Diversity vs. Parallelism in Distributed Computing with Redundancy

Pei Peng
Rutgers University
Email: pei.peng@rutgers.edu

Emina Soljanin
Rutgers University
Email: emina.soljanin@rutgers.edu

Philip Whiting
Macquarie University
Email: philip.whiting@mq.edu.au

***Abstract*—Distributed computing enables *parallel* execution of tasks that make up a large computing job. Random fluctuations in service times (inherent to computing environments) often cause a non-negligible number of straggling tasks with long completion time. Redundancy, in the form of task replication and erasure coding, has emerged as a potentially powerful way to curtail the variability in service time, as it provides *diversity* that allows a job to be completed when only a subset of redundant tasks gets executed. Thus both redundancy and parallelism reduce the execution time, but compete for resources of the system. In situations of constrained resources (here fixed number of parallel servers), increasing redundancy reduces the available level of parallelism. We characterize the *diversity vs. parallelism* tradeoff for three common models of task size dependent execution times. We find that different models operate optimally at different levels of redundancy, and thus may require very different code rates.**

I. Introduction

As numerous machine learning and other algorithms increase in complexity and data requirements, distributed, parallel computing becomes necessary, as it provides simultaneous execution of smaller tasks that make up a large computing job. However, the large-scale sharing of computing resources causes random fluctuations in task service times [1]. Therefore, although executed in parallel, some tasks, known as stragglers, take much more time to complete, which consequently increases the job service time. Redundancy, in the form of simple task replication, and more recently, erasure coding, is emerging as a potentially powerful way to shorten the job execution time, as it provides diversity that allows a job to be completed as only a subset of redundant tasks gets executed thus avoiding stragglers, see e.g. [2]–[7] and references therein.

Both parallelism and diversity are essential in reducing job service time, but compete for the system's resources. To understand that, let us consider two extreme ways to assign a job to n servers. One is *splitting* or maximum *parallelism* with no redundancy. Here, the job is divided among the n workers, and thus it gets completed when all workers execute their tasks. The other extreme way is n-fold *replication* or maximum *diversity*. Here, the entire job is given to each worker, and thus it gets completed when at least one of the workers executes its task. Roughly speaking, splitting (maximum parallelism) is appropriate for large jobs with deterministic service time, and replication (maximum diversity) is appropriate for small jobs with highly variable service time.

In general, given a fixed number of workers n, the question becomes which fraction of servers should provide parallelism and which diversity. When jobs are split in $k \geq 1$ tasks and encoded into $n \geq k$ tasks s.t. execution of any k is sufficient for job completion, then tasks whose size is k times smaller than the original job size are executed in parallel. Therefore, the smaller the k the larger the task each server is given to execute, but the smaller the k the fewer servers have to execute tasks for job completion. The choice of k thus dictates the tradeoff between the parallelism (increases with k) and diversity (decreases with k). We are here concerned with characterising the diversity vs. parallelism tradeoff for different service time and task execution models, and ultimately with finding an optimal k for a given n.

There is a large body of literature on replication and erasure codes for classical machine learning and other algorithms (see e.g. [8]–[12] and references therein), and thus it is reasonable to assume that codes exist for many jobs and any n and k combination. However, very little is known about what exact n and k combination should be selected in a given scenario in order to optimize a particular metric or goal of interest. When the goal is only to have the job complete by a certain time and it is known that at most ℓ workers will not respond by that time, then simply setting $k = n - \ell$ will achieve the goal.

However, service time in computing systems is a random variable, and one can only talk about the probability of task completion by a certain time. The question then becomes which k minimizes the expected job completion time, or (not-equivalently) maximizes the probability of job completion by a certain time. To answer these questions, we need to know the service time probability distribution (PD) and how it *scales* (changes) with the size of the task. Various service time PD have been assumed in the literature. For theoretical analysis, Pareto was used in e.g., [13]–[18], Erlang in e.g. [19]–[24], shifted exponential in e.g., [8], [25]–[27], and even exponential distribution for sufficiently small computing units in e.g., [28]–[30]. Some general classes of distributions (log-concave/convex) were considered in [3]. A model that decouples the inherent job size from the server side slowdown was developed in [31]. There is no consensus on how these PDs scale with the task size either.

In this short paper, we consider Shifted-Exponential service time and three common assumptions about its scaling with the size of the task. In Sec. II, we present the system model, and in Sec. III, IV, V, we characterize diversity vs. parallelism tradeoff for three common scaling models. Other service time

978-1-7281-6432-8/20/$31.00 ©2020 IEEE

ISIT 2020

models proposed in the literature are addressed in the long version of this paper [32], and stated here in Table I. We find that different models operate optimally at different levels of redundancy, and thus may require very different code rates.

II. System Model

System Architecture: We adopt a system model shown in Fig. 1, consisting of a single front end master node and multiple computing servers we refer to as workers. Distributed, parallel computing system architectures where a single master node manages the entire computing cluster of nodes is commonly implemented in modern frameworks such as Kubernetes [33] and Apache Mesos [34].

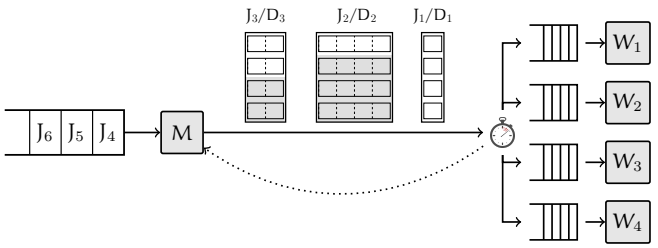

Fig. 1. **A Distributed Computing System:** Master node M partitions jobs J_i into tasks, possibly generates redundant tasks, and dispatches them to workers W_1, W_2, W_3, W_4. Shaded regions in the pre-processed jobs J_2, J_3 indicate redundancy. Here, each job consists of 4 computing units. Job J_1 is executed with maximum parallelism (splitting), J_2 with maximum diversity (replication), and J_3 is encoded by a $[4, 2]$ erasure code.

Computing Jobs, Tasks, and Units: We are concerned with computing jobs that can be split into tasks which can be executed independently in parallel on different workers. An example of such a job is vector by matrix multiplication, a basic operation in e.g. regression analysis and PageRank, and another is the gradient descent residing at the core of almost any machine learning algorithm [35]–[38]. We assume that there is some minimum size task below which distributed computing would be inefficient because of e.g., the overhead of communication and data transfer costs. We refer to this minimum-size task as the *computing unit (CU)*. For example, if the job is to multiply vector X by matrix A, the CU can be the scalar product of a row of A and X. We will measure the size of the job by the number of its computing units.

Master node M partitions jobs into tasks, possibly generates redundant tasks, and dispatches them to workers. If the number of workers assigned to a job is smaller than the size of the job (number of CUs) with added redundancy, then each worker will be responsible for multiple CUs constituting its task. Fig. 1 shows some possible ways in which the master server can pre-process a job, namely, partition a job into CUs, group the CUs into tasks, and add redundancy. Pre-processing of jobs J_1, J_2, and J_3 results in redundant service demands D_1, D_2, and D_3. No redundant tasks are formed for job J_1, and thus D_1 and the original job are identical. Job J_2 is replicated on 4 workers, and thus the size of D_2 is 4 times the size of J_2. Job J_3 is encoded by a systematic $[4, 2]$ MDS code that generated 2 coded tasks of 2 CUs size. Its redundant version D_3 is organized as follows: Workers W_1 and W_2 are each given a task consisting of 2 different CUs of J_3. Workers W_3 and W_4 are each given a coded task of 2 CUs size.

Models for Service Time: We model a computing unit service time as a random variable (RV) V, and refer to the tasks that are still running after certain time elapsed as stragglers. In this short paper, we consider only the Shifted-Exponential service: $V \sim \Delta + X$, where Δ is a constant modelling the minimum service time, and $X \sim \mathrm{Exp}(W)$ is an exponentially distributed RV with the expectation W (rate $1/W$) modelling straggling. This simple model will allow us to point out the dependence of the parallelism/diversity tradeoff on the assumptions made about the execution time of consecutive CUs. Other service time models proposed in the literature are addressed in the long version of this paper [32].

Models for Service Time of Consecutive Computing Unit: A common assumption is that service time of consecutive CUs are independent. Some other models have been considered in literature. For example, if a CUs service time is exponential with expectation W, then some models assume that the service time for s computing units will be also exponential with expectation $s \cdot W$ (i.e., scaled exponential [8], [30]) while other models assume that it will be Erlang with rate expectation $s \cdot W$ and shape s (i.e., sum of s independent exponential RVs). If a computing unit service time is a shifted exponential with expectation W and shift Δ, then some models assume that the service time for s computing units will be also be shifted exponential with the unchanged rate W and an s times larger shift $s \cdot \Delta$ [10], [25], [27], [39], [40].

We consider three different, commonly adopted models for service time of consecutive CUs execution on the same server. For all three models, we assume independence across the servers. The models are described next, and their impact on the diversity vs. parallelism tradeoff is analysed in the next section. We explain the models by giving the distribution for execution time Y of s consecutive tasks on a single server under the assumption that a single task execution time V is $\text{S-Exp}(\Delta, W)$ distributed as described above.

Model 1 – *Server-Dependent Execution Time:* The assumption here is that the straggling effect depends on the server and is identical for each CU executed on that server. Namely, there is some initial handshake time Δ after which server i completes its first and each subsequent CU in time $X_i \sim \mathrm{Exp}(W)$. Therefore, $Y = \Delta + s \cdot X_i$, that is, $Y \sim \text{S-Exp}(\Delta, s \cdot W)$.

Model 2 – *Data-Dependent Execution Time:* The assumptions here are that 1) each CU in a task of s CUs takes Δ time units to complete and 2) there are some inherent additive system randomness at each server which does not depend on the task size s that determines the straggling effect $X_i \sim \mathrm{Exp}(W)$. Therefore, $Y = s \cdot \Delta + X_i$, that is $Y \sim \text{S-Exp}(s \cdot \Delta, W)$.

Model 3 – *Additive Execution Time:* The assumption here is that the execution times of CUs are independent and identically distributed. Therefore, $Y = V_1 + \cdots + V_s$ where $V_i \sim$

S-Exp(Δ, W) are independent. and thus $Y \sim \text{Erlang}(s{\cdot}W, s)$.

Mathematical Tools: In executing jobs with redundant tasks, the notion of order statistics plays a central role. We here state the results we rely on in the rest of the paper. More information can be found in, e.g., [41]–[44].

Let $X_1, X_2, \ldots, X_n$ be n samples of some RV X. Then the k-th smallest is an RV denoted by $X_{k:n}$ and known as the k-th order statistics of $X_1, \ldots X_n$. If X is Exp(W), then $X_{1:n}$ is Exp(W/n), and

$$X_{k:n} = X_{1:n} + X_{1:(n-1)} + \cdots + X_{1:2} + X_{1:(n-k+1)}. \quad (1)$$

The expectation of $X_{k:n}$ is given by

$$\mathbb{E}[X_{k:n}] = W \sum_{i=1}^{k} \frac{1}{n-k+i} = W\big(H_n - H_{n-k}\big) \quad (2)$$

where H_n is (generalized) harmonic numbers defined as $H_n = \sum_{j=1}^{n} \frac{1}{j}$. We often use the approximation $H_n = \log n + \gamma + \mathcal{O}(n^{-1})$, where $\gamma = 0.5772156649$ is the Euler's constant.

If X is Erlang(s, W), then, according to the formula of gamma order statistics in [45], we have

$$X_{k:n} = \frac{Wk}{(s-1)!}\binom{n}{k}\sum_{i=0}^{k-1}(-1)^i\binom{k-1}{i} \sum_{j=0}^{(s-1)(n-k+i)} \alpha_j(s, n-k+i)\frac{(s+j)!}{(n-k+i+1)^{s+j+1}}$$

where $\alpha_z(x, y)$ is the coefficient of t^z in the expansion of $\left(\sum_{l=0}^{x-1} t^l/l!\right)^y$. This complex formula does not provide any insight into the diversity vs. parallelism tradeoff, and we will only use it for the numerical analysis.

Parameters and Notation:

n	–	number of workers (number of CUs in a job)
k	–	number of workers that have to execute their tasks for job completion
s	–	number of CUs per task, $s = n/k$
$Y_{n,k}$	–	job completion time when each worker's task size is $\frac{n}{k} = s$

III. Server-Dependent Execution Time

Theorem 1. *The expected job completion time for the server-dependent execution model is given by*

$$\mathbb{E}[Y_{n,k}] = \Delta + sW(H_n - H_{n-k}) = \Delta + W\frac{n}{k}(H_n - H_{n-k})] \geq \Delta + W$$

which is minimized by replication (maximal diversity).

Proof. The equality above follows from the observation that

$$Y_{n,k} = \Delta + X_{k:n}, \quad \text{where} \quad X \sim \text{Exp}(s \cdot W)$$

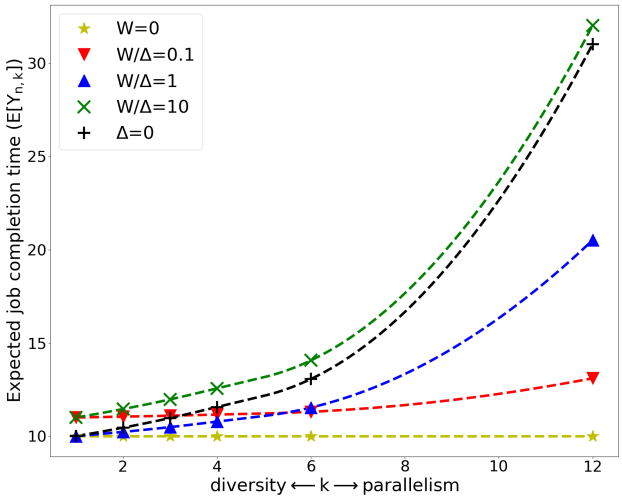

Fig. 2. Diversity vs. parallelism for server-dependent execution time model. Number of workers $n = 12$ is fixed. Replication or maximum diversity always gives the best results.

and (2). The inequality above follows from the definition of H_n and is saturated when $k = 1$. We can also show that $\mathbb{E}[Y_{n,k}]$ is an increasing function of k for a given n, as follows:

$$\begin{aligned}\mathbb{E}[Y_{n,k+1}] &= \Delta + W\frac{n}{k+1}(H_n - H_{n-k-1}) \\ &= \Delta + W\frac{n}{k+1}\Big(H_n - H_{n-k} + \frac{1}{n-k}\Big) \\ &= \mathbb{E}[Y_{n,k}] + \frac{Wn}{k+1}\Big[\frac{1}{n-k} - \frac{1}{k}(H_n - H_{n-k})\Big]\end{aligned}$$

where the term in square brackets is positive. □

Numerical Analysis: We numerically evaluate the expression of Thm. 1 to see how the expected job completion time $\mathbb{E}[Y_{n,k}]$ changes with the value of k. We consider a system with $n = 12$ workers for five different values of W/Δ: 1. $W = 0$ ($\Delta = 10$); 2. $W/\Delta = 0.1$ ($W = 1$, $\Delta = 10$); 3. $W/\Delta = 1$ ($W = 5$, $\Delta = 5$); 4. $W/\Delta = 10$ ($W = 10$, $\Delta = 1$); 5. $\Delta = 0$ ($W = 10$). Note that different values of W and Δ with the equal W/Δ give different values of $\mathbb{E}[Y_{n,k}]$. Therefore the results plotted in the figures throughout the paper should not be compared only on the basis of W/Δ.

The numerical evaluation results are plotted in Fig. 2. When $W/\Delta \to 0$, the value of Δ dominates the expected job completion time $\mathbb{E}[Y_{n,k}]$, which thus changes little with k. When $W/\Delta > 0$, we see that $\mathbb{E}[Y_{n,k}]$ always reaches the minimum at $k = 1$, which means replication or maximum diversity is optimal. As W/Δ increases, the slope of the corresponding curves also increases. Although it is always optimal, maximal diversity is much more effective when W/Δ is large, and has less payoff when W/Δ is small.

IV. Data-Dependent Execution Time

Theorem 2. *The expected job completion time for the data-dependent execution model is given by*

$$\mathbb{E}[Y_{n,k}] = s\Delta + W(H_n - H_{n-k}) = W\Big[\frac{n}{k}\cdot\frac{\Delta}{W} + (H_n - H_{n-k})\Big]$$

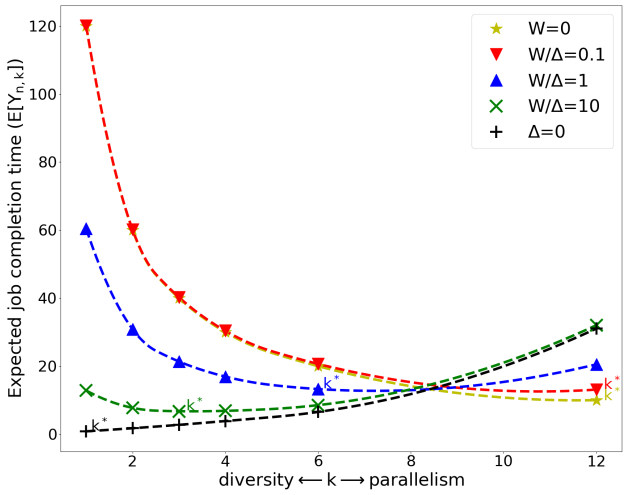

Fig. 3. Diversity vs. parallelism for data-dependent execution time model. Number of workers $n = 12$ is fixed. In low W/Δ scenarios, parallelism is more expected; In high W/Δ scenarios, diversity is more expected.

The result follows from (2) and the observation that

$$Y_{n,k} = s \cdot \Delta + X_{k:n}, \quad \text{where} \quad X \sim \text{Exp}(W)$$

By taking the log approximation of the harmonic numbers, we find k^* which minimizes the above expression as follows:

$$k^* = n(-r/2 + \sqrt{r + r^2/4}), \quad r = \Delta/W$$

Note that this expression depends only on the ratio $r = \Delta/W$, which is intuitively expected. For $\Delta \gg W$, (large r) the execution is essentially deterministic. It is optimal to use maximum parallelism, that is, splitting ($k = n$) is optimal. On the other hand, when $W \gg \Delta$,(small r) execution time is much more variable and it is optimal to operate with maximum diversity, that is, replication ($k = 1$) is optimal (cf. [25]).

Numerical Analysis: We numerically evaluate the expression in Thm 1 for $\mathbb{E}[Y_{n,k}]$ vs. k. We again consider a system with $n = 12$ workers and the same values of W/Δ, W and Δ as in Sec. III. The numerical evaluation results are plotted in Fig. 3. By comparing different W/Δ scenarios, we can easily conclude that when W/Δ (e.g. 0, 0.1) is small, the expected job completion time $\mathbb{E}[Y_{n,k}]$ always decreases with k, which means splitting is optimal. When W/Δ is large (e.g. 1, 10), we find that $\mathbb{E}[Y_{n,k}]$ reaches the minimum for $k = 3$ or 6, which means coding at a certain non-trivial rate is optimal. When W/Δ (e.g. $\Delta = 0$) is very large, $\mathbb{E}[Y_{n,k}]$ always increases with k, which means replication is optimal.

V. Additive Execution Time

The job completion time for the additive model is

$$Y_{n,k} = s \cdot \Delta + X_{k:n}, \quad \text{where} \quad X \sim \text{Erlang}(s, W)$$

The expectation of the k-th order statistics of Erlang distribution is given by the formula stated in Sec. II, which is not helpful in analysing the diversity/parallelism tradeoff. We here find an analytic expressions for the expected job completion time under splitting and replication, and show that splitting outperforms replication for sufficiently large n. We then show that rate $1/2$ coding outperforms splitting when $\Delta = 0$.

Splitting, $s = 1$, and Replication, $s = n$

Under splitting, the job completion time is given by

$$Y_{n,n} = \Delta + X_{1:n} + X_{1:(n-1)} + \cdots + X_{1:2} + X_{1:1} \tag{3}$$

(see (1)), and therefore,

$$\mathbb{E}[Y_{n,n}] = \Delta + W H_n \tag{4}$$

Under replication, we have

$$\mathbb{E}[Y_{n,1}] = n\Delta + W\frac{1}{n}\int_0^\infty e^{-t}\Big[R_n\Big(\frac{t}{n}\Big)\Big]^n dt$$

$$\text{where} \quad R_n(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^{n-1}}{(n-1)!}$$

This result is a corollary of the following theorem:

Theorem 3. *Let service times of tasks be independent and exponential with rate 1. If a job with k tasks is replicated over n workers, then the expected completion time is*

$$\frac{1}{n}\int_0^\infty e^{-t}\Big[S_k\Big(\frac{t}{n}\Big)\Big]^n dt \tag{5}$$

$$\textit{where} \quad S_k(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^{k-1}}{(k-1)!}$$

Proof. Let $t_1, t_2, \ldots$ be time epochs at which a task gets completed on any of the n servers. Because all k tasks of the job are replicated on each of the n servers, the job is completed when k tasks get completed on any single server, which happens at some time $t_{\ell_{k,n}}$. Note that $\ell_{k,n}$ is a random variable. We represent $t_{\ell_{k,n}}$ as a sum of the task inter-completion times.

$$t_{\ell_{k,n}} = \sum_{j=1}^{\ell_{k,n}} (t_j - t_{j-1}), \quad \text{where } t_0 \text{ is set to 0.} \tag{6}$$

Note that 1) $t_j - t_{j-1}$ are independent and exponentially distributed with rate n (the minimum of n independent exponentials with rate 1), and 2) $t_j - t_{j-1}$ are independent from $\ell_{k,n}$ Observe next that Wald's identity can be applied to (6). Therefore,

$$\mathbb{E}[t_{\ell_{k,n}}] = \frac{1}{n} \cdot \mathbb{E}[\ell_{k,n}]$$

Now observe that $\mathbb{E}[\ell_{k,n}]$ corresponds to the expected number of draws from n coupons until a coupon shows up k times. The claim follows from the result for $\mathbb{E}[\ell_{k,n}]$ in [46]. □

Paper [46] also gives a useful approximation of (5) when n goes infinity, which further simplifies the expression $\mathbb{E}[Y_{n,1}]$:

$$\mathbb{E}[Y_{n,1}] \sim n\Delta + \frac{W}{n}\sqrt[n]{n!}\,\Gamma(1 + 1/n)n^{1-\frac{1}{n}}, \quad \text{as} \quad n \to \infty \tag{7}$$

Lemma 1. *For large enough n, splitting (maximal parallelism) outperforms replication (maximal diversity).*

TABLE I
OPTIMAL STRATEGY DEPENDENCE ON THE SERVICE TIME PD AND SCALING

Service time scaling \ Service time PD	(Shifted-)Exp	Pareto	Bi-Modal
Server-Dependent	replication	splitting⟶coding*	splitting⟶coding⟶splitting
Data-Dependent	splitting⟶replication	splitting⟶replication	splitting⟶coding⟶splitting
Additive	splitting⟶coding	splitting⟶coding	splitting⟶coding⟶splitting

* ⟶ indicates how the optimal strategy changes as the straggling probability increases (the PD tail becomes heavier).

Proof. By using Stirling's formula for $\sqrt[n]{n!}$ in (7), we have

$$\begin{aligned}(7) &\geq n\Delta + \frac{W}{n}\sqrt[n]{\sqrt{2\pi}n^{n+\frac{1}{2}}e^{-n}}\,\Gamma(1+1/n)n^{1-\frac{1}{n}} \\ &> n\Delta + \frac{W}{en}\sqrt[2n]{2\pi}\,n^{1+\frac{2}{n}}n^{1-\frac{1}{n}} > n\Delta + \frac{W}{e}n^{1+\frac{1}{n}} \\ &> \Delta + WH_n = \mathbb{E}[Y_{n,n}]\end{aligned}$$

Observe that by choosing large enough n, we can make $\mathbb{E}[Y_{n,1}]$ arbitrarily close to (7). Recall that $H_n = \mathcal{O}(\log n)$. Note that the theorem holds even for $\Delta = 0$. □

Rate $1/2$ Coding, $s = 2$: We consider the special case when $\Delta = 0$, n is even, and $s = n/k = 2$. Therefore, $k = n/2$ workers have to complete their two tasks in order for the job itself to be complete.

Let $Y_{n,n}$ be the time to complete the job under splitting, as given by (3) for $\Delta = 0$, and $Y_{n,n/2}$ the random time to complete the job under coding with $s = 2$. Thm 4 below shows that $\mathsf{P}\{Y_{n,n/2} > x\} \leq \mathsf{P}\{Y_{n,n} > x\}$. It follows that

$$\mathbb{E}[Y_{n,n/2}] \leq \mathbb{E}[Y_{n,n}]$$

since for any non-negative random variable X, we have $\mathbb{E}[X] = \int_0^\infty \mathsf{P}(X > x)\,dx$.

It is, therefore, better to use a rate half code than splitting.

Theorem 4. *Suppose that $n = 2k \geq 4$ is even. Then $Y_{n,n/2}$ stochastically dominates $Y_{n,n}$, that is,*

$$\mathsf{P}\{Y_{n,n/2} > x\} \leq \mathsf{P}\{Y_{n,n} > x\} \tag{8}$$

Proof. Consider the $s = 2$ system where scheduling is done as follows. The system runs until one server completes the first of its 2 CUs, at which point it gets halted. This happens at a random time distributed as $X_{1:n}$. The system of the remaining $n-1$ servers runs until one server completes the first of its 2 CUs, at which point it gets halted. This happens at a time distributed as $X_{1:(n-1)}$ measured from the moment the first server was halted. The process continues in the same manner until $k = n/2$ servers have completed the first of their 2 CUs, at which point all remaining servers get halted. This happens at a random time T_1 given as

$$T_1 = X_{1:n} + X_{1:(n-1)} + X_{1:(n-k+1)} \tag{9}$$

At this point, the $n - k = k$ servers with one completed CU get restarted. The job is complete when each server completes its other CU, which happens at a random time T_2 given as

$$T_2 = X_{1:(n-k)} + X_{1:(n-k-1)} + X_{1:1} \tag{10}$$

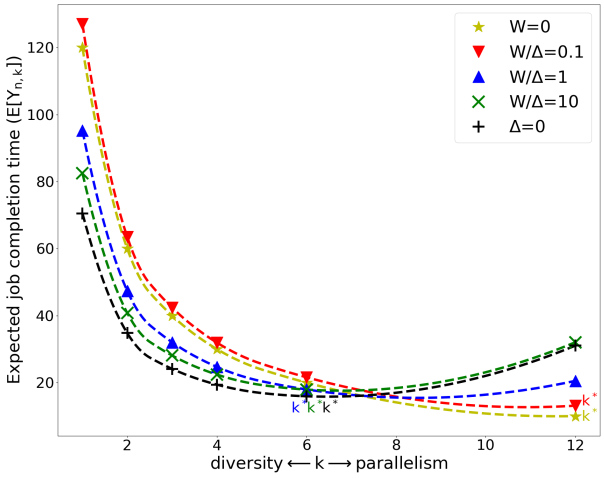

Fig. 4. Diversity vs. parallelism for additive execution time model. Number of workers $n = 12$ is fixed. Parallelism is always expected. In low W/Δ scenarios, splitting or maximum parallelism is the best; In high W/Δ scenarios, there is a balance between diversity and parallelism.

Because some servers get halted, this system cannot perform better the original $s = 2$ system. On the other hand, it performs as as well as the $s = 1$ system since $Y_{n,n} = T_1 + T_2$. □

Numerical Analysis: We numerically evaluated the the derived expressions for $\mathbb{E}[Y_{n,k}]$; the results are shown in Fig.4. We see that when W/Δ is small (e.g., 0, 0.1), splitting (maximum parallelism) gives the best performance. On the other hand, when W/Δ is larger (e.g., 1, 10, ∞), we need to find the optimal balance between the diversity and parallelism. The figure also supports Lemma 1 and Thm 4 that say that splitting is always better than replication and the rate half code is better than splitting when $\Delta = 0$.

VI. CONCLUSIONS

We addressed the question concerning the code rate that minimizes the expected computing time for a given the service time probability distribution and its scaling with the task size. Results for the shifted-exponential service are derived in this paper. Table I summarises more general results reported in [32]. We found that different models operate optimally at different levels of redundancy, and thus may require very different code rates and even no coding at all.

ACKNOWLEDGMENT

Part of this research is based upon work supported by the National Science Foundation under Grant No. CIF-1717314.

REFERENCES

[1] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[2] M. F. Aktas and E. Soljanin, "Straggler mitigation at scale," *IEEE/ACM Trans. Netw.*, vol. 27, no. 6, pp. 2266–2279, 2019.

[3] G. Joshi, E. Soljanin, and G. Wornell, "Efficient redundancy techniques for latency reduction in cloud systems," *ACM Trans. Modeling and Performance Evaluation of Computing Sys. (TOMPECS)*, vol. 2, 2017.

[4] G. Joshi, E. Soljanin, and G. W. Wornell, "Efficient replication of queued tasks for latency reduction in cloud systems," in *53rd Annual Allerton Conf. on Commun., Control, and Computing*, 2015, pp. 107–114.

[5] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 1988–1992.

[6] E. Ozfatura, D. Gündüz, and S. Ulukus, "Speeding up distributed gradient descent by utilizing non-persistent stragglers," in *2019 IEEE Internat. Symposium on Information Theory (ISIT)*, 2019.

[7] A. Behrouzi-Far and E. Soljanin, "On the effect of task-to-worker assignment in distributed computing systems with stragglers," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2018, pp. 560–566.

[8] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," *Advances in Neural Information Processing Systems*, vol. 29, pp. 2100–2108, 2016.

[9] S. Li, M. Maddah-Ali, and A. S. Avestimehr, "Coded mapreduce," in *Proceedings of the Allerton Conference on Communication, Control and Computing*, Oct. 2015.

[10] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Jul. 2016.

[11] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. Internat. Conf. on Machine Learning (ICML)*, Aug. 2017, pp. 3368–3376.

[12] C. Karakus, Y. Sun, S. N. Diggavi, and W. Yin, "Redundancy techniques for straggler mitigation in distributed optimization and learning." *Journal of Machine Learning Research*, vol. 20, no. 72, pp. 1–47, 2019.

[13] K. Psounis, P. Molinero-Fernández, B. Prabhakar, and F. Papadopoulos, "Systems with multiple servers under heavy-tailed workloads," *Performance Evaluation*, vol. 62, no. 1, pp. 456–474, 2005.

[14] N. Bansal and M. Harchol-Balter, *Analysis of SRPT scheduling: Investigating unfairness*. ACM, 2001, vol. 29, no. 1.

[15] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 283–294.

[16] M. F. Aktas, P. Peng, and E. Soljanin, "Effective straggler mitigation: Which clones should attack and when?" *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, pp. 12–14, 2017.

[17] G. Joshi, "Synergy via redundancy: Boosting service capacity with adaptive replication," *SIGMETRICS Performance Evaluation Review*, vol. 45, no. 3, pp. 21–28, Mar. 2018.

[18] M. F. Aktas, P. Peng, and E. Soljanin, "Straggler mitigation by delayed relaunch of tasks," in *Proc. of the IFIP WG 7.3 Performance*, 2017.

[19] S. Kwon and N. Gautam, "Time-stable performance in parallel queues with non-homogeneous and multi-class workloads," *Biological Cybernetics*, vol. 24, no. 3, pp. 1322–1335, 2016.

[20] T. Vasantam, A. Mukhopadhyay, and R. R. Mazumdar, "Mean-field analysis of loss models with mixed-erlang distributions under power-of-d routing," in *Teletraffic Congress (ITC 29), 2017 29th International*, vol. 1. IEEE, 2017, pp. 250–258.

[21] A. Gorbunova, I. Zaryadov, S. Matyushenko, and E. Sopin, "The estimation of probability characteristics of cloud computing systems with splitting of requests," in *International Conference on Distributed Computer and Communication Networks*. Springer, 2016, pp. 418–429.

[22] F. Poloczek and F. Ciucu, "Contrasting effects of replication in parallel systems: From overload to underload and back," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 1, pp. 375–376, 2016.

[23] A. Thomasian, "Analysis of fork/join and related queueing systems," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, p. 17, 2015.

[24] C. Banerjee, A. Kundu, A. Agarwal, P. Singh, S. Bhattacharya, and R. Dattagupta, "Priority based k-erlang distribution method in cloud computing," *International Journal on Recent Trends in Engineering & Technology*, vol. 10, no. 1, p. 135, 2014.

[25] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, 2012, pp. 326–333.

[26] G. Liang and U. C. Kozat, "Fast cloud: Pushing the envelope on delay performance of cloud storage with coding," *IEEE/ACM Transactions on Networking*, vol. 22, no. 6, pp. 2012–2025, 2014.

[27] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing latency for secure coded computing using secret sharing via staircase codes," *IEEE Transactions on Communications*, 2020.

[28] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. Hyytia, "Reducing latency via redundant requests: Exact analysis," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, pp. 347–360, 2015.

[29] M. F. Aktas, E. Najm, and E. Soljanin, "Simplex queues for hot-data download," in *Proceedings of the SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, 2017.

[30] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 989–997, 2014.

[31] K. Gardner, M. Harchol-Balter, A. Scheller-Wolf, and B. Van Houdt, "A better model for job redundancy: Decoupling server slowdown and job size," *IEEE/ACM transactions on networking*, vol. 25, no. 6, pp. 3353–3367, 2017.

[32] P. Peng, E. Soljanin, and P. Whiting, "Diversity vs. parallelism in distributed computing with redundancy." [Online]. Available: https://emina.flywheelsites.com

[33] "Kubernetes Documentation," https://kubernetes.io/docs/reference/, 2018, [Online; accessed 26-Oct-2018].

[34] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *NSDI*, vol. 11, 2011, pp. 22–22.

[35] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ML via a stale synchronous parallel parameter server," in *Advances in neural information processing systems*, 2013, pp. 1223–1231.

[36] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, and M. Devin, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[37] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," *arXiv preprint arXiv:1604.00981*, 2016.

[38] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 69–77.

[39] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 15–26.

[40] A. Behrouzi-Far and E. Soljanin, "Data replication for reducing computing time in distributed systems with stragglers," *arXiv preprint arXiv:1912.03349*, 2019.

[41] A. Rényi, "On the theory of order statistics," *Acta Mathematica Hungarica*, vol. 4, pp. 191–231, 1953.

[42] B. C. Arnold, N. Balakrishnan, and H. N. Nagaraja, *A First Course in Order Statistics (Classics in Applied Mathematics)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008.

[43] H. A. David, *Order Statistics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1039–1040.

[44] B. C. Arnold, *Pareto distribution*. Wiley Online Library, 2015.

[45] S. S. Gupta, "Order statistics from the gamma distribution," *Technometrics*, vol. 2, no. 2, pp. 243–262, 1960.

[46] M. S. Klamkin and D. J. Newman, "Extensions of the birthday surprise," *Journal of Combinatorial Theory*, vol. 3, no. 3, pp. 279–282, 1967.