb{E}_{T^{(1)}}[\mathbb{E}[W_{pn:pn}]] \quad \text{for } \pi_{\text{keep}}(p,r). \tag{37}$$

Recall from Definition 3 that the expected cost $\mathbb{E}[C]$ is the sum of the running times of all machines, normalized by the number of tasks n. We can analyze $\mathbb{E}[C]$ by dividing it into sum of machine runtimes before and after forking:

$$\mathbb{E}[C] = \mathbb{E}\left[C^{(1)}\right] + \mathbb{E}\left[C^{(2)}\right], \tag{38}$$

$$\mathbb{E}[C^{(1)}] = \frac{1}{n} \sum_{i=1}^{(1-p)n} \mathbb{E}[X_{i:n}] + \frac{np}{n} \mathbb{E}\left[T^{(1)}\right] \tag{39}$$

$$= \frac{1}{n} \sum_{i=1}^{(1-p)n} \left(F_X^{-1}\left(\frac{i}{n}\right) + O\left(\frac{1}{n}\right)\right) + p\left(F_X^{-1}(1-p) + O\left(\frac{1}{n}\right)\right) \tag{40}$$

$$= \int_0^{1-p} F_X^{-1}(h)dh + pF_X^{-1}(1-p) + O\left(\frac{1}{n}\right), \tag{41}$$

$$\mathbb{E}\left[C^{(2)}\right] = \frac{1}{n} \sum_{j=1}^{pn} (r+1)\mathbb{E}[Y_j] \tag{42}$$

$$= \begin{cases} (r+1)p \cdot \mathbb{E}[Z] & \text{for } \pi_{\text{kill}}(p,r), \\ (r+1)p \cdot \mathbb{E}_{T^{(1)}}\left[\mathbb{E}\left[W \middle| T^{(1)} = t_1\right]\right] & \text{for } \pi_{\text{keep}}(p,r). \end{cases} \tag{43}$$

The cost before forking $\mathbb{E}[C^{(1)}]$ consists of the cost for the $(1-p)n$ tasks that finish first, plus the cost for the pn straggling tasks. The first term in (39) is the sum of the expected values of the smallest $(1-p)n$ execution times. Lemma 5 indicates that the i-th term in the summation is asymptotically $F_X^{-1}(i/n)$, resulting the first term in (40). The analysis of Riemann sum indicates it converges to the integral in the first term of (41) with error term $O(1/n)$. The second term in (39) is the normalized running time of the pn straggling tasks before forking. Similarly, we apply Lemma 5 to $\mathbb{E}[T^{(1)}]$ to arrive at the second term in (40).

The cost after forking, $\mathbb{E}[C^{(2)}]$, is the normalized sum of the runtimes of the $r+1$ replicas of each of the pn straggling tasks. The residual execution time of the j-th straggling task is Y_j. For π_{kill}, it is $Z = X_{1:r+1}$, and for π_{keep}, it is W, whose tail distribution is given by (36). Since the scheduler kills all replicas as soon as one replica finishes, the expected runtime for the j^{th} straggling task is $(r+1)\mathbb{E}[Y_j]$. Thus, the cost in (42) is the sum of $(r+1)\mathbb{E}[Y_j]$ over the pn tasks, normalized by n, which can be reduced to (43). □

Proof of Lemma 1. When we keep the original copy, the residual execution time of a straggling task is

$$W = \min\left\{X_{1:r}, \left(X\middle|X > T^{(1)}\right)\right\}, \tag{44}$$

$$\Pr(W > x) = \Pr(X > x)^r \frac{\Pr(X > x + T^{(1)})}{\Pr(X > T^{(1)})}, \tag{45}$$

where $\mathbb{P}[X > x + T^{(1)} \,||\, X > T^{(1)}]$ is the additional time needed for the original copy to finish after forking time $T^{(1)}$. When we kill the original copy, $r + 1$ new copies of the straggling task are launched at the forking point. Thus, the residual execution time is

$$Z = \min\{X_{1:r}, X\}, \tag{46}$$

$$\Pr(Z > x) = \Pr(X > x)^{r+1}. \tag{47}$$

Keeping the original task is better than killing it if Z stochastically dominates W—that is, $\Pr(W > x) \leq \Pr(Z > x)$ for all x. This is true if the condition (9) is satisfied. Conversely, killing the original task is better when the reverse condition holds. □

A.3 Analysis for Shifted Exponential Execution Time

Proof of Theorem 2. By Theorem 1,

$$\mathbb{E}[T] = F_X^{-1}(1-p) + O\left(\frac{1}{n}\right) + \mathbb{E}\left[T^{(2)}\right] \tag{48}$$

$$= \Delta - \frac{1}{\mu}\ln p + O\left(\frac{1}{n}\right) + \mathbb{E}\left[T^{(2)}\right] \tag{49}$$

$$\mathbb{E}[C] = \int_0^{1-p} F_X^{-1}(h)dh + pF_X^{-1}(1-p) + \mathbb{E}\left[C^{(2)}\right] + O(1/n) \tag{50}$$

$$= \int_0^{1-p} \left(\Delta - \frac{1}{\mu}\ln(1-h)\right)dh + p\left(\Delta - \frac{1}{\mu}\ln p\right) + \mathbb{E}\left[C^{(2)}\right] + O(1/n) \tag{51}$$

$$= \Delta(1+p) + \frac{1-p}{\mu} + O(1/n) + \mathbb{E}\left[C^{(2)}\right]. \tag{52}$$

To find $\mathbb{E}[T^{(2)}]$ and $\mathbb{E}[C^{(2)}]$, we consider the cases of killing and keeping the original task separately.

Case 1: Killing the original task (π_{kill}). The residual execution time of each straggling task after killing and relaunching the original task, and launching r additional replicas, is

$$Z = \min\{X_1, X_2, \ldots X_{r+1}\} \sim \text{ShiftedExp}(\Delta, (r+1)\mu). \tag{53}$$

Based on Theorem 5, for $\eta(z) = 1/((r+1)\mu)$, we have

$$\lim_{z\to\omega(F_Z)} \frac{\bar{F}_Z(z + u\eta(z))}{\bar{F}_Z(z)} = e^{-u}. \tag{54}$$

By Theorem 6 and Theorem 5, the maximum of shifted exponential belongs to the Gumbel family with

$$a_{pn} = \frac{1}{\mu(1+r)}$$

$$b_{pn} = \bar{F}_Z^{-1}(1/n) = \Delta + \frac{\ln(pn)}{\mu(r+1)}.$$

Thus, the expected latency after forking is given by

$$\mathbb{E}\left[T^{(2)}\right] = \tilde{a}_{pn}\mathbb{E}[\Lambda] + \tilde{b}_{pn} \tag{55}$$

$$= \frac{1}{\mu(1+r)}\gamma_{\text{EM}} + \Delta + \frac{\ln(pn)}{\mu(r+1)} \tag{56}$$

$$\mathbb{E}\left[C^{(2)}\right] = (r+1)p \cdot \mathbb{E}[Z] \tag{57}$$

$$= (r+1)p\left(\Delta + \frac{1}{(r+1)\mu}\right). \tag{58}$$

Case 2: Keeping the original task (π_{keep}). Due to the memoryless property of the tail of the shifted exponential distribution, the residual execution time W is independent of $T^{(1)}$ and is given by

$$W = \min\{\text{Exp}(\mu), \Delta + \text{Exp}(r\mu)\}.$$

The first term does not include Δ because for large n, the original task would have run for at least Δ seconds. Thus, the tail distribution of W is given by

$$\bar{F}_W(w) = \begin{cases} e^{-\mu w} & 0 < w < \Delta, \\ e^{\mu r\Delta}e^{-\mu(r+1)w} & w \geq \Delta. \end{cases} \tag{59}$$

By Theorem 6 and Theorem 5, the maximum of W's belongs to the Gumbel family with

$$a_{pn} = \frac{1}{\mu(1+r)},$$

$$b_{pn} = \bar{F}_W^{-1}(1/n) = \frac{r}{r+1}\Delta + \frac{\ln(pn)}{\mu(r+1)}.$$

Thus, we have

$$\mathbb{E}\left[T^{(2)}\right] = \tilde{a}_{pn}\mathbb{E}[\Lambda] + \tilde{b}_{pn} \tag{60}$$

$$= \frac{1}{\mu(1+r)}\gamma_{\text{EM}} + \frac{r}{r+1}\Delta + \frac{\ln(pn)}{\mu(r+1)} \tag{61}$$

$$\mathbb{E}\left[C^{(2)}\right] = (r+1)p \cdot \mathbb{E}[W] \tag{62}$$

$$= (r+1)p\left(\int_0^{\Delta} e^{-\mu w}dw + \int_{\Delta}^{\infty} e^{\mu r\Delta}e^{-\mu(r+1)w}\right), \tag{63}$$

$$= \frac{p(r+1)(1-e^{-\mu\Delta})}{\mu} + \frac{pe^{-\mu\Delta}}{\mu}. \tag{64}$$

Substituting these in (49) and (52), we get the result. □

A.4 Analysis for Pareto Execution Time

Proof of Theorem 3. From Theorem 1, we have

$$\mathbb{E}[T] = F_X^{-1}(1-p) + \mathbb{E}\left[T^{(2)}\right] + O(1/n) \tag{65}$$

$$= x_m p^{-1/\alpha} + \mathbb{E}\left[T^{(2)}\right] + O(1/n) \tag{66}$$

$$\mathbb{E}[C] = \int_0^{1-p} F_X^{-1}(h)dh + pF_X^{-1}(1-p) + \mathbb{E}\left[C^{(2)}\right] + O(1/n) \tag{67}$$

$$= x_m \int_0^{1-p} (1-h)^{-1/\alpha}dh + px_m p^{-1/\alpha} + \mathbb{E}\left[C^{(2)}\right] + O(1/n) \tag{68}$$

$$= x_m \frac{\alpha}{\alpha-1}[1-p^{1-1/\alpha}] + x_m p^{1-1/\alpha} + \mathbb{E}\left[C^{(2)}\right] + O(1/n) \tag{69}$$

$$= x_m \frac{\alpha}{\alpha-1} - x_m \frac{p^{1-1/\alpha}}{\alpha-1} + \mathbb{E}\left[C^{(2)}\right] + O(1/n). \tag{70}$$

To obtain (66), we use the tail distribution of Pareto given in (14). To derive the expected cost (70), we substitute $F_X^{-1}(h) = x_m(1-h)^{-1/\alpha}$ in the first and second terms in (67) and simplify the expression. Next we find $\mathbb{E}[T^{(2)}]$ and $\mathbb{E}[C^{(2)}]$ separately for the cases of killing the original task (π_{kill}) and keeping the original task (π_{keep}).

Case 1: Killing the original task (π_{kill}). For a single-fork policy that kills the original task, the scheduler waits for $(1-p)n$ tasks to finish and then relaunches each of the pn straggler tasks on a new machine, and also launches r additional replicas per task. Thus, the residual execution time of each straggling task is

$$\begin{aligned} Z &= \min(X_1, X_2, \ldots X_{r+1}), \\ Z &\sim \text{Pareto}((r+1)\alpha, x_m). \end{aligned} \tag{71}$$

From Theorem 6, we can show that $F_Z \in \text{DA}(\Phi_{(r+1)\alpha})$. And from (25), we can evaluate $\tilde{a}_{pn}$ as follows

$$\tilde{a}_{pn} = F_Z^{-1}\left(1 - \frac{1}{pn}\right) = x_m (pn)^{1/(r+1)\alpha}.$$

Substituting this in from Lemma 3, we have

$$\mathbb{E}\left[T^{(2)}\right] = \mathbb{E}[Z_{pn:pn}] = \Gamma\left(1 - \frac{1}{(r+1)\alpha}\right) x_m (pn)^{1/(r+1)\alpha} + o(1). \tag{72}$$

And $\mathbb{E}[C^{(2)}]$ can be evaluated as

$$\mathbb{E}\left[C^{(2)}\right] = (r+1)p \cdot \mathbb{E}[Z] = (r+1)p \cdot \frac{x_m (r+1)\alpha}{(r+1)\alpha - 1}. \tag{73}$$

Case 2: Keeping the original task (π_{keep}). For a single-fork policy that keeps the original task, the scheduler keeps the original copy and adds r additional replicas for each straggling task. Thus, the residual execution time can be expressed as

$$W = \min(\text{Pareto}(\alpha, t_1) - t_1, \text{Pareto}(r\alpha, x_m)), \tag{74}$$

$$\bar{F}_{W|T^{(1)}=t_1}(w) = \left(\frac{x_m}{w}\right)^{\alpha r} \left(\frac{t_1}{w+t_1}\right)^{\alpha}. \tag{75}$$

We can show that $F_W \in \text{DA}(\Phi_{(r+1)\alpha})$. From (25) in Theorem 6, $\tilde{a}_{pn} = \bar{F}^{-1}_{W|T^{(1)}}(\frac{1}{pn})$. Substituting this into (7), we can show that the expected value of $T^{(2)}$ is

$$\mathbb{E}\left[T^{(2)}\right] = \mathbb{E}_{T^{(1)}}\left[a_{pn} \cdot \Gamma\left(1 - \frac{1}{(r+1)\alpha}\right) \middle| T^{(1)} = t_1\right] + o(1), \tag{76}$$

$$= \mathbb{E}_{T^{(1)}}\left[\bar{F}^{-1}_{W|T^{(1)}}\left(\frac{1}{pn}\right) \middle| T^{(1)} = t_1\right] \cdot \Gamma\left(1 - \frac{1}{(r+1)\alpha}\right) + o(1), \tag{77}$$

where $T^{(1)} = X_{(1-p)n:n}$. And $\mathbb{E}[C^{(2)}]$ is given by

$$\mathbb{E}\left[C^{(2)}\right] = (r+1)p \cdot \mathbb{E}_{T^{(1)}}\left[\mathbb{E}\left[W|T^{(1)} = t_1\right]\right] \tag{78}$$

$$= (r+1)p \cdot \mathbb{E}_{T^{(1)}}\left[\int_0^\infty \left(\frac{x_m}{w}\right)^{\alpha r}\left(\frac{t_1}{w+t_1}\right)^{\alpha} dw\right] \tag{79}$$

$$= (r+1)p \cdot \int_0^\infty \left(\frac{x_m}{w}\right)^{\alpha r}\left(\frac{F_X^{-1}(1-p)}{w+F_X^{-1}(1-p)}\right)^{\alpha} dw, \tag{80}$$

where (80) follows from Theorem 4 and the monotone convergence theorem. Substituting $\mathbb{E}[T^{(2)}]$ and $\mathbb{E}[C^{(2)}]$ into (66) and (70) yields the result. □

Proof of Lemma 2. As given by (77) shown earlier, the expected latency after forking for π_{keep} and large n,

$$\mathbb{E}\left[T^{(2)}\right] = \mathbb{E}_{T^{(1)}}\left[\bar{F}^{-1}_{W|T^{(1)}}\left(\frac{1}{pn}\right)\middle| T^{(1)} = t_1\right] \cdot \Gamma\left(1 - \frac{1}{(r+1)\alpha}\right) + o(1). \tag{81}$$

Since $\bar{F}^{-1}_{W|T^{(1)}}(\cdot)$ is always non-negative, we can apply Fatou's lemma to show that

$$\liminf_{n\to\infty} \mathbb{E}_{T^{(1)}}\left[\bar{F}^{-1}_{W|T^{(1)}}\left(\frac{1}{pn}\right)\middle| T^{(1)} = t_1\right] \geq \mathbb{E}_{T^{(1)}}\left[\liminf_{n\to\infty} \bar{F}^{-1}_{W|T^{(1)}}\left(\frac{1}{pn}\right)\middle| T^{(1)} = t_1\right] \tag{82}$$

$$= \bar{F}^{-1}_{\tilde{W}}\left(\frac{1}{pn}\right). \tag{83}$$

Since the limits exist on both sides, we can replace lim inf by lim in (82). For large n, $T^{(1)}$ concentrates around $F_X^{-1}(1-p)$, and thus we can replace $T^{(1)}$ by $F_X^{-1}(1-p)$ on the right-hand side of (82). By substituting (83) into (81), we obtain the result. □

ACKNOWLEDGMENTS

We thank Devavrat Shah for helpful discussions.

REFERENCES

Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. 2013. Effective straggler mitigation: Attack of the clones. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*. 185–198.

G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. 2010. Reining in the outliers in map-reduce clusters using Mantri. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*. 1–16.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning* 3, 1 (2011), 1–122.

Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Józefowicz. 2016. Revisiting distributed synchronous SGD. arXiv:1604.00981. http://arxiv.org/abs/1604.00981

Qi Chen, Cheng Liu, and Zhen Xiao. 2014. Improving MapReduce performance using smart speculative execution strategy. *IEEE Transactions on Computers* 63, 4 (April 2014), 954–967. http://dx.doi.org/10.1109/TC.2013.15

H. A. David and H. N. Nagaraja. 2003. *Order Statistics*. John Wiley, Hoboken, NJ.

L. de Haan and A. Ferreira. 2006. *Extreme Value Theory: An Introduction*. Springer, New York.

Jeffrey Dean and Luis Barroso. 2013. The tail at scale. *Communications of the ACM* 56, 2 (2013), 74–80.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'Aurelio Ranzato, et al. 2012. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25*. 1223–1231.

Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified data processing on large clusters. *ACM Communications Magazine* 51, 1 (Jan. 2008), 107–113.

Sanghamitra Dutta, Viveck Cadambe, and Pulkit Grover. 2016. Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Advances in Neural Information Processing Systems*. 2100–2108. http://papers.nips.cc/paper/6329-short-dot-computing-large-linear-transforms-distributedly-using-coded-short-dot-products.pdf.

Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. 2018. Slow and stale gradients can win the rate: Error-runtime trade-offs in distributed SGD. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'18)*.

B. Efron and R. Tibshirani. 1986. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical Science* 1, 1 (1986), 54–75.

N. S. Ferdinand and S. C. Draper. 2016. Anytime coding for distributed computation. In *Proceedings of the Allerton Conference on Communication, Control, and Computing*. 954–960.

L. Flatto and S. Hahn. 1984. Two parallel queues created by arrivals with two demands I. *SIAM Journal on Applied Mathematics* 44, 5 (1984), 1041–1053.

K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf. 2015. Reducing latency via redundant requests: Exact analysis. In *Proceedings of the ACM SIGMETRICS Conference*.

K. Gardner, S. Zbarsky, M. Velednitsky, M. Harchol-Balter, and A. Scheller-Wolf. 2016. Understanding response time in the redundancy-d system. In *Proceedings of the Workshop on Mathematical Performance Modeling and Analysis*.

G. Ghare and S. T. Leutenegger. 2005. Improving speedup and response times by replicating parallel programs on a SNOW. In *Proceedings of the International Conference on Job Scheduling Strategies for Parallel Processing*. 264–287.

G. Joshi. 2016. *Efficient Redundancy Techniques to Reduce Delay in Cloud Systems*. Ph.D. Dissertation. Massachusetts Institute of Technology, Cambridge, MA.

G. Joshi, Y. Liu, and E. Soljanin. 2012. Coding for fast content download. In *Proceedings of the Allerton Conference on Communication, Control, and Computing*. 326–333.

G. Joshi, Y. Liu, and E. Soljanin. 2014. On the delay-storage trade-off in content download from coded distributed storage systems. *IEEE Journal on Selected Areas in Communications* 32, 5 (May 2014), 989–997.

Gauri Joshi, Emina Soljanin, and Gregory Wornell. 2015. Efficient replication of queued tasks for latency reduction in cloud systems. In *Proceedings of the Allerton Conference on Communication, Control, and Computing*.

S. Kochar and D. Wiens. 1987. Partial orderings of life distributions with respect to their aging properties. *Naval Research Logistics* 34, 6 (1987), 823–829.

K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran. 2016. Speeding up distributed machine learning using codes. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT'16)*.

Ankur Mallick, Malhar Chaudhari, and Gauri Joshi. 2018. Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication. arXiv:1804.10331.

W. Neiswanger, C. Wang, and E. Xing. 2013. Asymptotically exact, embarrassingly parallel MCMC. arXiv:1311.4780. http://arxiv.org/abs/1311.4780

R. Nelson and A. Tantawi. 1988. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Transactions on Computers* 37, 6 (June 1988), 739–743.

K. Ousterhout, P. Wendell, Matei Zaharia, and Ion Stoica. 2013. Sparrow: Distributed, low latency scheduling. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP'13)*. 69–84.

James Pickands III. 1968. Moment convergence of sample extremes. *Annals of Mathematical Statistics* 39, 3 (June 1968), 881–889. DOI: https://doi.org/10.1214/aoms/1177698320

M. J. D. Powell. 2007. *A View of Algorithms for Optimization Without Derivatives*. Technical Report DAMTP 2007/NA03. Cambridge University.

C. Reiss, A. Tumanov, G. Ganger, R. H. Katz, and M. A. Kozuch. 2012. *Towards Understanding Heterogeneous Clouds at Scale: Google Trace Analysis*. Technical Report. Intel Science and Technology Center for Cloud Computing. http://www.pdl.cs.cmu.edu/PDL-FTP/CloudComputing/ISTC-CC-TR-12-101.pdf.

Charles Reiss, John Wilkes, and Joseph L. Hellerstein. 2011. Google Cluster-Usage Traces: Format + Schema. Available at http://code.google.com/p/googleclusterdata/wiki/TraceVersion2.

Nihar Shah, Kangwook Lee, and Kannan Ramachandran. 2014. The MDS queue: Analyzing the latency performance of erasure codes. In *Proceedings on the IEEE International Symposium on Information Theory*.

Yin Sun, Zizhan Zheng, Can Emre Koksal, Kyu-Han Kim, and Ness B. Shroff. 2015. Provably delay efficient data retrieving in storage clouds. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'15)*.

Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA.

Rashish Tandon, Qi Lei, Alexandros G. Dimakis, and Nikos Karampatziakis. 2017. Gradient coding: Avoiding stragglers in distributed learning. In *Proceedings of the 34th International Conference on Machine Learning (PMLR 70)*. 3368–3376.

Elizabeth Varki, Arif Merchant, and Hui Chen. 2008. The M/M/1 Fork-Join Queue With Variable Sub-Tasks. Retrieved from http://www.cs.unh.edu/~varki/publication/2002-nov-open.pdf.

A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. 2013. Low latency via redundancy. In *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT'13)*. 283–294.

D. Wang. 2014. *Computing With Unreliable Resources: Design, Analysis and Algorithms*. Ph.D. Dissertation. Massachusetts Institute of Technology, Cambridge, MA.

D. Wang, G. Joshi, and G. Wornell. 2014. Efficient task replication for fast response times in parallel computation. In *Proceedings of ACM SIGMETRICS Conference*.

D. Wang, G. Joshi, and G. Wornell. 2015. Using straggler replication to reduce latency in large-scale parallel computing. In *Proceedings of the ACM SIGMETRICS Distributed Cloud Computing Workshop*.

H. Xu and W. C. Lau. 2014. Speculative execution for a single job in a MapReduce-like system. In *Proceedings of the IEEE International Conference on Cloud Computing*. 586–593.

Yaoqing Yang, Pulkit Grover, and Soummya Kar. 2017. Coded distributed computing for inverse problems. In *Advances in Neural Information Processing Systems*. 709–719. http://papers.nips.cc/paper/6673-coded-distributed-computing-for-inverse-problems.pdf.

Qian Yu, Mohammad Ali Maddah-Ali, and Amir Salman Avestimehr. 2017. Polynomial codes: An optimal design for high-dimensional coded matrix multiplication. In *Advances in Neural Information Processing Systems (NIPS'17)*.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, et al. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. 15–28.

Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*. 10.

Received August 2017; revised August 2018; accepted January 2019