Single-Forking of Coded Subtasks for Straggler Mitigation

Ajay Badita, *Graduate Student Member, IEEE*, Parimal Parag, *Senior Member, IEEE*, and Vaneet Aggarwal, *Senior Member, IEEE*

***Abstract*— Given the unpredictable nature of the nodes in distributed computing systems, some of the tasks can be significantly delayed. Such delayed tasks are called stragglers. Straggler mitigation can be achieved by redundant computation. In maximum distance separable (MDS) redundancy method, a task is divided into k subtasks which are encoded to n coded subtasks, such that a task is completed if any k out of n coded subtasks are completed. Two important metrics of interest are task completion time, and server utilization which is the aggregate completed work by all servers in this duration. We consider a proactive straggler mitigation strategy where n_0 out of n coded subtasks are started at time 0 while the remaining $n - n_0$ coded subtasks are launched when $\ell_0 \leq \min\{n_0, k\}$ of the initial ones finish. The coded subtasks are halted when k of them finish. For this flexible forking strategy with multiple parameters, we analyze the mean of two performance metrics when the random service completion time at each server is independent and distributed identically (*i.i.d.*) to a shifted exponential. From this study, we find a tradeoff between the metrics which provides insights into the parameter choices. Experiments on Intel DevCloud illustrate that the shifted exponential distribution adequately captures the random coded subtask completion times, and our derived insights continue to hold.**

***Index Terms*— Straggler mitigation, distributed computing, completion time, scheduling, forking points.**

I. Introduction

MOTIVATED by scalability, availability, and reliability, there has been a paradigm shift from centralized computation at a large supercomputer to distributed computing on a large cluster of regular compute servers to perform complex tasks. In distributed compute setting, a single task is fragmented into a smaller number of subtasks, and processed by the compute cluster. Task completion time is limited by the slowest execution time of the parallel subtasks. The lagging tasks are referred to as stragglers, and they delay the entire task execution. Straggling servers is one of the challenges in distributed computing.

Stragglers can be mitigated by adding redundant subtasks, where a task divided into k subtasks can be encoded into n redundant subtasks, and each coded subtask is processed individually at a unique server. A popular redundancy technique to mitigate stragglers is replication, where each of the finite k subtasks can be replicated to $\frac{n}{k}$ servers each [2]–[4]. However, the task will only be done if one sever from each of the $\frac{n}{k}$ partitions finish processing their corresponding subtasks. In [5], [6], the authors replicate subtasks only for a fraction of k subtasks, and the subtask is said to be complete only if any one of the corresponding replicated subtask is complete. In general, redundancy can be achieved by erasure coding schemes more general than replication. The key advantage of erasure coding is that it reduces storage cost while providing similar reliability as replicated systems [7], [8], and thus has now been widely adopted for storage by companies like Facebook [9], Microsoft [10], and Google [11]. It has been shown that this more flexible redundancy scheme can also be employed for certain computing tasks in distributed compute systems [12]–[18].

We focus on an efficient erasure coding scheme called maximum distance separable (MDS) coding [12], [15], where multiple servers process coded version of k subtasks such that the task is complete if any k out of n servers finish processing their coded subtasks. MDS coding is a more general form of redundancy than simple replication. We observe that even though MDS codes are efficient, the decoding complexity is polynomial in the length of the code [19], [20]. When the decoding time is negligible as compared to the coded subtask completion time, MDS codes are an attractive choice. In fact, it has been shown in [21] that Intel Storage Acceleration Library (ISA-L) provides a highly optimized implementation of Reed-Solomon (RS) codes which significantly decreases the time taken for encoding and decoding operations. Further, [21] showed that ISA-L can achieve a significant large decoding throughput of 163 Mbps, demonstrating negligible decoding overheads for erasure-coded systems.

Assuming that each server is working on a unique coded subtask, an additional benefit of redundancy schemes is the lower completion time due to parallelization gains [4],

Manuscript received August 25, 2020; revised February 15, 2021 and April 7, 2021; accepted April 14, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor B. Ji. Date of publication July 2, 2021; date of current version December 17, 2021. This work was supported in part by the Science and Engineering Research Board under Grant DSTO-1677, in part by the Department of Telecommunications, Government of India, under Grant DOTC-0001, in part by the Robert Bosch Center for Cyber-Physical Systems, in part by the Centre for Networked Intelligence of the Indian Institute of Science, Bengaluru, in part by the Visiting Advanced Joint Research (VAJRA) Fellowship, in part by the National Science Foundation under Grant CNS-1618335, and in part by CISCO. This work was presented in part at IEEE INFOCOM, July 2020 [1]. *(Corresponding author: Vaneet Aggarwal.)*

Ajay Badita and Parimal Parag are with the Department of Electrical Communication Engineering, Indian Institute of Science, Bengaluru 560012, India (e-mail: ajaybadita@iisc.ac.in; parimal@iisc.ac.in).

Vaneet Aggarwal is with the School of Industrial Engineering, Purdue University, West Lafayette, IN 47907 USA, and also with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: vaneet@purdue.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNET.2021.3075377, provided by the authors.

Digital Object Identifier 10.1109/TNET.2021.3075377

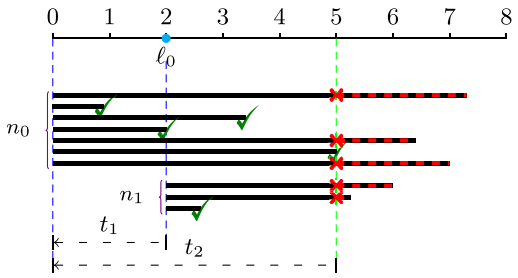

Fig. 1. The start and completion times of different coded subtasks, for a single-forked task, divided into $k = 5$ subtasks, MDS coded to $n = 10$ coded subtasks, where $n_0 = 7$ coded subtasks are initiated at n_0 servers. After the completion of $\ell_0 = 2$ coded subtasks, remaining $n_1 = 3$ coded subtasks are initiated at n_1 servers. The task completion time is the time to finish first $k = 5$ coded subtasks.

[12]–[18], [20], [22]. However this latency reduction comes at the cost of high server utilization [3], [4], [6]. High aggregate utilization of all servers leads to increased operation cost and hence it is desirable to reduce the server utilization. In this setting, we consider the following important question. When should the n coded subtasks be started to obtain an optimal trade-off between task completion time and server utilization cost? One option is to start all coded subtasks at time 0, corresponding to the task request time. This leads to using all n servers until the first k of them have finished, resulting in low task completion time and a high server utilization. Another option is to use only k servers and start with all of them at time 0. This would help avoid the excess server utilization for the remaining $n-k$ servers, resulting in low server utilization and larger task completion time.

A more flexible approach is to start with $n_0 < n$ coded subtasks at time 0. When $\ell_0 < k$ of them are finished, we launch the remaining $n_1 = n - n_0$ servers. This launching time instant is called *forking point*, and the threshold ℓ_0 on number of coded subtask completions is referred to as *fork task threshold*. An example of this delayed launch of coded subtasks is illustrated in Fig. 1 for $n = 10, n_0 = 7, \ell_0 = 2$, where we plot the start and completion time for each coded subtask for a single task.

In the first option of starting all n servers, we have $n_0 = n$, $\ell_0 = k$, $n_1 = 0$. This is the setting of an (n, k)-fork-join [23]. In the option of using only k servers and starting with all of them, we have $n_0 = \ell_0 = k$. Thus, the proposed approach affords a flexible framework for launching the coded subtasks. It is not *apriori* clear as to how should these parameters be chosen so that both the metrics are optimized. In this article, we aim to find the impact of design parameters n_0, n_1, and ℓ_0 on the two metrics; the task completion time, and the server utilization.

A. Related Work

Given the unpredictable nature of the nodes in distributed systems, coding theoretic techniques have been used for straggler mitigation in the face of uncertainty. Coding-theoretic approaches have been shown to provide a tradeoff between access latency and server utilization in distributed storage systems [24]. It was shown in [22] that MDS codes are the latency-minimizing code among a class of symmetric codes for distributed storage systems. Coding theoretic techniques have been provided for mitigating stragglers in matrix multiplication [15]–[18]. The authors of [12], [14] consider the problem of computing gradients in a distributed system, and propose a novel coded computation scheme tailored for computing a sum of functions. While most of the works focus on the application of coded computation to linear operations, coding has also been found useful in distributed computing frameworks involving nonlinear operations [13]. Efficient coding theoretic techniques to reduce the communication cost in the process of transferring the results of mappers to reducers have been studied in [15], [25]–[28].

We note that we are considering task completion time and server utilization for a single task. Mean task completion time for a sequence of task arrivals for the (n, k)-fork-join queue is considered in [3], [22]–[24], [29]–[33]. These articles have provided analytical results for a static (n, k) redundancy under various simplified settings. For memoryless service, tight numerical bounds are presented in [29], analytical bounds are provided in [23], [24], [30], [31], tight analytical approximations in [22], and exact analysis for small systems in [3]. An exact analysis of tail index for Pareto-distributed file sizes is studied in [33], and an exact analysis for random independent scheduling for asymptotically large number of servers in [32]. These works demonstrate the improvement of task completion times with the use of coding. However, this line of work does not take the server utilization into account. In our work, we have generalized the static coded redundancy studied in the above-mentioned articles. We show that server utilization can be reduced by dynamic coded redundancy, where the number of parallel servers available to a task changes with time. To keep the analysis tractable, we do not consider the task arrivals. In this case, dynamic coded redundancy is tantamount to efficient launching times of the different coded subtasks.

One of the cost-effective approaches to mitigate the effect of stragglers is to either re-launch a certain task if it is delayed, or preemptively assign each task to multiple nodes and move on with the copy that completes first. Speculative execution have been studied in [34], which acts after the tasks have already slowed down. In a proactive mitigation approach, one can launch redundant copies of a task hoping that at least one of them will finish in a timely manner. The authors of [2] perform cloning to mitigate the effect of stragglers. The authors of [4], [6] analyzed the latency and cost for replication-based strategies for straggler mitigation. A machine learning approach for predicting and avoiding these stragglers has been studied in [35]. Recently, coding-theory-inspired approaches have been applied to mitigate the effect of straggling as mentioned earlier. Single-fork analysis with coding has been studied in [36], where k coded subtasks are started at $t = 0$. Further, after a fixed deterministic time Δ, additional $n - k$ coded subtasks are started. Our work differs from [36] since

(i) we allow for general number of initial coded subtasks,
(ii) the start time of new coded subtasks is random and based on the completion time of certain number of coded subtasks rather than a fixed constant, and

(iii) our framework allows for an optimization of different parameters to provide a tradeoff between server utilization and task completion time.

We note that the problem is important even when there are stochastic arrivals since this procedure of forking can be used for any arriving task. The exact queueing analysis for coded-tasks with forking is not straightforward to extend and remains open, while the analysis in this article provide insights on how to efficiently fork a task in lightly-loaded scenarios. This scenario arises in the case of low arrival rates so that the queues are empty with high probability, and hence the system can be modeled as an $M/G/1$ queue where the service time of a task is the completion time computed in this work. Thus, one can achieve a tradeoff between the two performance metrics for lightly loaded queueing systems.

B. Contributions

We characterize the means of the two performance metrics: task completion time and the server utilization, for a single (n, k)-MDS coded task with single-forking. The MDS coding implies that the task is fragmented into k subtasks and encoded into n coded subtasks, where completion of any k coded subtasks finishes the task. The single-forking implies that the task is started with n_0 coded subtasks at the task request time, and another n_1 coded subtasks are started on completion of $\ell_0 < k$ out of initial n_0 coded subtasks.

1) We first show that when execution times are either constant or *i.i.d.* random with bounded support and certain constraints, then the regime of $n_0 < k$ is not interesting.
2) We then explicitly compute the means of two performance metrics when the random execution time of each coded subtask is assumed to be *i.i.d.* with a shifted exponential distribution. This assumption is shown to be a decent approximation of service completion times on compute clusters [18], [36].
3) We compute the two performance metrics for the choice of system parameters n_0, n_1, and ℓ_0, and demonstrate the quantitative tradeoff between these two metrics. For comparison, we consider the no-forking case, when $n_0 = n$. We find there is no advantage to choose $n_0 < k$ for either of the metrics as compared to no-forking case. This is because the server utilization does not change with the value of n_0 when $n_0 < k$ while the task completion time increases as n_0 decreases. Thus, one should not perform forking with $n_0 < k$, and hence the only regime of interest is $n_0 \geq k$.
4) In this regime $n_0 \geq k$, we make the following observations. Keeping parameters ℓ_0 and n fixed, we observe that the mean server utilization is not monotone in the initial number of coded subtasks n_0, whereas the mean task completion time decreases with n_0 as expected. For a fixed n_0 and n, increasing the fork task threshold ℓ_0 increases the task completion time while decreases the server utilization. Thus, there is a tradeoff in the two metrics and efficient choice of parameters can be decided by the system designer based on the weighted combination of the two metrics.
5) We empirically studied two-forking for a single task with k subtasks encoded to n coded subtasks using MDS coding, where execution times are *i.i.d.* with shifted exponential distribution. We observed that the performance curve obtained by two-forking does not offer significant gains when compared to the single-forking for the choice of parameters we selected.
6) We also performed numerical studies for single and multiple-forking, when the execution time at individual servers has a heavy-tailed distribution. In particular, we chose execution time distributions to be Pareto and Weibull. We observed that insights derived from the analytical study of single-forking with shifted exponential distribution continues to hold in this case.
7) In addition to the analytical studies for the shifted exponential distribution, we also studied the impact of single and multiple-forking on a real compute cluster. We observe that the execution time of coded subtask at each server in the compute cluster can be well modeled by the shifted exponential distribution, and hence the insights obtained from the analytical studies continue to hold for this real compute cluster.

C. Organization

The rest of the paper is organized as follows. Section II describes the system model. In Section III, we present general methodology for analytical computation of performance metrics, and provide results for general execution time distribution. Section IV provides the analytical results for single forking point with shifted exponential distribution for execution time. In particular, we compute the performance metrics for the two cases of $n_0 < k$ and $n_0 \geq k$. Section V provides a performance tradeoff between the two metrics for single forking point, and the comparisons to two-forking are made in Section VI. Section VII provides the experimental results on a real compute cluster, Intel DevCloud. Empirical studies for heavy-tailed execution time distributions are provided in Appendix G. Section VIII concludes the paper, with directions for future work.

II. System Model

In this section, we describe the different components of the system model in detail. We consider a distributed compute system with n identical servers. We will use the following notations throughout this article. We denote the set of integers by $\mathbb{Z}$, the set of non-negative integers by $\mathbb{Z}_+$, the set of positive integers by $\mathbb{N}$, the set of first n consecutive positive integers by $[n]$, the set of reals by $\mathbb{R}$, and the set of non-negative reals by $\mathbb{R}_+$. For two real numbers a, b, the minimum is denoted by $a \wedge b$, and the maximum is denoted by $a \vee b$.

A. Coding Model

We assume that each compute task can be divided into k subtasks, which are encoded into n coded subtasks and sent to n distinct servers. We assume the tasks to be MDS coded, which implies that the coded subtask completion at any k out of these n distinct servers results in the completion of the original task.

B. Single-Fork Scheduling

We assume a single-fork scheduling, where a task starts at n_0 parallel servers at time $t_0 = 0$, and adds $n_1 = n - n_0$ servers at a random time instant t_1 corresponding to service completion time of the ℓ_0th coded subtask out of n_0 initial servers. The total task completion time is given by t_2 when the remaining coded subtasks at $\ell_1 = k - \ell_0$ servers are completed. Since we can't have more completions than the number of servers in service and the number of subtasks, we have $\ell_0 \leq n_0 \wedge k$ and $\ell_0 + \ell_1 = k \leq n$. We denote the service completion time of rth coded subtask in stage $i \in \{0, 1\}$ by $t_{i,r}$. Since each stage consists of ℓ_i service completions, we have $r \in \{0, \ldots, \ell_i\}$ such that $t_{i,0} = t_i$ and $t_{i,\ell_i} = t_{i+1,0} = t_{i+1}$. We denote the number of ON servers in the duration $[t_{i,r}, t_{i,r+1})$ of rth coded subtask completion in stage i by the number $N_{i,r}$.

C. Service Model

Each server $i \in [n]$ is assumed to have an *i.i.d.* random execution time T_i with distribution function F for each scheduled coded subtask on this server. Recent works [15], [24], [31], [37] suggest that a shifted exponential distribution is a good fit for modeling the service time distribution in distributed computation networks. It is suggested that the service time for each computation of coded subtask can be modeled by two aggregate components; a constant server start-time and a random memoryless component. These studies along with the goal of analytical tractability influenced us to assume the service time distribution for each coded subtask to be a shifted exponential with rate μ and shift c, such that the complementary distribution function $\bar{F} = 1 - F$ is

$$\bar{F}(x) \triangleq P\{T_1 > x\} = \mathbb{1}_{\{x \in [0,c]\}} + e^{-\mu(x-c)} \mathbb{1}_{\{x \geq c\}}. \quad (1)$$

D. Performance Metrics

The task completion time for k coded subtasks is denoted by S and the server utilization by W. Since, we are assuming a single-forking scenario, we have two contiguous stages. The time interval $[t_0, t_1)$ corresponds to the stage 0, and the interval $[t_1, t_2]$ corresponds to the stage 1. The task completion time is sum of the duration of two stages, and can be written as $S = t_2 = (t_2 - t_1) + (t_1 - t_0)$. The duration of stage i can be written as the following telescopic sum of duration of rth coded subtask completion in stage i,

$$t_{i+1} - t_i = \sum_{r=0}^{\ell_i - 1} (t_{i,r+1} - t_{i,r}). \quad (2)$$

The server utilization is measured in terms of the amount of work done by all servers that were on until the task completion. This utilization is the sum of the utilization in each of the two stages, written as $W = W_0 + W_1$. Assuming that a server is discarded after its coded subtask completion, we can write the server utilization in stage i as the time-integral of number of servers that are on in the ith stage duration $[t_i, t_{i+1})$. Since the number of on servers in the duration $[t_{i,r}, t_{i,r+1})$ is the constant number $N_{i,r}$, we can write the server utilization in stage i as

$$W_i = \sum_{r=0}^{\ell_i - 1} N_{i,r}(t_{i,r+1} - t_{i,r}). \quad (3)$$

We are interested in the optimal tradeoff between the mean task completion time $\mathbb{E}[S]$ and the mean server utilization $\mathbb{E}[W]$ for k coded subtasks scheduled in two stages over these n servers. To this end, we will analytically compute the mean task completion time and the mean server utilization, for a fixed number of servers n, as a function of choice of initial servers n_0 and threshold ℓ_0 on number of coded subtask completions for forking.

III. Computation of Performance Metrics

In this section, we will write the task completion time and the server utilization when coded subtask completion times at n servers are random and *i.i.d.* denoted by $T \triangleq (T_1, \ldots, T_n)$, with a common general distribution $F : \mathbb{R}_+ \to [0, 1]$. Recall that in stage 0, we switch on n_0 initial servers at instant $t_0 = 0$. This stage is completed at the single-forking point denoted by the instant t_1, when ℓ_0 coded subtasks out of n_0 are completed. At the beginning of stage 1, additional $n_1 = n - n_0$ servers are switched on, each working on a unique coded subtask. The task is completed at the end of this second stage denoted by instant t_2, when remaining $k - \ell_0$ coded subtasks are completed. At the instant t_i that indicates the beginning of stage i, the number of servers that are on is given by $N_{i,0}$, where $N_{0,0} = n_0$ and $N_{1,0} = n_0 - \ell_0$. Therefore, we can write $N_{i,0} = \sum_{j=0}^{i} (n_j - \ell_j) + \ell_i$. Further, in the duration $[t_i, t_{i,r+1})$, first r servers in stage i available from the time instant t_i have completed coded subtasks, and hence $N_{i,r} = N_{i,0} - r$. Therefore, the server utilization in stage i is

$$W_i = \sum_{r=0}^{\ell_i - 1} (t_{i,r+1} - t_{i,r}) \Big(\sum_{j=0}^{i} (n_j - \ell_j) + \ell_i - r \Big). \quad (4)$$

Rearranging terms and interchanging summation order in Eq. (4), we observe that the server utilization in stage i can be written

$$W_i = (t_{i+1} - t_i) \sum_{j=0}^{i} (n_j - \ell_j) + \sum_{r=1}^{\ell_i} (t_{i,r} - t_i).$$

Definition 1: We define a family of functions $f_n : \mathbb{R}^n \times [n] \to \mathbb{R}$ for each $n \in \mathbb{N}$ such that $f_n(x, k)$ is the kth order statistics of n real values $x = (x_1, \ldots, x_n)$.

Without loss of generality, we can assume that the initial n_0 coded subtasks are forked at first n_0 servers with execution times $(T_1, \ldots, T_{n_0})$. Then, the completion time of rth coded subtask in stage 0 is given by

$$t_{0,r} = f_{n_0}(T_1, \ldots, T_{n_0}, r). \quad (5)$$

We can write the duration of stage 0 as $t_1 - t_0 = t_1 = t_{0,\ell_0}$, and the server utilization in this stage as

$$W_0 = t_1(n_0 - \ell_0) + \sum_{r=1}^{\ell_0} t_{0,r}.$$

Remark 1: As the number of initial servers n_0 increases, the collection $(T_1, \ldots, T_{n_0})$ increases. For a fixed threshold ℓ_0 and any $r \in [\ell_0]$, the rth order statistics cannot increase as n_0 increases. Therefore, it follows that the completion time $t_{0,r}$ of rth coded subtask in stage 0 is a non-increasing function of the number of initial servers n_0 for a fixed threshold ℓ_0 and any $r \in [\ell_0]$.

Remark 2: For a fixed number of initial servers n_0, the forking instant $t_1 = t_{0,\ell_0}$ is a non-decreasing function of threshold ℓ_0.

Remark 3: We observe that the order statistics have the linear shift property, i.e. for any $x \in \mathbb{R}^n$ and $c \in \mathbb{R}$

$$f_n(c + x, k) = c + f_n(x, k).$$

Remark 4: We further observe that for a vector $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ such that $\sup_{i\in[m]} y_i \leq \inf_{j\in[n]} x_j$, then we have

$$f_n(x, k) = f_{n+m}(x, y, m + k).$$

From Remark 3 and Remark 4, it follows that the completion time of rth coded subtask in stage 1 is given by

$$t_{1,r} = f_n(T_1, \ldots, T_{n_0}, t_1 + T_{n_0+1}, \ldots, t_1 + T_n, \ell_0 + r). \quad (6)$$

We can write the duration of stage 1 as $t_2 - t_1 = t_{1,\ell_1} - t_1$, and the server utilization in this stage as

$$W_1 = (t_2 - t_1)(n - k) + \sum_{r=1}^{\ell_1} (t_{1,r} - t_1).$$

Remark 5: For a fixed threshold ℓ_0, the forking instant $t_1 = t_{0,\ell_0}$ is a non-increasing function of initial number of servers n_0. Therefore, it follows from Eq. (6) that $t_{1,r}$ is non-increasing function of n_0 for each $r \in [\ell_1]$. In particular, the task completion time $t_2 = t_{1,\ell_1}$ is non-increasing function of initial number of servers n_0.

Remark 6: Since t_1 is non-decreasing function of threshold ℓ_0 for a fixed number of initial servers n_0, it follows that $t_2 = t_{1,\ell_1}$ is also a non-decreasing function of threshold ℓ_0.

Above remarks imply that to minimize the service completion time S, the number of initial servers n_0 should be as large as possible, and the fork task threshold ℓ_0 should be as small as possible. Next, we focus on the impact of parameters n_0, ℓ_0 on the server utilization, which can be written as

$$W = t_2(n - k) + t_1(n_0 - n) + \sum_{r=1}^{\ell_0} t_{0,r} + \sum_{r=1}^{\ell_1} t_{1,r}. \quad (7)$$

The total server utilization consists of four terms. Except for the term $t_1(n_0 - n)$, the rest three terms are all non-increasing function of number of initial servers n_0. Therefore, in general, the monotonicity of the server utilization is not clear as a function of number of initial servers n_0 and coded subtask threshold ℓ_0.

We will show that when the service distribution satisfies certain properties, it is always beneficial to start with coded subtasks at $n_0 \geq k$ servers. We first consider the case when the subtask completion times are identically constant.

Lemma 1: When the subtask completion times are identically constant, the optimal number of initial servers is number of uncoded subtasks k.

Proof: The proof is provided in Appendix A. ■

We next consider the case of random execution times, where $c_1 \leq T_i \leq c_2$ for each server $i \in [n]$.

Theorem 1: Consider a distributed compute system with n servers and general coded subtask completion times $(T_1, \ldots, T_n)$, such that $T_i \in [c_1, c_2]$ for all $i \in [n]$ and the constants satisfy the following condition

$$\frac{c_2}{c_1} \leq \min\left\{2, \frac{(n-1)}{(k-1)}\right\}.$$

Then, for single-forking with $n_0 < k$ subtasks has higher server utilization and task completion time when compared to a distributed compute system with $n_0 = \ell_0 = k$ servers initialized at time $t_0 = 0$.

Proof: The proof is provided in Appendix B. ■

In this section, we observed that the behavior of server utilization depends on the distribution of execution times in general. However, when the execution time is constant, or the execution time is random with finite support satisfying certain conditions, we observed that the case $n_0 < k$ is not interesting. We will see that this observation continues to hold true for random variables with infinite support, for specific parameter values.

IV. Shifted Exponential Distribution

In this section, we derive the mean task completion time and the mean server utilization when the execution time distribution is shifted exponential with parameters (c, μ). Recall that the shifted exponential distribution for coded subtask completion times suggest that the execution time for each coded subtask computation consists of two components; a constant server start-time c and a random memoryless component of rate μ.

From Eq. (2) on the duration of stage i, and Eq. (4) on server utilization of stage i, we observe that they both depend on the intervals $[t_{i,r-1}, t_{i,r})$ for $r \in [\ell_i]$. The random variable $t_{i,r}$ is the rth coded subtask completion time in stage i and is related to order statistics of random variables. As such, we need the following two standard results on order statistics.

Remark 7 ([38]): Let $T \in \mathbb{R}^n$ be an *i.i.d.* random vector with common distribution function F, then the distribution of jth order statistic $f_n(T, j)$ of this collection is given by

$$P\{f_n(T, j) \leq x\} = \sum_{i=j}^{n} \binom{n}{i} F(x)^i \bar{F}(x)^{n-i}.$$

Remark 8: For the *i.i.d.* vector $T \in \mathbb{R}_+^n$ of (c, μ)-shifted exponential random execution times, we can define the shifted vector $T' \triangleq T - c$ such that the ith component is $T'_i = T_i - c$ for all $i \in [n]$. Then, the *i.i.d.* random vector T' is distributed exponentially with rate μ.

Remark 9 ([38]): We define $X_j^n \triangleq f_n(T', j)$ for all $j \in [n]$ and $X_0^n = 0$. From the memoryless property of T'_i, we observe the following equality in the joint distribution of two vectors

$$(X_j^n - X_{j-1}^n : j \in [n]) = \left(\frac{T'_j}{n - j + 1} : j \in [n]\right).$$

Recall that the task completion time can be written as the sum of durations of stage 0 and stage 1, and the server utilization can be decomposed into sum of the server utilization in each of the two stages. Accordingly, we will separately analyze these two stages in the following subsections.

A. Stage 0 Analysis

In this subsection, we would compute the mean of the interval $[t_{0,r-1}, t_{0,r})$ for each $r \in [\ell_0]$, and subsequently obtain the mean duration of the stage 0, and the mean server utilization in this stage.

Lemma 2: For the single-forking scheme with i.i.d. *shifted exponential coded subtask completion times, the mean time between two coded subtask completions in stage* 0 *is*

$$\mathbb{E}\left[t_{0,r} - t_{0,r-1}\right] = \begin{cases} c + \dfrac{1}{\mu n_0}, & r = 1, \\ \dfrac{1}{\mu(n_0 - r + 1)}, & r \in \{2, \ldots, \ell_0\}. \end{cases} \tag{8}$$

Proof: Since $t_{0,r}$ is the completion time of first r coded subtasks out of n_0 parallel coded subtasks, we have $t_{0,r} = c + X_r^{n_0}$ from Remark 8. Hence, for each $r \in [\ell_0]$, we have

$$t_{0,r} - t_{0,r-1} = (c + X_r^{n_0}) - (c + X_{r-1}^{n_0}).$$

The coded subtasks are initiated at time $t_{0,0} = t_0 = 0$ and hence the first coded subtask is completed at $t_{0,1} - t_{0,0} = X_1^{n_0}$. From Remark 9, we can write the following equality in distribution

$$t_{0,r} - t_{0,r-1} = \begin{cases} c + \dfrac{T_1'}{n_0}, & r = 1, \\ \dfrac{T_r'}{(n_0 - r + 1)}, & r \in \{2, \ldots, \ell_0\}, \end{cases}$$

where $(T_1', \ldots, T_n')$ are *i.i.d.* exponentially distributed random variables with rate μ. Taking expectations on both sides, we get the result. ∎

Corollary 1: For the single-forking scheme with i.i.d. *shifted exponential coded subtask completion times, the mean forking time is*

$$\mathbb{E}\left[t_1\right] = c + \sum_{r=1}^{\ell_0} \frac{1}{\mu(n_0 - r + 1)}, \tag{9}$$

and the mean server utilization in stage 0 *is given by*

$$\mathbb{E}\left[W_0\right] = \frac{1}{\mu}\ell_0 + cn_0. \tag{10}$$

Proof: Taking expectation on both sides of Eq. (2) and Eq. (4), we can find the mean of forking time and mean of server utilization respectively, as weighted sum of $\mathbb{E}\left[t_{0,r} - t_{0,r-1}\right]$ for $r \in [\ell_0]$. The result follows by substituting the mean time between consecutive coded subtask completions in stage 0, computed in Lemma 2. ∎

B. Stage 1 Analysis

In this subsection, we will compute the mean of the interval $[t_{1,r-1}, t_{1,r})$ for each $r \in [\ell_1]$, and subsequently obtain the mean duration of the stage 1, and the mean server utilization in this stage. In stage 0, all the n_0 servers are initialized at time $t_0 = 0$, and hence the start-up time for all n_0 servers in stage 0 is synchronized. This simplified the computation of mean time between consecutive coded subtask completions in stage 0.

However, this is not the case in stage 1 which begins at forking time t_1. At this instant, the remaining coded subtasks are initiated at additional n_1 servers. These additional servers have synchronized start at the forking time t_1, and all of them have the identical constant start-up time c. Therefore, none of them can finish executing before time $t_1 + c$. However, there are $n_0 - \ell_0$ remaining servers from stage 0, which were initialized at time $t_0 = 0$. These servers are not synchronized with the servers forked at time t_1. Specifically, any of these servers can finish executing in the interval $(t_1, t_1 + c]$. This complicates the computation of mean time between consecutive coded subtask completions in stage 1.

We denote the number of coded subtask completions in the interval $(t_1, t_1 + c]$ by a random variable $N_1(c)$ that takes values in $\{0, \ldots, n_0 - \ell_0\}$. The event of this random variable taking a value $j - \ell_0$ for any $j \in \{\ell_0, \ldots, n_0\}$ is denoted by

$$E_{j-\ell_0} \triangleq \{N_1(c) = j - \ell_0\}. \tag{11}$$

Since in the time interval $[0, t_1 + c]$ only servers that complete executing their coded subtasks are from initial n_0 servers, and ℓ_0 of them finish at time t_1, the event $E_{j-\ell_0}$ is the event of j completions before any of the additional n_1 servers enter their memoryless phase. Therefore, we can write

$$E_{j-\ell_0} = \left\{X_j^{n_0} \leq c + X_{\ell_0}^{n_0} < X_{j+1}^{n_0}\right\}. \tag{12}$$

In the following we would describe the duration between two consecutive coded subtask completions in stage 1 for three cases. First, when the completions occur before any of the stage 1 forked servers enter their memoryless phase. In this case, we can write the interval between rth and $(r-1)$th coded subtask completions for $r \in [N_1(c)]$ as

$$(t_{1,r} - t_{1,r-1}) = (X_r^{n_0-\ell_0} - X_{r-1}^{n_0-\ell_0}). \tag{13}$$

Second, the duration between completions just before and after the stage 1 forked servers enter their memoryless phase. There is a phase change at time $t_1 + c$, and we can write the difference between rth and $(r-1)$th coded subtask completion time instants for $r = N_1(c) + 1$, as the following telescopic sum of two durations

$$(t_{1,r} - t_{1,r-1}) = [(t_{1,r} - (t_1 + c)) + (t_1 + c - t_{1,r-1})]. \tag{14}$$

Third, the completion durations post the first completion after stage 1 forked servers enter their memoryless phase. Due to memoryless property, there are $n - \ell_0 - N_1(c)$ *i.i.d.* parallel memoryless servers executing from time $t_1 + c$. Therefore, we can write the interval between rth and $(r-1)$th coded subtask completions for $r \in \{N_1(c) + 2, \ldots, k - \ell_0\}$ as

$$(t_{1,r} - t_{1,r-1}) = (X_{r-N_1(c)}^{n-\ell_0-N_1(c)} - X_{r-N_1(c)-1}^{n-\ell_0-N_1(c)}). \tag{15}$$

We conclude that the duration between $(r-1)$th and rth coded subtask completions depends on, whether $r \in [N_1(c)]$, $r = N_1(c) + 1$, or $N_1(c) + 1 < r \leq k - \ell_0$. Therefore, we can

write the mean of this duration, in terms of the event indicators $(\mathbb{1}_{E_{j-\ell_0}} : \ell_0 \le j \le n_0 \wedge k)$, as

$$\begin{aligned}\mathbb{E}\left[(t_{1,r} - t_{1,r-1})\right] &= \sum_{j=\ell_0}^{n_0\wedge k} \mathbb{E}\Big[(t_{1,r} - t_{1,r-1})(\mathbb{1}_{\{r\le j-\ell_0\}} \\ &\quad + \mathbb{1}_{\{r=j-\ell_0+1\}} + \mathbb{1}_{\{r>j-\ell_0+1\}}) \mid E_{j-\ell_0}\Big] P(E_{j-\ell_0}).\end{aligned} \tag{16}$$

Thus, to evaluate the above mean, we first need to compute the probability mass function of the discrete valued random variable $N_1(c)$. We denote the probability of an exponential random variable being smaller than or equal to c by

$$\alpha \triangleq 1 - e^{-\mu c}. \tag{17}$$

That is, α is the probability that any remaining $n_0 - \ell_0$ servers initialized at time 0, finish executing their coded subtask in the interval $(t_1, t_1 + c]$.

Lemma 3: The probability distribution of the number of coded subtask completions $N_1(c)$ in the interval $(t_1, t_1 + c]$ for $\ell_0 \le j \le n_0$ is given by $p_{j-\ell_0} \triangleq P(E_{j-\ell_0})$ where

$$p_{j-\ell_0} = \binom{n_0 - \ell_0}{j - \ell_0} \alpha^{j-\ell_0} (1-\alpha)^{(n_0 - j)}, \tag{18}$$

and α is defined in Eq. (17)*.*

Proof: The detailed proof is provided in Appendix C. ■

In the following, we will find the conditional mean for the duration between coded subtask completions in Stage 1. For $m \in \{0, \ldots, (n_0 \wedge k) - \ell_0\}$, we compute the conditional mean

$$\mathbb{E}\left[(t_{1,r} - t_{1,r-1}) \mid E_m\right], \quad r \in [k - \ell_0]. \tag{19}$$

We next provide a definition that is needed to compute this conditional mean.

Definition 2: We denote the Pochhammer function $(a)_n \triangleq \frac{\Gamma(a+n)}{\Gamma(a)}$ to define the z-transform of hypergeometric series as

$${}_pF_q(z) \triangleq {}_pF_q\left[\begin{matrix}a_1, \ldots, a_p \\ b_1, \ldots, b_q\end{matrix}; z\right] = \sum_{n=0}^{\infty} \frac{\prod_{i=1}^{p} (a_i)_n z^n}{\prod_{j=1}^{q} (b_j)_n (n)!}. \tag{20}$$

Because generalizations of the above series also exist [39], this series is referred to here as the hypergeometric series rather than the generalized hypergeometric series.

As shown in Eq. (16), we need to compute three different conditional means to compute the mean $\mathbb{E}[t_{1,r} - t_{1,r-1}]$. Following three propositions provide the conditional means for the cases when $r \le j - \ell_0$, $r = j - \ell_0 + 1$, and $r > j - \ell_0 + 1$. The proofs are provided in Appendix E.

Proposition 1 (Phase-1): Let $m \in \{0, \ldots, (n_0 \wedge k) - \ell_0\}$, $\alpha = 1 - e^{-\mu c}$, and the z-transform ${}_pF_q(z)$ of hypergeometric series defined in Eq. (20)*. Then, for any $r \in [m]$, we have*

$$\mathbb{E}\left[(t_{1,r} - t_{1,r-1}) | E_m\right] = {}_2F_1\left(\begin{matrix}1, r \\ m+2\end{matrix}; \alpha\right) \frac{r\alpha}{\mu(m+1)}. \tag{21}$$

Proposition 2 (Phase-2): For $r = j - \ell_0$ and $\alpha = 1 - e^{-\mu}$, we have

$$\mathbb{E}\left[(t_{1,r+1} - t_{1,r}) | E_{j-\ell_0}\right] = \frac{c}{\alpha^r} - \sum_{i=1}^{r} \frac{\alpha^{i-r}}{i\mu} + \frac{1}{\mu(n-j)}. \tag{22}$$

Proposition 3 (Phase-3): For $r > j - \ell_0$ and $\alpha = 1 - e^{-\mu c}$, we have

$$\mathbb{E}\left[(t_{1,r+1} - t_{1,r}) | E_{j-\ell_0}\right] = \frac{1}{\mu(n - \ell_0 - r)}. \tag{23}$$

We can now compute the unconditional mean of the interval between $(r-1)$th and rth coded subtask completion times in Stage 1. This unconditional mean can be computed by substituting the conditional means computed for three different phases in Proposition 1, Proposition 2, Proposition 3, and probability mass function of $N_1(c)$ computed in Lemma 3, in Eq. (16).

Corollary 2: The mean duration between $(r-1)$th and rth completion of coded subtasks in Stage 1 is given by

$$\begin{aligned}\mathbb{E}\left[t_{1,r} - t_{1,r-1}\right] &= \sum_{m=0}^{(n_0\wedge k)-\ell_0} p_m \Big[\frac{1}{\mu(n - \ell_0 - r + 1)} \mathbb{1}_{\{r>m\}} \\ &\quad + \Big(\frac{c}{\alpha^{r-1}} - \sum_{i=1}^{r-1} \frac{\alpha^{i-r+1}}{i\mu} + \frac{1}{\mu(n-j)}\Big) \mathbb{1}_{\{r=m\}} \\ &\quad + {}_2F_1\Big(\begin{matrix}1, r \\ m+2\end{matrix}; \alpha\Big) \frac{r\alpha}{\mu(m+1)} \mathbb{1}_{\{r\le m\}}\Big].\end{aligned}$$

C. Performance Metrics Computation

Since we have defined the unconditional mean for the duration between two coded subtask completions in both Stage 0 and Stage 1, we can now compute the means of task completion time and server utilization. We saw in Section III that when the service distribution satisfies certain properties, it is always beneficial to start with coded subtasks at $n_0 \ge k$ servers. We will show that this continues to hold true for shifted exponential distribution, even though it doesn't satisfy any of those certain properties. Accordingly, we consider both the possibilities $n_0 < k$ and $n_0 \ge k$.

1) Case $n_0 < k$: Recall that fork task threshold $\ell_0 \le n_0 < k$ for this case. That is, n_0 initial servers can never complete the k coded subtasks, and hence $t_2 > t_1 + c$ almost surely. We now compute the mean task completion time and mean task utilization for $n_0 < k$ case.

Theorem 2: For the single-forking of a single task on n servers, with the initial number of servers $n_0 < k$, the mean server utilization is

$$\mathbb{E}[W] = nc + \frac{k}{\mu}. \tag{24}$$

The mean task completion time for this case is

$$\mathbb{E}[t_2] = c + \mathbb{E}[t_1] + \frac{1}{\mu} \sum_{j=\ell_0}^{n_0} p_{j-\ell_0} \sum_{i=j}^{k-1} \frac{1}{(n-i)}, \tag{25}$$

where $\mathbb{E}[t_1]$ is given in Eq. (9) *and $p_{j-\ell_0}$ is given in Eq.* (18)*.*

Proof: The detailed proof is provided in Appendix F. ■

Remark 10: We make the following three observations.

1) From Theorem 2, it follows that the mean server utilization $\mathbb{E}[W]$ remains unchanged for all values of initial number of servers $n_0 < k$ and fork task threshold ℓ_0.

2) From Remark 5, it follows that the mean task completion time $\mathbb{E}[t_2]$ is a non-increasing function of the number of initial servers n_0.
3) From Remark 6, it follows that the mean task completion time $\mathbb{E}[t_2]$ is a non-decreasing function of the fork task threshold ℓ_0.

From the above observations, we conclude that the joint best choices for (n_0^*, ℓ_0^*) are $(k-1, 1)$, when $n_0 < k$.

Remark 11: Consider the no-forking case when $n_0 = n$, i.e. when we initialize all n coded subtasks at unique servers at time 0. From Remark 5, the mean task completion time is lowest for this case, among all choices of n_0. Further, for all coded subtasks $r \in [k]$, we have $t_{1,r} - t_{1,r-1} = X_r^n - X_{r-1}^n$. From Remark 9, it follows that

$$\mathbb{E}[t_{1,r} - t_{1,r-1}] = \frac{1}{\mu(n-r+1)}, \quad r \in [k].$$

Therefore, the mean server utilization is $nc + \frac{k}{\mu}$, identical to that for the case when $n_0 < k$.

Thus, as compared to no-forking, a single-forking with $n_0 < k$ has the same mean server utilization while it has higher mean task completion time. Therefore, we focus only on the case when $n_0 \geq k$.

2) Case $n_0 \geq k$: In this case, the initial number of servers n_0 is always greater than the required number of coded subtasks k. Hence, the number of completed coded subtasks $\ell_0 \in \{0, \ldots, k\}$ at the forking point t_1. The mean task completion time and the mean server utilization for $n_0 \geq k$, are given in the following theorem.

Theorem 3: For the single-forking of a single task on n servers, with the initial number of servers $n_0 \geq k$, the mean task completion time $\mathbb{E}[t_2]$ is

$$\mathbb{E}[t_1] + \left[\sum_{r=1}^{k-\ell_0} \mathbb{E}[t_{1,r} - t_{1,r-1}]\right] \mathbb{1}_{\{\ell_0 < k\}}. \tag{26}$$

The mean server utilization $\mathbb{E}[W]$ for this case, is

$$\mathbb{E}[W_0] + \sum_{r=1}^{k-\ell_0} (n - \ell_0 - r + 1)\mathbb{E}[t_{1,r} - t_{1,r-1}]\, \mathbb{1}_{\{\ell_0 < k\}}. \tag{27}$$

The mean duration $\mathbb{E}[t_{1,r} - t_{1,r-1}]$ is given in Corollary 2. The mean forking time $\mathbb{E}[t_1]$ and the mean server utilization $\mathbb{E}[W_0]$ in Stage 0 are given in Corollary 1.

Proof: Recall that since completion of any k coded subtasks suffice for the task completion, the fork task threshold $\ell_0 \leq k$. When fork task threshold $\ell_0 = k$, all the required k coded subtasks are finished on initial n_0 servers without the need of any forking. In this case,

$$t_2 = t_1 \mathbb{1}_{\{\ell_0 = k\}}, \quad W = W_1 \mathbb{1}_{\{\ell_0 = k\}}. \tag{28}$$

When fork task threshold $\ell_0 < k$, the task completion occurs necessarily in Stage 1. Therefore, we have

$$t_2 = (t_1 + (t_2 - t_1))\mathbb{1}_{\{\ell_0 < k\}}, \quad W = (W_1 + W_2)\mathbb{1}_{\{\ell_0 < k\}}.$$

The result follows from Eq. (2) for the duration $t_2 - t_1$ of Stage 1 and Eq. (4) for server utilization W_2 in Stage 1. ■

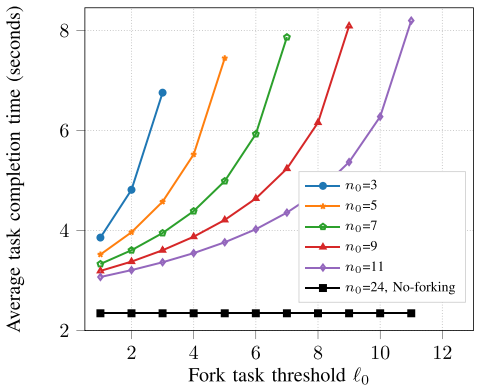

Fig. 2. The empirical mean of task completion time as a function of fork task threshold $\ell_0 \in [n_0]$ for single-forking, when the number of initial servers $n_0 < k = 12$.

Remark 12: We observe that when $n_0 \geq k$, the mean server utilization depends on the initial number of servers n_0 as well as the total number of servers n, unlike the case $n_0 < k$ where this utilization depends only on the total number of servers n.

In the following section, we numerically investigate the tradeoff between the two performance metrics, which allows us the proper choice of system parameters to work in a specified regime.

V. Numerical Studies: Single-Forking

In this section, we evaluate the empirical performance of single-forking systems, by Monte Carlo methods [40]. We simulated multiple instances for a single task, with subtask fragmentation $k = 12$ and a maximum redundancy factor of $n/k = 2$. That is, we choose the total number of servers $n = 24$. Coded subtask completion time at each server was chosen to be an *i.i.d.* random variable having a shifted exponential distribution, with the shift parameter $c = 1$ and the exponential rate $\mu = 0.5$. For these values of system parameters, we computed empirical average of task completion times and server utilizations. We compare the two cases $n_0 < k$ and $n_0 \geq k$ with no-forking case $n_0 = n$, where all the available servers are used as initial servers.

We first study the case when $n_0 < k$. For this case, we plot the empirical mean of task completion time as a function of fork task threshold $\ell_0 \in [n_0]$ in Fig. 2, for different values of initial servers $n_0 \in \{3, 5, 7, 9, 11\}$. The analytical results in Theorem 2 are substantiated, by observing that the empirical mean of task completion time is increasing with fork task threshold ℓ_0 and decreasing with initial number of servers n_0. In this case, the empirical mean of server utilization is constant for any choice of n_0 and ℓ_0, and hence it is not depicted. From Fig. 2, we infer that as compared to the no-forking case of $n_0 = n$, the case of single-forking with $n_0 < k$ has higher average task completion time for the *same* average server utilization. Thus, the regime $n_0 < k$ is not interesting for practical applications.

We next study the case when $n_0 \geq k$. For this case, we plot the empirical means of task completion time in Fig. 3 and server utilization in Fig. 4, both as a function of fork task threshold $\ell_0 \in [k]$, for different values of initial servers $n_0 \in \{12, 13, 14, 16, 18, 20\}$. The analytical results in Theorem 3 are substantiated by observing that the empirical mean of

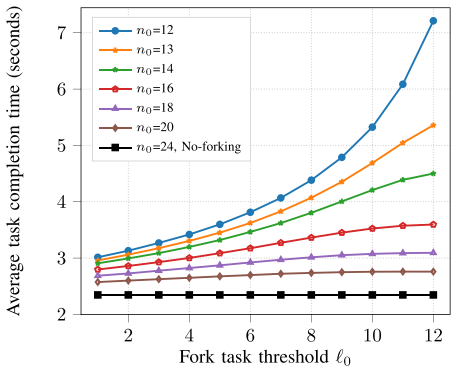

Fig. 3. The empirical mean of task completion time as a function of fork task threshold $\ell_0 \in [k]$ for single-forking, when the number of initial servers $n_0 \geq k = 12$.

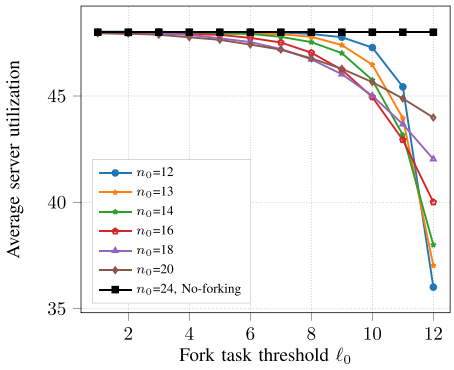

Fig. 4. The empirical mean of server utilization as a function of fork task threshold $\ell_0 \in [k]$ for single-forking, when the number of initial servers $n_0 \geq k = 12$.

task completion time increases with fork task threshold ℓ_0 and decreases with initial number of servers n_0. Further, the empirical mean of server utilization decreases with fork task threshold ℓ_0. As expected, we observe that the mean server utilization is highest for no forking when the number of initial servers $n_0 = n$, i.e., this curve is above all the other curves corresponding to $n_0 < n$. Further, when the fork task threshold $\ell_0 = k$, then the mean server utilization is monotonically smaller in the number of initial servers. However, we observe that the different server utilization curves cross when initial servers $n_0 < n$. This implies that the lower number of initial servers doesn't necessarily imply that the mean server utilization is uniformly smaller for all fork task thresholds. For this case, the mean server utilization curves cross for different fork task thresholds. Therefore, for each fork task threshold ℓ_0, the mean server utilization is not a monotone function of n_0, there exists an optimal number of initial servers n_0 that minimizes the mean server utilization.

From Fig. 3 for average task completion time and Fig. 4 for average server utilization, we conclude that there is a tradeoff between the two performance measures as a function of fork task threshold ℓ_0. The single forking tradeoff between the two performance metrics of interest is plotted in Fig. 5, for different number of initial servers $k \leq n_0 \leq n$. As discussed in Section I, we observe that the task completion time is minimized when all the n servers are initialized at time 0, i.e. $n_0 = n$. This corresponds to no-forking case and has the highest server utilization. Further, for any single forking

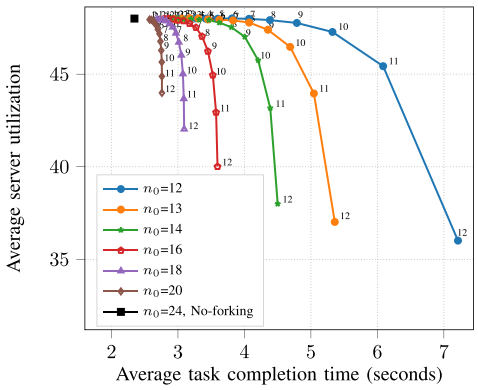

Fig. 5. The empirical mean of server utilization as a function of the empirical mean of task completion time for single-forking by increasing fork task threshold $\ell_0 \in [k]$ from left to right. The different curves correspond to different values of initial servers $n_0 \in \{12, 13, 14, 16, 18, 20\}$, where $k = 12$.

with $n_0 < n$ and for task threshold $\ell_0 < k$, the mean server utilization remains same to no-forking case, and the mean task completion time increases. Therefore, we only focus on the case when the initial number of servers $n_0 \in \{k, \ldots, n\}$. the fork task threshold $\ell_0 \in \{0, 1, \ldots, (n_0 \wedge k)\}$. The Fig. 5 suggests that for the number of initial servers $n_0 < n$, there is a tradeoff between the task completion time and the server utilization for different fork task thresholds $\ell_0 \in [k]$.

We next compare the single-forking case of $k \leq n_0 < n$ and the no-forking case of $n_0 = n$. From Fig. 5, we observe that the average task completion time increases only 17.635% while the average server utilization can be decreased 8.3617% taking $\ell_0 = k$, for $n_0 = 20$ when compared to $n_0 = n$. However, we see that average server utilization can't be reduced below 8.3617% for $n_0 = 20$, for any choice of fork task threshold $\ell_0 \leq k$. In order to have further reduction in average server utilization, one can choose the number of initial servers $n_0 = 18$. This choice of $n_0 = 18$ reduces the average server utilization by 12.43% at an expense of 31.888% increase in average task completion time, when compared to the no-forking case. The intermediate points on the performance curve for $n_0 = 18$ provide additional tradeoff points that can be chosen based on the desired combination of the two metrics as required by the system designer. One can further reduce the average server utilization by choosing lower values of n_0, until we reach the lowest possible choice of $n_0 = k$. This choice reduces average server utilization by 24.976% by having 207.49% increase in the average task completion time, as compared to the no-forking case.

Thus, we see that appropriate choice of n_0 and ℓ_0 provide tradeoff points that help reduce the average server utilization at the expense of increased average task completion time. We note that the insights obtained from this single-forking study for service time requirements having shifted exponential distribution, continue to hold when distributions are heavy-tailed. The corresponding empirical results for single-forking with heavy-tailed service time requirements are presented for Pareto distribution in Appendix G-A, and for Weibull distribution in Appendix G-B. Interestingly for these distributions, $n_0 < k$ is also an interesting regime for certain choice of distribution parameters.

VI. Numerical Studies: Tradeoff Between the Metrics

Consider a multi-forking scenario, under the limitation of finite number of servers n per compute task. That is, for an m-forking case, the free variables are fork task threshold sequence $(\ell_0, \ldots, \ell_m)$ and number of forked servers sequence $(n_0, \ldots, n_m)$, such that $\sum_{i=0}^{m} \ell_i = k$ and $\sum_{i=0}^{m} n_i \leq n$. The single-forking results in Theorem 3 suggest that, the tradeoff between the two performance metrics (the mean task completion time $\mathbb{E}[S]$ and the mean server utilization $\mathbb{E}[W]$), should continue to exist for the general case of multiple-forking points. However, the selection of fork task threshold sequence and the corresponding number of forked servers sequence, for a better tradeoff between these two performance metrics, is a multi-dimensional optimization problem and not easy to evaluate.

To understand multi-forking, we can empirically compute the means of two performance metrics for a given choice of fork task threshold sequence and forked server sequence. We then find performance points for all such choices, and take the lower envelope of all performance points to obtain the optimal performance curve. Formally, finding the lower envelope of this performance curve is equivalent to solving the following integer programming problem for all tradeoff parameters β,

$$\begin{aligned} \min_{(n_0,\ldots,n_m),(\ell_0,\ldots,\ell_m)} \quad & \mathbb{E}[S] + \beta \mathbb{E}[W] \\ \text{s.t.} \quad & (n_0, \ldots, n_m) \in \mathbb{Z}_+^{m+1}, \quad (\ell_0, \ldots, \ell_m) \in \mathbb{Z}_+^{m+1}, \\ & \text{and} \quad \ell_i \leq \sum_{j=0}^{i} n_j \quad \text{for all } i \in [m-1], \\ & \text{and} \quad \sum_{i=0}^{m} n_i \leq n, \quad \sum_{i=0}^{m} \ell_i = k. \end{aligned} \tag{29}$$

For the system simulation, we take the shifted exponential distribution for the coded sub-task execution time, with shift $c = 1$ and rate $\mu = 0.5$. For a single task, we chose the number of subtasks $k = 12$, that is encoded to $n = 24$ coded subtasks. In order to perform the optimization over discrete number of choices, we compute the objective for each feasible choice of constraints for a given β, and then choose the best parameters.

We plot the optimal tradeoff between empirical means of task completion time and server utilization for two-forking in Fig. 6. In the same figure, we also plot the performance curve of single-forking for comparison. We observe that the performance curve for two-forking is only marginally below the corresponding curve for single-forking. Further, the figure illustrates the tradeoff between the metrics, even for single forking.

An investigation of optimal forking points and the corresponding sequence for number of forked servers is an important future research direction. The insights obtained from the multi-forking study when service time requirements for each coded subtask has a shifted exponential distribution, continues to hold when the distribution is heavy-tailed. The supporting numerical results for multi-forking with Pareto distribution is presented in Appendix G-A, and with Weibull distribution in Appendix G-B.

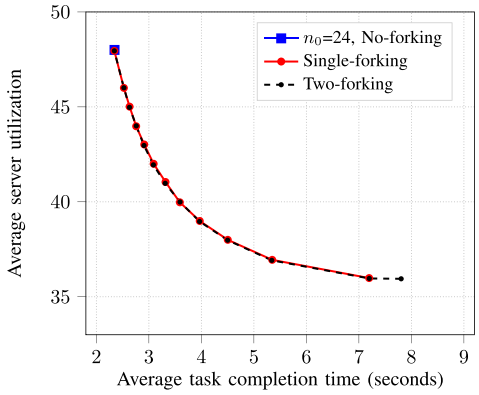

Fig. 6. The optimal tradeoff between empirical means of task completion time and server utilization, for both single-forking and two-forking.

VII. Experiments on Intel DevCloud Servers

To validate our findings, we performed experiments on a real compute cluster called Intel DevCloud. In particular, the objective of our experiment was to answer the following questions.

1) Is it possible to get a tradeoff between the average server utilization and average task completion time on real cloud setup with our forking mechanism?
2) Are the tradeoff curves for this practical setup qualitatively similar to the one predicted by the analytical study, i.e. can the distribution of random execution times be modeled as a shifted exponential?

Intel DevCloud is a cloud computing platform made available by Intel [41] for several profiles of researchers, students and professional engineers.[1] Intel DevCloud cluster consists of compute nodes, storage servers, and the login node. Each node has Intel Xeon processor of the Skylake architecture (Intel Xeon Scalable Processors family), an Intel Xeon Gold 6128 CPU, on-platform memory of 192 GB and a Gigabit Ethernet interconnect.

A. Setup

We performed single-forking and two-forking experiments on this compute cluster. A single task was divided into $k = 12$ subtasks, and encoded to $n = 24$ coded subtasks, such that execution of any k out of n coded subtasks suffice for the task completion. Specifically, we considered the coded subtask to be the addition of 6×10^9 positive integers.

In our experiment, we reserved one node per coded subtask. The empirical average of completion times for these coded subtasks was roughly 600 seconds on each executing node. We implemented both single-forking and two-forking. As soon as a coded subtask is completed, we logged that time stamp into a log file. At each forking point, multiple coded subtasks are synchronously initialized at unique nodes. This is achieved by explicitly requesting one distinct compute node per coded subtask. This ensures that all forked coded subtasks start at

[1]The authors would like to thank Intel for giving us access to the cluster for this project.

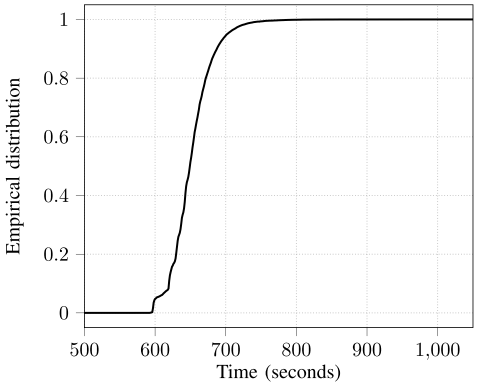

Fig. 7. The empirical distribution of coded subtask completion time, obtained from the Intel DevCloud experiments.

the same forking instant on all compute nodes, to which the coded subtasks are forked.

B. Evaluations

Using the observed coded subtask completion times for each run, we can compute the duration between two consecutive completions of coded subtasks $(t_{i,r} - t_{i,r-1} : r \in [\ell_i], i \in [m])$. Together with the sequence of number of forked servers $(n_0, \ldots, n_m)$ at each forking time, we can compute the task completion time from Eq. (2), and the server utilization from Eq. (4). We repeat this experiment $J = 1 \times 10^4$ times, and for each run $j \in [J]$, we record

(a) the task completion time denoted by $S^{(j)}$,
(b) the server utilization denoted by $W^{(j)}$, and
(c) the empirical distribution of coded subtask completion time at first k finishing nodes.

Subsequently, we compute the empirical average of these records over all J runs. The empirical mean of task completion time is denoted by $\hat{S} = \frac{1}{J}\sum_{j=1}^{J} S^{(j)}$, and the empirical mean of server utilization is denoted by $\hat{W} = \frac{1}{J}\sum_{j=1}^{J} W^{(j)}$.

C. Results

We first plot the empirical distribution of task completion time in Fig. 7. We observe that the empirical distribution of the coded subtask execution times at each node has characteristics of a shifted exponential distribution. The empirical distribution has a distinct constant shift corresponding to the start delay, and the random part of the task execution time has a light tail. Using QQplots [42], [43], we verify in Appendix H that the shifted exponential is a good fit for the empirical distribution of execution times. Since the qualitative behavior of execution time distribution at compute nodes resembles a shifted exponential, we expect our derived insights to continue to hold for this compute cluster. From the empirically computed means of two performance metrics, server utilization and task completion time, we can find the lower envelope of all performance points using Eq. (29), to obtain the optimal performance tradeoff curve between these two metrics. The optimal performance tradeoff curve is plotted in Fig. 8. We observe that the performance curve for two-forking is not significantly different than that for single-forking. Further, we corroborate the analytical results of single-forking obtained in Theorem 3, by observing that

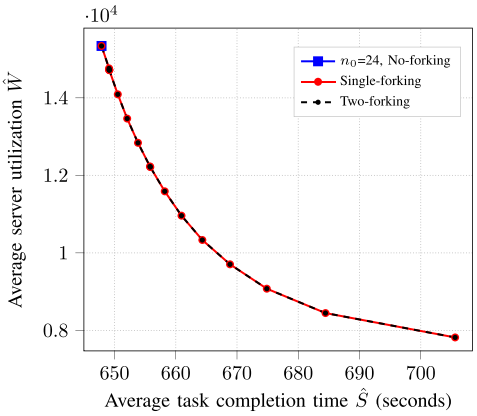

Fig. 8. The optimal tradeoff between the empirical means of server utilization and task completion time for single-forking and two-forking, when the coded subtasks are executed on compute nodes at Intel DevCloud.

the empirical mean of task completion time $\hat{S}$ increases with increase in fork task threshold ℓ_0. In addition, the tradeoff suggests that the initial number of servers n_0 is an important consideration for an efficient system design.

VIII. Conclusion

We study the single-forking for a single task that can be divided into k subtasks to be computed over n servers, in two stages. We assume that k subtasks can be coded into n computation coded subtasks using (n, k)-MDS coding, such that completion of any k coded subtasks lead to completion of the entire task. We assume that only n_0 out of n servers are started at time 0. After completion of ℓ_0 out of n_0 servers, remaining $n - n_0$ servers are initiated. Using the shifted exponential service times of the servers, we derive expressions for two performance metrics: (i) the mean task completion time which indicates the mean time when k servers have finished execution of coded subtasks, and (ii) the mean server utilization which indicates the aggregate mean of times each server is busy processing the coded subtasks. Further discussions of the results and future work directions can be seen in Appendix I.

References

[1] A. Badita, P. Parag, and V. Aggarwal, "Sequential addition of coded subtasks for straggler mitigation," in *Proc. IEEE Conf. Comput. Commun.*, Jul. 2020, pp. 746–755.
[2] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, vol. 13, 2013, pp. 185–198.
[3] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf, "Queueing with redundant requests: Exact analysis," *Queueing Syst.*, vol. 83, nos. 3–4, pp. 227–259, Aug. 2016.
[4] A. Badita, P. Parag, and V. Aggarwal, "Optimal server selection for straggler mitigation," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 709–721, Apr. 2020.
[5] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 3, pp. 7–11, Nov. 2015.
[6] D. Wang, G. Joshi, and G. W. Wornell, "Efficient straggler replication in large-scale parallel computing," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 4, no. 2, pp. 1–23, Jun. 2019.
[7] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.
[8] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Proc. Int. Workshop Peer Peer Syst. (IPTPS)*. Berlin, Germany: Springer-Verlag, 2002, pp. 328–338.

[9] M. Sathiamoorthy *et al.*, "XORing elephants: Novel erasure codes for big data," *Proc. VLDB Endow.*, vol. 6, no. 5, pp. 325–336, Mar. 2013.

[10] C. Huang *et al.*, "Erasure coding in windows azure storage," in *Proc. Annu. Tech. Conf. (ATC)*, Jun. 2012, pp. 15–26.

[11] A. Fikes. (2010). *Storage Architecture and Challenges*. Google Faculty Summit. Accessed: Apr. 20, 2021. [Online]. Available: https://bit.ly/3eh5ym3

[12] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, vol. 70, 2017, pp. 3368–3376.

[13] K. Lee, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Coded computation for multicore setups," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2413–2417.

[14] M. Ye and E. A. Abbe, "Communication-computation efficient gradient coding," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 5606–5615.

[15] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

[16] S. Dutta, V. Cadambe, and P. Grover, "'Short-Dot': Computing large linear transforms distributedly using coded short dot products," *IEEE Trans. Inf. Theory*, vol. 65, no. 10, pp. 6171–6193, Oct. 2019.

[17] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," *IEEE Trans. Inf. Theory*, vol. 66, no. 3, pp. 1920–1933, Mar. 2020.

[18] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing latency for secure coded computing using secret sharing via staircase codes," *IEEE Trans. Commun.*, vol. 68, no. 8, pp. 4609–4619, Aug. 2020.

[19] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. London, U.K.: Pearson, 2004.

[20] R. Jinan, A. Badita, P. Sarvepalli, and P. Parag, "Latency optimal storage and scheduling of replicated fragments for memory-constrained servers," 2020, *arXiv:2010.01589*. [Online]. Available: https://arxiv.org/abs/2010.01589

[21] D. Burihabwa, P. Felber, H. Mercier, and V. Schiavoni, "A performance evaluation of erasure coding libraries for cloud-based data stores," in *Proc. FIP Int. Conf. Distrib. Appl. Interoperable Syst.* Springer, 2016, pp. 160–173.

[22] A. Badita, P. Parag, and J.-F. Chamberland, "Latency analysis for distributed coded storage systems," *IEEE Trans. Inf. Theory*, vol. 65, no. 8, pp. 4683–4698, Aug. 2019.

[23] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 989–997, May 2014.

[24] Y. Xiang, T. Lan, V. Aggarwal, and Y.-F.-R. Chen, "Joint latency and cost optimization for erasure-coded data center storage," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2443–2457, Aug. 2016.

[25] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Trans. Inf. Theory*, vol. 64, no. 1, pp. 109–128, Jan. 2018.

[26] K. Wan, D. Tuninetti, M. Ji, and P. Piantanida, "Fundamental limits of distributed data shuffling," in *Proc. 56th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2018, pp. 662–669.

[27] M. A. Attia and R. Tandon, "Near optimal coded data shuffling for distributed learning," *IEEE Trans. Inf. Theory*, vol. 65, no. 11, pp. 7325–7349, Nov. 2019.

[28] L. Song, C. Fragouli, and T. Zhao, "A pliable index coding approach to data shuffling," *IEEE Trans. Inf. Theory*, vol. 66, no. 3, pp. 1333–1353, Mar. 2020.

[29] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" *IEEE Trans. Commun.*, vol. 64, no. 2, pp. 715–722, Feb. 2016.

[30] Y. Xiang, T. Lan, V. Aggarwal, and Y. F. R. Chen, "Joint latency and cost optimization for erasurecoded data center storage," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 2, pp. 3–14, Sep. 2014.

[31] A. O. Al-Abbasi and V. Aggarwal, "Video streaming in distributed erasure-coded storage systems: Stall duration analysis," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1921–1932, Aug. 2018.

[32] W. Wang, M. Harchol-Balter, H. Jiang, A. Scheller-Wolf, and R. Srikant, "Delay asymptotics and bounds for multitask parallel jobs," *Queueing Syst.*, vol. 91, nos. 3–4, pp. 207–239, Apr. 2019.

[33] A. O. Al-Abbasi, V. Aggarwal, and T. Lan, "TTLoC: Taming tail latency for erasure-coded cloud storage systems," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 4, pp. 1609–1623, Dec. 2019.

[34] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[35] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, and R. Katz, "Multi-task learning for straggler avoiding predictive job scheduling," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 3692–3728, 2016.

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

[37] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing latency for secure distributed computing," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Germany, Jun. 2017, pp. 2900–2904.

[38] S. S. Wilks, "Order statistics," *Bull. Amer. Math. Soc.*, vol. 54, no. 1, pp. 6–50,1948.

[39] G. Gasper, M. Rahman, and G. George, *Basic Hypergeometric Series*, vol. 96. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[40] R. Y. Rubinstein and D. P. Kroese, *Simulation and the Monte Carlo Method*, vol. 10. Hoboken, NJ, USA: Wiley, 2016.

[41] *Intel Devcloud*. Accessed: Oct. 6, 2020. [Online]. Available: https://software.intel.com/en-us/devcloud

[42] M. B. Wilk and R. Gnanadesikan, "Probability plotting methods for the analysis for the analysis of data," *Biometrika*, vol. 55, no. 1, pp. 1–17, 1968.

[43] J. I. Marden, "Positions and QQ plots," *Stat. Sci.*, vol. 19, no. 4, pp. 606–614, Nov. 2004.

[44] P. J. Davis, "Leonhard Euler's integral: A historical profile of the gamma function: In memoriam: Milton Abramowitz," *Amer. Math. Monthly*, vol. 66, no. 10, pp. 849–869, Dec. 1959.

[45] S. M. Ross, *Introduction to Probability Models*, 12th ed. New York, NY, USA: Academic, 2019.

[46] *Scipy Documentation*. Accessed: Aug. 2, 2021. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.rv_continuous.fit.html#scipy.stats.rv_continuous.fit

[47] H. Albrecher, J. Beirlant, and J. L. Teugels, *Reinsurance: Actuarial and Statistical Aspects*. Hoboken, NJ, USA: Wiley, 2017.

Ajay Badita (Graduate Student Member, IEEE) received the B.Tech. degree in electronics and communication engineering from JNTU Kakinada in 2011 and the M.Tech. degree in electronics and communication engineering from NIT Rourkela in 2015. He is currently pursuing the Ph.D. degree with the ECE Department, Indian Institute of Science, Bengaluru. His research interests include delay-sensitive communication, compute, and storage in distributed systems.

Parimal Parag (Senior Member, IEEE) received the B.Tech. and M.Tech. degrees in electrical engineering from IIT Madras in 2004, and the Ph.D. degree in electrical engineering from Texas A&M University in 2011. From 2011 to 2014, he was a Senior System Engineer (research and development) at Assia Inc., Redwood City. He is currently an Associate Professor with the ECE Department, Indian Institute of Science. He was a coauthor of the 2018 IEEE ISIT Student Best Paper. He was a recipient of the 2017 Early Career Award from the Science and Engineering Research Board.

Vaneet Aggarwal (Senior Member, IEEE) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 2005, and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, USA, in 2007 and 2010, respectively. He is currently an Associate Professor with Purdue University, West Lafayette, IN, where he has been since January 2015. From 2010 to 2014, he was a Senior Member of Technical Staff Research at AT&T Labs-Research, NJ. From 2013 to 2014, he was an Adjunct Assistant Professor with Columbia University, NY, USA. From 2018 to 2019, he was a VAJRA Adjunct Professor with IISc Bengaluru. His current research interests are in communications and networking, cloud computing, and machine learning.