Efficient Replication for Fast and Predictable Performance in Distributed Computing

Amir Behrouzi-Far and Emina Soljanin, *Fellow, IEEE*

***Abstract*— Master-worker distributed computing systems use task replication to mitigate the effect of slow workers on job compute time. The master node groups tasks into batches and assigns each batch to one or more workers. We first assume that the batches do not overlap. Using majorization theory, we show that a balanced replication of batches minimizes the average job compute time for a general class of service time distributions. We then show that the balanced assignment of non-overlapping batches achieves a lower average job compute time than the overlapping schemes proposed in the literature. Next, we derive the optimum redundancy level as a function of the task service time distribution. We show that the redundancy level that minimizes the average job compute time may not coincide with the redundancy level that maximizes job compute time predictability. Therefore, there is a trade-off in optimizing the two metrics. By running experiments on Google cluster traces, we observe that redundancy can reduce the job compute time by one order of magnitude. The optimum level of redundancy depends on the distribution of task service time.**

***Index Terms*— Redundancy, replication, distributed systems, distributed computing, latency, coefficient of variations.**

I. Introduction

DISTRIBUTED computing plays an essential role in modern data analytics and machine learning systems [3], [4]. Parallel task execution on multiple nodes can bring considerable speedup to many practical applications, e.g., matrix multiplication [5], model training in machine learning [6] and convex optimization [7]. However, distributed computing systems are prone to node failure and slowdowns.

In a master-worker architecture, where the master node waits for each worker to deliver its computation results (before possibly moving to the next stage of the algorithm), the latency is determined by the slowest workers, known as *"stragglers"* [8]. Straggler mitigation by task replication has been considered in, e.g., [9]–[12], and by erasure coding in, e.g., [13], [14]. Replication is easier to implement, but erasure coding is generally more efficient (in terms of resource usage) [14]. Furthermore, several works focus on performance analysis of systems level techniques [15]–[17].

We are concerned with distributed computing systems with master-worker architecture, which is used by commercial distributed computing engines such as MapReduce and Kubernetes, and considered in theoretical studies, e.g., [17]–[19]. With redundancy, some workers are selected as backups and thus the system can tolerate some number of stragglers. The current literature on coded computing, e.g. [20], measures the performance of redundancy techniques by the number of stragglers a system can tolerate. However, the core performance metric in practical systems, such as Google [6], Amazon [21] and Facebook [22], is the *latency* that jobs experience; see also [23] and references therein. Minimizing the average latency and maximizing the predictability of latency are among the main concerns in practical systems [8]. Among other results, we show that redundancy techniques that tolerate the same number of stragglers can have different average latency and, therefore, not equally effective in a practical system.

Most computing jobs in machine learning and data analytics applications require running an algorithm on a big dataset [6]. These jobs can be effectively distributed among workers, by assigning to each of them a fraction of the dataset and aggregating their results [18]. For example, the computing jobs that involve multiplication of matrices can be distributed effectively, by assigning rows and columns to the worker nodes and recombining the results by the master node [24]. Such computing jobs (parallelizable to smaller tasks) are common and have motivated this research.

In this article, we derive the efficient assignment of replicated redundancy in a master-worker computing model, where jobs are parallelizable to smaller tasks. For a fixed level of redundancy, we find the optimum task assignment to the workers which minimizes the average job compute time. For such assignments, we find the optimum redundancy level that minimizes the average job compute time, and the (often different) optimum redundancy level that maximizes the compute time predictability. Our specific contributions are as outlined below.

First, given a budget of N workers, a job with N parallel tasks, and a predefined level of redundancy, we derive the optimum task assignment that minimizes the average job compute time. With a predefined level of redundancy, each worker is assigned a *batch* of tasks. Using the results from majorization theory, we show that if the Complementary Cumulative Density Function (CCDF) of the batch compute time is stochastically decreasing and convex in the number of workers hosting the batch, then the minimum average

Manuscript received June 3, 2020; revised December 23, 2020 and February 5, 2021; accepted February 11, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor B. Ji. Date of publication March 24, 2021; date of current version August 18, 2021. This work was supported by NSF Award under Grant CIF-1717314. This work was presented in part in Annual Allerton Conference on Communication, Control and Computing and IEEE international conference on Big Data. *(Corresponding author: Amir Behrouzi-Far.)*

The authors are with the Department of Electrical and Computer Engineering, Rutgers University, New Brunswick, NJ 08901 USA (e-mail: amir.behrouzifar@rutgers.edu; emina.soljanin@rutgers.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNET.2021.3062215, provided by the authors.

Digital Object Identifier 10.1109/TNET.2021.3062215

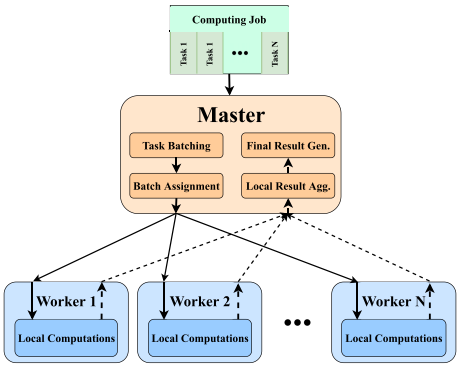

Fig. 1. The master-worker architecture considered in this work. The master node (redundantly) assigns the tasks to the worker nodes. Upon receiving the computation results from a large enough group of workers, the master node generates the overall result.

job compute time is achieved by balanced assignment of non-overlapping batches (see Fig. 2). We then argue that, even though the previously proposed overlapping [18] or non-balanced [25] batch assignments can tolerate (with the same redundancy) the same number of stragglers as the balanced assignment of non-overlapping batches, they are not optimal for minimizing the average job compute time.

Second, with the balanced assignment of non-overlapping batches, we derive the optimum level of redundancy for different service time distributions of tasks. We observe that the higher the randomness in workers' service time, the higher the optimum level of redundancy. Furthermore, when the service time distribution of batches is heavy-tailed, the benefits are larger than when the service times are exponential.

Third, we derive the *Coefficient of Variations* (CoV) of job compute time to measure the degree of *predictability* of job compute time. CoV is among the most important performance metrics both in theory [26] and in practice [8]. We observe that the level of redundancy that minimizes the coefficient of variations of job compute time is not necessarily the same as the level that minimizes the average job compute time. Thus, there exists a trade-off between the two metrics that can be regulated by the redundancy level.

Finally, we run experiments on the Google cluster traces, using the runtime information of several jobs in the dataset. We observe that jobs with both heavy-tail and (shifted) exponential service time distributions of tasks are present in Google clusters. We confirm our theoretical findings that 1) replication can reduce the average compute time, 2) the optimum level of redundancy depends on the service time distribution of tasks, and 3) jobs with heavy-tailed distribution of service time benefit more from redundancy.

This article is organized as follows: In Sec. II, we describe the system architecture and compute job model, task assignment model and the probability distributions we use throughout the article. The task batching schemes, non-overlapping and overlapping batching, are described in the same section. In Section III, we study the optimum assignment of non-overlapping batches. In Sec. IV, we compare the average job compute time of overlapping batches and non-overlapping batches. Optimum redundancy levels that minimize the average job compute time and maximize the compute time predictability are studied in Sec. V. We present our experimental results in Section VI and conclude in Sec. VII. All the proofs are presented in the Appendix.

II. System Model

A. System Architecture and Task Batching Schemes

We study a master-worker architecture, as shown in Fig. 1. We refer to that as system 1 throughout. A compute job is N-parallelizable, that is, it consists of N smaller *tasks* that can be concurrently executed at N workers. The master node 1) forms B batches of tasks (task batching), 2) assigns batches to workers (batch assignment), 3) aggregates computing results from the workers (local result aggregation), and 4) generates the overall result (final result generating). A worker executes all its tasks and only then communicates the aggregate results to the master. Thus, for each batch, a worker communicates only once with the master, which makes the task execution order within a batch inconsequential.

Each task is *redundantly* placed into multiple batches and each batch is assigned to one or more workers. There are at least as many workers as batches, that is, $N \geq B$. A batch is completed as soon as its first replica is completed. This model has been used for a wide range of problems, e.g., matrix multiplication [27], gradient-based optimizers [7], model training in machine learning problems [18]. Batches can have tasks in common (overlapping batches) or not (non-overlapping batches).

1) Non-Overlapping Batches: the N tasks are partitioned into B batches, each of size N/B (see Fig. 2). The two extreme cases are $B = 1$, when all the tasks are in the same batch, and $B = N$, when each batch consists of a single task. With $B < N$, a batch can be assigned to more than one worker. A balanced assignment assigns each batch to a fixed number of workers. The random assignment of non-overlapping batches to workers was considered in the literature [25]. In Section III, we show that the resulting imbalanced distribution of batches among workers is not optimum in terms of average job compute time. Furthermore, random distribution of batches may leave some batches not selected at all. This is shown analytically in Appendix A. In that case, the results of the computations will not be accurate, since they are based on a subset of tasks.

2) Overlapping Batches: the N tasks are redundantly grouped in N batches (see Fig. 2). The batch assignment here is not an issue since each batch is assigned to a single worker. However, we do need to decide how the batches should overlap to minimizes the expected job compute time. In its most general form, this is a hard combinatorial optimization problem, and we will study only the batching schemes that satisfy certain properties as follows. The overlapping batches could be grouped into smaller subsets, such that each task appears exactly once in each subset. In Fig. 2, the batches at W_1 and W_3 make one subset and the batches at W_2 and W_4 make another subset. The reason we consider this particular batching scheme is that 1) the number of workers assigned to each task is fixed (there is no task preference), and 2) a task can appear only once at each worker and thus each task experiences maximum diversity in its service time, as opposed to assigning multiple copies of the same task to one worker. Batching schemes with this property have been proposed in the literature, cf. [18], [25]. With this property,

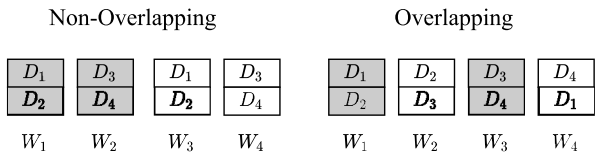

Fig. 2. Task replication policies. Either the dark group or the white group are enough for generating the overall compute result.

the only question is that, within a subset, how should tasks be assigned to the batches. This question will be answered in Section IV.

3) An Example: Consider the problem of optimizing a model $\bar{\beta}$ with distributed gradient descent algorithm [18]. The goal is to minimize the loss function $L(\bar{\beta}; \mathbb{D})$, where $\mathbb{D} = \{D_1, D_2, D_3, D_4\}$ is the dataset. Here, D_is are disjoint subsets of $\mathbb{D}$. At the ith iteration of the algorithm, the model is given by $\bar{\beta}_i = \bar{\beta}_{i-1} - \gamma \nabla L(\bar{\beta}_{i-1}; \mathbb{D})$, where γ is the step size. One can rewrite this equation as $\bar{\beta}_i = \bar{\beta}_{i-1} - \gamma \nabla \sum_{k=1}^{4} L(\bar{\beta}_{i-1}; D_k)$, and the summation can be redundantly distributed among four workers. Two examples of assignments are provided in Fig. 2. With the non-overlapping assignment, the results from the fastest worker among W_1, W_3 and the fastest worker among W_2, W_4 are enough for the master node to generate the overall result. If X_i is the service time random variable (RV) at worker i, with the non-overlapping assignment, the job compute time follows the distribution of $\max\{\min\{X_1, X_3\} \min\{X_2, X_4\}\}$. On the other hand, with the overlapping assignment, the results form the fastest group of workers among W_1, W_3 and W_2, W_4 is sufficient to the overall result. Thus, the job compute time follows the distribution of $\min\{\max\{X_1, X_3\} \max\{X_2, X_4\}\}$. Accordingly, the statistics of job compute time varies with the type of assignment.

B. Service Time Model

Workers take random time to execute tasks, and they are statistically identical. We adopt the the following terminology.
Task service time τ: the time a worker takes to complete a task.
Batch service time X: the time it takes a worker to complete a batch of tasks. With N/B tasks in a batch we have $X \sim \frac{N}{B}\tau$.
Batch compute time Y: the time it takes the first worker (among all hosting the same batch) to complete a batch. Batch i hosted by N_i workers and X_{ij} is the service time of batch i at its jth host worker. Then we have $Y_i \sim \min\{X_{i1}, X_{i2}, \ldots, X_{iN_i}\}$.
Job compute time T: the time it takes to complete the job, i.e. compute all the batches. With B batches we have $T \sim \max\{Y_1, Y_2, \ldots, Y_B\}$.

We assume the tasks have an equal *size* corresponding to their minimum execution time. This is a common assumption in coded computing literature [18]. In particular, in a master-worker architecture, the master node can split a job into tasks of equal size and assign them to the worker nodes. For a given compute job and a fixed number of statistically identical worker nodes, the statistics of job compute time is determined by 1) the task-to-worker assignment policy and 2) the service time distribution of batches/tasks. The following

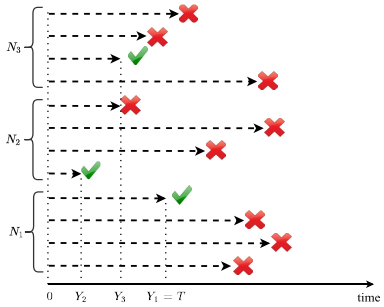

Fig. 3. Time diagram of system 1. Batch i is assigned to $N_i = 4$ workers. The computations of batch i is completed as soon as one of its 4 host workers finishes the computations. Since the master node requires the results of all batches, it has to wait until T_1, at which the slowest group N_1 of workers, assigned to batch 1, complete the computations.

common service time distributions are considered to model the service time of a batch/task.

i. Exponential distribution with rate μ, $X \sim \text{Exp}(\mu)$ where
$$\Pr\{X > x\} = \mathbb{1}(x \geq 0)\mathrm{e}^{-\mu x}. \quad (1)$$

ii. Shifted-exponential distribution with shift Δ and with rate μ, $X \sim \text{SExp}(\Delta, \mu)$ where
$$\Pr\{X > x\} = 1 - \mathbb{1}(x \geq \Delta)\left[1 - \mathrm{e}^{-\mu(x-\Delta)}\right], \quad (2)$$

iii. Pareto distribution with shape α and scale σ, $X \sim \text{Pareto}(\alpha, \sigma)$ where
$$\Pr\{X > x\} = 1 - \mathbb{1}(x \geq \sigma)\left[1 - \left(\frac{x}{\sigma}\right)^{-\alpha}\right]. \quad (3)$$

III. Compute Time With Non-Overlapping Batches

In this section, we show that if the batch compute time is a stochastically decreasing and convex random variable in the number of workers, then the balanced assignment of non-overlapping batches achieves the minimum average job compute time. Let N_i be the number of workers hosting batch i, and define $\bar{N} = (N_1, \ldots, N_B)$ as *batch assignment vector* (cf. Fig. 3). The computation of batch i is completed as soon as one of its N_i host workers finishes the computations, which we refer to as *batch compute time*. Let X_{ij} be the execution time of batch i at the jth host worker, for $j \in \{1, 2, \ldots, N_i\}$. Then, the compute time of the ith batch is the minimum of N_i i.i.d RVs:
$$Y_i \sim \min\left(X_{i1}, X_{i2}, \ldots, X_{iN_i}\right), \quad \forall i \in \{1, 2, \ldots, B\}. \quad (4)$$

For generating the overall result, the master node has to wait for the computation results of all batches. In other words, the overall result can be generated only after the slowest group of workers (hosting the same batch) deliver the local result. Hence, the job compute time T could be written as,
$$T \sim \max\left(Y_1, Y_2, \ldots, Y_B\right). \quad (5)$$

To study the effect of batch assignment on the average job compute time T, we need the following definitions.

Definition 1: The real-valued RV Z_1 is greater than or equal to the real valued RV Z_2 in the sense of ***usual stochastic ordering****, shown by $Z_1 \underset{st}{\geq} Z_2$, if their associated CCDF satisfy*
$$\Pr\{Z_1 > \beta\} \geq \Pr\{Z_2 > \beta\}, \quad \forall \beta \in \mathcal{R}, \quad (6)$$

i.e., $\mathbb{E}[\phi(Z_1)] \geq \mathbb{E}[\phi(Z_2)]$, *for any non-decreasing function* ϕ.

Definition 2: The RV $Z(\theta)$ *is* ***stochastically decreasing and convex*** *in* θ *if its CCDF is a decreasing and convex in* θ.

Definition 3: For any $V_p = (v_{p1}, v_{p2}, \ldots, v_{pM})$ *in* $\mathbb{R}^M$, *the* ***rearranged coordinate vector*** $V_{[p]}$ *is defined as* $V_{[p]} = (v_{[p1]}, v_{[p2]}, \ldots, v_{[pM]})$, *the elements of which are the elements of* V *rearranged in decreasing order, i.e,* $v_{[p1]} > \cdots > v_{[pM]}$.

Definition 4: Let $V_{[p]} = \left(v_{[p1]}, v_{[p2]}, \ldots, v_{[pM]}\right)$ *and* $V_{[q]} = \left(v_{[q1]}, v_{[q2]}, \ldots, v_{[qM]}\right)$ *be two rearranged coordinate vectors in* $\mathbb{R}^M$. *Then* V_p ***majorizes*** V_q, *denoted by* $V_p \succeq V_q$, *if*

$$\sum_{i=1}^{m} v_{[pi]} \geq \sum_{i=1}^{m} v_{[qi]}, \ \forall m \in \{1, 2, \ldots, M\},$$
$$\sum_{i=1}^{M} v_{[pi]} = \sum_{i=1}^{M} v_{[qi]}.$$

Definition 5: Function $\phi : \mathbb{R}^M \to \mathbb{R}$ *is* ***schur convex*** *if for every* $\theta_1, \theta_2 \in \mathbb{R}^M$, $\theta_1 \succeq \theta_2$ *implies* $\phi(\theta_1) \geq \phi(\theta_2)$.

Definition 6: Real-valued RV $Z(\theta)$, $\theta \in \mathbb{R}^M$, *is* ***stochastically schur convex*** *in* θ, *in the sense of usual stochastic ordering, if for any* $\theta_1, \theta_2 \in \mathbb{R}^M$, $\theta_1 \succeq \theta_2$ *implies* $Z(\theta_1) \underset{st}{\geq} Z(\theta)$.

The following lemmas give the batch assignment that minimizes job compute time, under an assumption about the distribution of batch compute time. All proofs hereafter are postponed to the Appendix.

Lemma 1: If the batch assignment $\bar{\mathcal{N}}_1 = (N_{11}, \ldots, N_{1B})$ *majorizes the batch assignment* $\bar{\mathcal{N}}_2 = (N_{21}, N_{22}, \ldots, N_{2B})$, *and the batch compute times are i.i.d stochastically decreasing, convex RVs, then the corresponding job compute times* $T(\bar{\mathcal{N}}_1)$ *and* $T(\bar{\mathcal{N}}_2)$ *satisfy* $\mathbb{E}[T(\bar{\mathcal{N}}_1)] \geq \mathbb{E}[T(\bar{\mathcal{N}}_2)]$,

Proof: The job compute time for the batch assignment vector $\bar{\mathcal{N}}_k$ $\forall k \in \{1, 2\}$, is given by $T(\bar{\mathcal{N}}_k) \sim \max\left(Y_{k1}, Y_{k2}, \ldots, Y_{kB}\right)$, where Y_{ki} $i \in \{1, 2, \ldots, B\}$ is stochastically decreasing and convex in N_{ki}. Since $\max(\cdot)$ is a schur convex function, $T(\bar{\mathcal{N}}_k)$ is stochastically decreasing and schur convex function of $\bar{\mathcal{N}}_k$ (see [28] Section 2 for the detailed discussion). Hence, by Definition 6, $\bar{\mathcal{N}}_1 \succeq \bar{\mathcal{N}}_2$ implies $T(\bar{\mathcal{N}}_1) \underset{st}{\geq} T(\bar{\mathcal{N}}_2)$ in the sense of usual stochastic ordering, which in turn implies that for any non-decreasing function ϕ, we have $\mathbb{E}[\phi(T(\bar{\mathcal{N}}_1))] \geq \mathbb{E}[\phi(T(\bar{\mathcal{N}}_2))]$. Substituting ϕ by the unit ramp function completes the proof. □

Lemma 2: The balanced batch assignment, defined as $\bar{\mathcal{N}}_b = (N/B, N/B, \ldots, N/B)$ *with* N *and* B *being the respective number of workers and batches, is majorized by any other batch assignment policy.*

From Lemma 1 and Lemma 2, when the batch compute times are stochastically decreasing and convex in the number of host workers, the balanced assignment achieves the minimum average job compute time, across all non-overlapping batch assignments.

In the following, we show that for all three service time distributions of batches, the balanced assignment of non-overlapping batches minimizes the average job compute time. Several other assignments are proposed in the literature, cf. [18], [25], which according to our results are not optimal.

A. Exponential Distribution

With exponential service time distribution of batch $X_{ij} \sim \text{Exp}(\mu)$, the compute time of batch $i \in \{1, 2, \ldots, B\}$ is the minimum of N_i i.i.d exponential RVs. Hence,

$$Y_i \sim \text{Exp}\left(N_i \mu\right), \quad \forall i \in \{1, 2, \ldots, B\}. \tag{7}$$

It can be verified that Y_i is stochastically decreasing and convex in N_i. Thus, the Theorem 1 follows.

Theorem 1: With exponential service time distribution of batch $X_{ij} \sim Exp(\mu)$, *among all (non-overlapping) batch assignment policies, the balanced assignment achieves the minimum average job compute time.*

B. Shifted-Exponential Distribution

With shifted-exponential service time distribution of batch, the compute time of a batch consists of a deterministic part and a random part. The deterministic part can be thought of as the minimum required time to complete a batch. Thus, this part is batch dependent and varies with the size of a batch. On the other hand, the random part can be thought of as a worker-dependent component. Note that, in this case the random part is i.i.d exponentially distributed across the workers. Accordingly, the compute time of a batch with $X_{ij} \sim \text{Sexp}(\Delta, \mu)$, is a Δ plus RV (7). Thus, the Corollary 1 follows from Theorem 1.

Corollary 1: With shifted-exponential service time distribution of batch $X_{ij} \sim SExp(\Delta, \mu)$, *among all (non-overlapping) batch assignment policies, the balanced assignment achieves the minimum average job compute time.*

C. Pareto Distribution

With pareto service time distribution of batch, the compute time of a batch consists of a deterministic part and random part. Similar to the shifted-exponential case, the deterministic part can be associated with the batch size and the random part is worker dependent and may vary across the workers.

Theorem 2: Whit Pareto service time distribution of batch $X_{ij} \sim Pareto(\sigma, \alpha)$, *among all (non-overlapping) batch assignment policies, the balanced assignment achieves the minimum average job compute time.*

We showed the desirable decreasing and convexity properties for our choice of distributions. Future studies may consider the same behaviours for other service time distributions. Other extension of this work may consider the optimal batch assignment, when the decreasing and convexity properties do not hold. Furthermore, the optimality of a batch assignment can be studied under different metrics, such as job compute time variability/predictability. However, due to space limitations we leave these problems open for future studies.

IV. Compute Time With Overlapping Batches

We next show that, regardless of service time distribution of batches, the balanced assignment of non-overlapping batches achieves lower average job compute time compared to any overlapping assignments. Let us recall the original problem of assigning N tasks redundantly among N workers. Each worker is assigned with N/B tasks, for a given parameter $B|N$. There are several schemes to group the N tasks into N batches of size N/B. Here, we focus on the schemes where

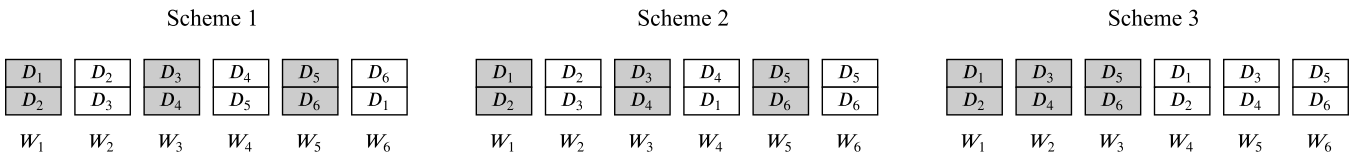

Fig. 4. Three task-to-worker assignment schemes: 1) cyclic overlapping, 2) combination of cyclic overlapping and non-overlapping and 3) non-overlapping.

the set of overlapping batches could be divided into subsets, such that each task appears exactly once in each subset. With these schemes, the number of workers assigned to a task is equal for all tasks. It is worth mentioning that non-overlapping batches can be thought of as an especial case of overlapping batches, where within each subset the batches do not overlap.

Another special we consider is the scheme where the set of tasks is divided into N overlapping batches, each with size N/B, in a cyclic order, as follows. The first batch comprises tasks 1 through N/B, the second batch comprises tasks 2 through $N/B+1$, and so on (see Fig. 4). Note that the batches built with cyclic scheme and non-overlapping batches are the two end of the spectrum. With cyclic scheme the number of batches that share at least one task with any given batch is maximum, whereas with non-overlapping batches this number is minimum. Precisely, with the cyclic scheme, each batch shares at least one common task with $2\left(N/B-1\right)$ other batches. With non-overlapping batches this number is $N/B-1$. With any other batching scheme that conforms to our assumption, this number is greater than $N/B-1$ and smaller than $2\left(N/B-1\right)$. As an example, Scheme 2 in Fig. 4 consists of a cyclic part and a non-overlapping part. In what follows, we aim to compare the average job compute time with the three different batching schemes provided in Fig. 4. Although we provide the comparison for the especial case of $N=6$ and $B=3$, the extension of our method to general values of N and B is straightforward.

Consider system 1, with $N=6$, $B=3$, and three different batching schemes, shown in Fig. 4. In each scheme, there are two subsets of batches which contain exactly one copy of every task. The subsets are shown by different colors. Let $X_i\ \forall i \in \{1,2,\ldots,6\}$ be the i.i.d RV of batch i service time. Let us assume, without loss of generality, that W_1 is the fastest worker delivering its local result before the rest of the workers. Then the job compute time with Scheme 1 is

$$T^{(1)} \sim \min\left(\max\left(X_3,X_5\right),\max\left(X_2,X_4,X_6\right)\right). \tag{8}$$

With Scheme 2, the job compute time could be written as

$$\begin{aligned} T^{(2)} \sim\ & \min(\max(X_3,\min(X_5,X_6)),\\ & \max\left(\max\left(X_2,X_4\right),\min\left(X_5,X_6\right)\right). \end{aligned} \tag{9}$$

Comparing (8) and (9) gives $\mathbb{E}[T^{(2)}]<\mathbb{E}[T^{(1)}]$, since

$$\begin{aligned} &\mathbb{E}[\max\left(X_3,\min\left(X_5,X_6\right)\right)]\\ &\quad<\mathbb{E}[\max\left(X_3,X_5\right)]\ \text{ and}\\ &\quad\ \ \mathbb{E}[\max\left(\min\left(X_5,X_6\right),\max\left(X_2,X_4\right)\right)]\\ &\quad<\mathbb{E}[\max\left(X_2,X_4,X_6\right)]. \end{aligned}$$

On the other hand, with Scheme 3,

$$T^{(3)} \sim \max\left(\min\left(X_3,X_4\right),\min\left(X_5,X_6\right)\right).$$

To be able to compare the job compute time of Scheme 2 and Scheme 3, we rewrite (9) as follows:

$$T^{(2)} \sim \max\left(\min\left(X_3,\max\left(X_2,X_4\right)\right),\min\left(X_5,X_6\right)\right).$$

Since $\mathbb{E}[\min\left(X_3,X_4\right)]<\mathbb{E}[\min\left(X_3,\max\left(X_2,X_4\right)\right)]$. we have that $\mathbb{E}[T^{(3)}]<\mathbb{E}[T^{(2)}]$. Similarly, for the average job compute times of three batching schemes in Fig. 4 we have $\mathbb{E}[T^{(3)}]<\mathbb{E}[T^{(2)}]<\mathbb{E}[T^{(1)}]$. Note that this result does not depend on the service time distribution of batches. Thus, we can conclude that, the balanced assignment of non-overlapping batches achieves a lower average of job compute time when compared to overlapping batch assignment. This is an important result because overlapping assignments have been often proposed in the literature, cf. [18] and [29].

V. Optimum Redundancy Level

In this section, we address the problem of finding the optimum level of redundancy for different service time distributions of tasks. We show that, the optimum redundancy level depends on this distribution, and that for a given distribution it is a function of the distribution's parameters. Furthermore, we show that, the the optimum redundancy level is not necessarily the same for average job compute time and predictability, and that there exists a trade-off between the two metrics when optimizing for the redundancy level.

In a system with B batches and N workers, each batch is replicated N/B times, and we call N/B the *level of redundancy*. Recall the original problem of redundantly assigning tasks to workers in system 1. In one extreme case, all tasks could be assigned to every worker. We refer to this case as *full diversity*, as each task experiences maximum diversity in execution time by getting assigned to every worker. In the other extreme case, each task is assigned to only one worker. We refer to this case as *full parallelism*, as no worker performs redundant executions. Between full diversity and full parallelism there is a spectrum of assignments, each with different redundancy levels. We refer to this spectrum as *diversity-parallelism* spectrum. Note that we make no specific assumption about the maximum allowable level of redundancy in the system. However, in practice, a system may fail if the load exceeds a certain threshold. The implicit assumption in the results of this section is that the system can perform normally under all levels of redundancy.

We are interested in finding the level of redundancy that is optimal according to two important performance metrics: 1) the average job compute time, and 2) the job compute time predictability. While the former has been defined in the preceding sections, we define the latter as the Coefficients of Variations (CoV) of job compute time. Among two jobs with two different coefficients of variations, the one with smaller CoV has more predictable compute time. For a given

redundancy level, the tasks are assigned to workers such that the average job compute time is minimized, as studied in Section III. An extension of our work may consider finding the optimum level of redundancy, where for a given redundancy level the task assignment is done with the objective of maximizing the job compute time predictability.

The service time of a batch depends on the number of tasks in the batch (its size), which is determined by the redundancy level. Several models have been used to describe how the batch service time at a worker scales with its size (see e.g., [23], [30]–[32] and references therein). The model we use here is the most common in the coded computing literature (see e.g., [23] and references therein), described as follows.

In our model, all tasks have identical computing size. We assume that the (random) task service time is determined by the worker it is assigned to, and thus all tasks in a batch experience the same service time. Each worker takes a random time τ to execute a task, which is i.i.d. across the workers. Therefore, the service time of all tasks hosted by the same worker is the same realization of the random variable τ, and the random batch service time is $\frac{N}{B}\tau$. The service times of the replicas of the same task (at different workers) are i.i.d. i.e., different realizations of the random variable τ. This model was called *server dependent scaling* in [23].

The results derived based on this model could as well be used to optimize the redundancy level in systems which follow the proposed model in [30]. According to [30], the batch service time is the product of two factors: 1) an RV associated with the batch size, and 2) an RV associated with the worker's *slowdown*. With equal size tasks, the batch size is constant across the workers, and the only randomness in its service time comes from the slowdown RV. Accordingly, with the slowdown τ and the special case of identical batch sizes of N/B, the batch service time at a worker is $\frac{N}{B}\tau$. Thus, in this special case, the batch service time has the same expression as in our model and our results could be used.

In the rest of the article, we assume N is even and greater than 4, unless otherwise is stated. The extension of the results to odd numbers is straightforward.

A. Shifted-Exponential Distribution

For shifted-exponential service time distribution of tasks, the following theorem gives the optimum level of redundancy that minimizes the average job compute time.

Theorem 3: With shifted-exponential service time distribution of task $\tau \sim SExp(\Delta, \mu)$, the optimum B, achieving the minimum average job compute time, is the solution of the following discrete unconstrained optimization problem,

$$\min_{B \in F_B} \quad \frac{N\Delta}{B} + \frac{1}{\mu} H_{(B,1)}, \tag{10}$$

where F_B is the set of all feasible values for B.

Finding the optimum level of redundancy in (10) requires a search in all the feasible values of B. The following insights can be easily seen. When Δ and μ are large, the term $N\Delta/B$ dominates the objective function. In that case, increasing B, or decreasing the redundancy level reduces the average job compute time. With larger Δ and μ, there is less uncertainty in the tasks' service time. Thus, reducing the redundancy level reduces the average job compute time. On the other hand, for small values of Δ and μ, the term $\frac{1}{\mu} H_{(B,1)}$ dominates the objective function, and thus higher redundancy is more beneficial. This is expected because with higher randomness in tasks' service time, the probability that a worker experiences a severe slowdown is higher, in which case, more redundancy is beneficial.

To reduce the complexity of the search algorithm in (10), the following theorem establishes a connection between the optimum operating point in the diversity-parallelism spectrum and the parameters of tasks' service time distribution.

Theorem 4: With shifted-exponential service time distribution of task $\tau \sim SExp(\Delta, \mu)$, the optimum operating point in the diversity-parallelism spectrum, achieving the minimum average job compute time, is

- *at full diversity when $\Delta\mu < 1/N$,*
- *at a middle point when $1/N \leq \Delta\mu \leq \sum_{k=N/2+1}^{N} 1/k$,*
- *at full parallelism when $\sum_{k=N/2+1}^{N} 1/k < \Delta\mu$.*

Corollary 2: With shifted-exponential service time distribution of task $\tau \sim SExp(\Delta, \mu)$ and $1/N \leq \Delta\mu \leq \sum_{k=N/2+1}^{N} 1/k$, the optimum operating point in the diversity-parallelism spectrum, achieving the minimum average job compute time, is the solution of the following discrete unconstrained optimization problem,

$$\min_{B \in F_B} \quad |B - N\Delta\mu|, \tag{11}$$

where F_B is the set of all feasible values for B.

Note that the complexity of solving (11) is $O(\log |F_B|)$. The average job compute time vs. B for different values of μ, when the task service time follows shifted-exponential distribution, is plotted in Fig. 5, for $N = 100$ and $\Delta = 0.05$. For this set of parameters, $1/N = 0.01$ and $\sum_{k=N/2+1}^{N} 1/k \approx 0.69$. Hence, for $\mu < 0.2$ full diversity should minimize the average job compute time. For $0.2 \leq \mu \leq 13.8$ the optimum point should be in the middle of the spectrum. Finally, for $\mu > 13.8$ full parallelism should minimize the average job compute time. All these regions could be verified in Fig. 5.

Next, we find the optimum level of redundancy that minimizes the coefficient of variations of job compute time.

Lemma 3: With shifted-exponential service time distribution of task $\tau \sim SExp(\Delta, \mu)$, the coefficient of variations of the job compute time is given by

$$CoV[T] = \sqrt{H_{(B,2)}} \Big/ \big[N\Delta\mu/B + H_{(B,1)}\big]. \tag{12}$$

Theorem 5: With shifted-exponential service time distribution of task $\tau \sim SExp(\Delta, \mu)$, the optimum operating point in the diversity-parallelism spectrum, achieving the minimum coefficient of variations of job compute time, is

- *at full parallelism when $\Delta\mu < 3/(\sqrt{5}-1)N$,*
- *at either end of the spectrum when $\frac{3}{(\sqrt{5}-1)N} \leq \Delta\mu \leq \frac{H_{(N,1)}\sqrt{H_{(N/2,2)}} - H_{(N/2,1)}\sqrt{(H_{(N,2)})}}{2\sqrt{H_{(N,2)}} - \sqrt{H_{(N/2,2)}}}$,*
- *at full diversity when $\Delta\mu > \frac{H_{(N,1)}\sqrt{H_{(N/2,2)}} - H_{(N/2,1)}\sqrt{(H_{(N,2)})}}{2\sqrt{H_{(N,2)}} - \sqrt{H_{(N/2,2)}}}$.*

When tasks' service times follow i.i.d shifted-exponential distribution, the optimum operating point, minimizing the coefficient of variations of job compute time, is either full

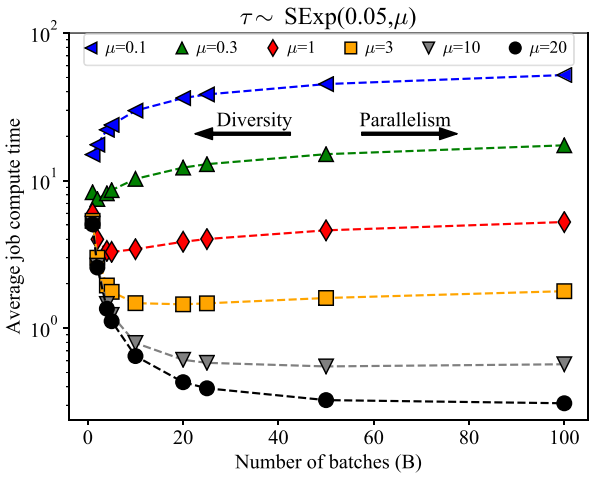

Fig. 5. Average job compute time with $\tau \sim SExp(0.05, \mu)$, versus the number of batches, for different values of μ. The minimum value of $\mathbb{E}[T]$ moves toward the full parallelism as μ increases.

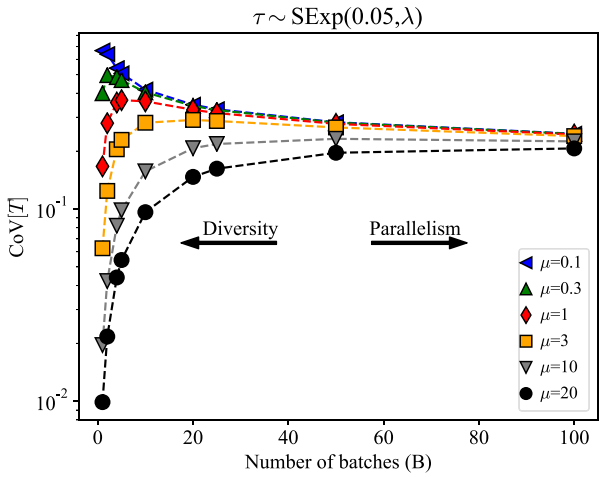

Fig. 6. Coefficient of variations of job compute time with $\tau \sim SExp(0.05, \mu)$, versus the number of batches, for different values of μ. The minimum value of $\mathrm{CoV}[T]$ moves toward the full diversity as μ increases.

diversity or full parallelism. With higher randomness in the tasks' service time, i.e $\Delta\mu \in (-\infty, 3/(\sqrt{5}-1)N)$, then assigning each worker with a single task minimizes the coefficient of variations of job compute time. On the other hand, with lower randomness in the tasks' service time, i.e., large $\Delta\mu$, full diversity $B = 1$ is optimal. This is a sharp contrast with levels of redundancy that minimizes the average job compute time. Specifically, with high randomness, a high redundancy level is required to minimize the average job compute time, but a low redundancy level is required to minimize the coefficient of variations of job compute time. In general, with shifted-exponential distribution of tasks' service time, one could not minimize both the expected value and the coefficient of variations of job compute time. This result sheds light on the impossibility of reducing the average latency of compute jobs and maximizing the predictability of their service time in practical systems.

We simplify the problem of minimizing the coefficient of variations by the following corollary, which gives the optimum level of redundancy for middle values of $\Delta\mu$ and large N.

Corollary 3: With shifted-exponential service time distribution of task $\tau \sim SExp(\Delta, \mu)$, $N > 11$ *and* $\frac{3}{(\sqrt{5}-1)N} \leq \Delta\mu \leq \frac{H_{(N,1)}\sqrt{H_{(N/2,2)}} - H_{(N/2,1)}\sqrt{(H_{(N,2)})}}{2\sqrt{H_{(N,2)}} - \sqrt{H_{(N/2,2)}}}$, *the optimum operating point in the diversity-parallelism spectrum, achieving the minimum coefficient of variations of job compute time, is*

- *at full parallelism when* $\Delta\mu < H_{(N,1)}/(N\sqrt{H_{(N,2)}} - 1)$,
- *at full diversity when* $H_{(N,1)}/(N\sqrt{H_{(N,2)}} - 1) \leq \Delta\mu$.

The coefficient of variations of job compute time is plotted in Fig. 6, for $N = 100$ and $\Delta = 0.05$. For this set of parameters, $H_{(N,1)}/N(\sqrt{H_{n,2}} - 1) \approx 0.04$. Thus, for $\mu < 0.04/\Delta \approx 0.8$ full diversity and for $\mu > 0.8$ full parallelism should be optimal. These regions can be verified in Fig. 6. Finally, we present the results for the limit case, when $N \to \infty$ in the following corollary.

Corollary 4: With $\tau \sim SExp(\Delta, \mu)$, *the optimum operating point in the diversity-parallelism spectrum, achieving the minimum coefficient of variations of job compute time, occurs at full diversity, as* $N \to \infty$.

With shifted-exponential distribution of tasks' service times, we conclude that the expected value and the coefficient of variations of job compute time may not be optimized by the same redundancy level. For small and large values of $\Delta\mu$ product, the optimum points are at the opposite ends of the spectrum. In other words, the level of redundancy that minimizes the average compute time results in the maximum coefficient of variations, and vice versa. Therefore, there is an inevitable trade-off between the average value and the coefficient of variations of job compute time, when tasks' service time follow shifted-exponential distribution. As a rule of thumb, when $\Delta\mu$ is small, the average job compute time is smaller at high diversity and the coefficient of variations of job compute time is smaller at high parallelism. Whereas, when $\Delta\mu$ is large, the average job compute time is smaller at high parallelism regime and the coefficient of variations is smaller at high diversity.

B. Pareto Distribution

With Pareto distribution of tasks' service time, the following theorem gives the optimal level of redundancy that minimizes the average job compute time.

Theorem 6: With Pareto service time distribution of task $\tau \sim Pareto(\sigma, \alpha)$, *the optimum level of redundancy, achieving the minimum average job compute time, is the solution of the following discrete unconstrained optimization problem,*

$$\min_{B \in F_B} \quad \frac{N\sigma}{B} \cdot \frac{\Gamma(B+1) \cdot \Gamma(1 - B/N\alpha)}{\Gamma(B+1-B/N\alpha)}, \tag{13}$$

where $\Gamma(\cdot)$ *is the Gamma function and* F_B *is the set of all feasible values for* B.

From (13), the average compute time grows linearly with the scale parameter σ. Nevertheless, its behaviour depends solely on the shape parameter α. Therefore, the optimum level or redundancy is a function of α, as follows.

Theorem 7: With Pareto service time distribution of task $\tau \sim Pareto(\sigma, \alpha)$, *the optimum operating point in the*

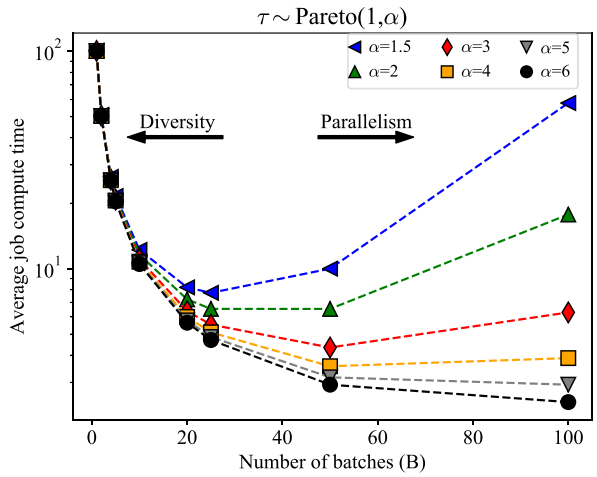

Fig. 7. Average job compute time with $\tau \sim \text{Pareto}(1, \alpha)$, versus the number of batches, for different values of α. The minimum value of $\mathbb{E}[T]$ moves toward the full parallelism point as α increases.

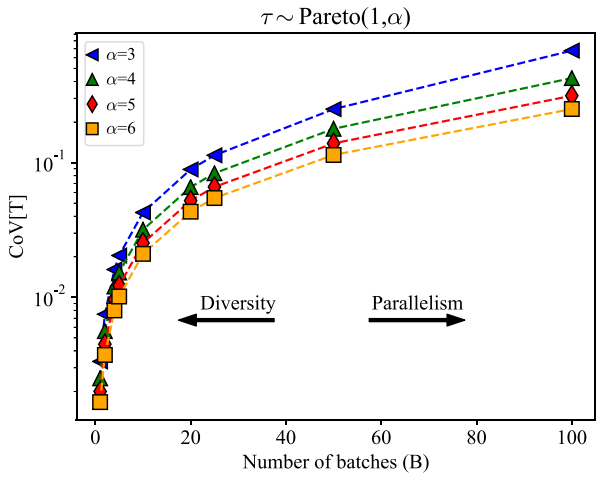

Fig. 8. Coefficient of variations of job compute time with $\tau \sim \text{Pareto}(1, \alpha)$, versus the number of batches, for different values of α. The minimum value of $\text{CoV}[T]$ is at the full diversity for all $\alpha > 2$.

diversity-parallelism spectrum, achieving the minimum average job compute time, is

- *at a middle point when* $1 < \alpha < \alpha^*$, *and*
- *at full parallelism when* $\alpha \geq \alpha^*$,

where α^* *is the solution of the following equation,*

$$\frac{4\alpha^2 + (\alpha-1)^2}{2\alpha(\alpha-1)} - \sqrt{\pi} N^{-1/2\alpha} 2^{1+1/2\alpha} - 0.58 = 0. \quad (14)$$

A Pareto RV with large shape parameter α has a lighter tail and thus less randomness. Therefore, with large enough α full parallelism minimizes the average job compute time. With smaller values of α, redundancy may be required to reduce the randomness in tasks' service time. Therefore, the optimal operating point should move towards the full diversity end of the spectrum. For $N = 100$ and $\sigma = 1$ the average job compute time is plotted in Fig. 7. With this set of parameters, $\alpha^* \approx 4.7$. Thus, for $\alpha < 4.7$ the optimum B lies in a mid point of the diversity-parallelism spectrum. On the other hand, for $\alpha > 4.7$ the optimum B is at full parallelism, which can be verified by the plots in Fig. 7.

Lemma 4: With Pareto service time distribution of task $\tau \sim Pareto(\sigma, \alpha)$, *the coefficient of variations of job compute time is given by*

$$CoV(T) = \sqrt{\frac{\Gamma(B+1-B/N\alpha).\Gamma(1-2B/N\alpha)}{\Gamma(B+1-2B/N\alpha).\Gamma(1-B/N\alpha)} - 1}.$$

With Pareto distribution, the coefficient of variations does not depend on the scale parameter σ of the distribution. The following theorem gives the optimum level of redundancy that minimizes the coefficient of variation given by Lemma 4.

Theorem 8: With Pareto service time distribution of task $\tau \sim Pareto(\sigma, \alpha)$, *the coefficient of variations of job compute time is minimized at full diversity.*

Fig. 8 shows the coefficient of variations of job compute time with Pareto distribution of tasks' service time. The optimum operating point is at full diversity, regardless of the value of α. However, full diversity maximizes the average compute time for all values of α, as it is shown in Fig. 7. This result shows the trade-off between the expected value and the coefficient of variations of job compute time, with heavy-tail distribution of tasks' service time. For both exponential tail and heavy tail distributions of the workers' slow down, we have shown that there exists a trade-off between the average job compute time and the compute time predictability. In other words, minimizing the average latency of compute jobs and maximizing the predictability of this latency at the same time may not be possible. Therefore, in practical systems, this trade-off has to be considered in order to balance a reasonable balance between the average latency and predictability.

VI. Experiments

We next present results based on experiments on the dataset from Google cluster traces [33] which provides the runtime information of the jobs in Google clusters. Jobs consists of multiple tasks, each executed by a worker. The recorded information for each task includes, among others, its scheduling and finish times. We recorded the service time of a task as the difference between its finish time and scheduling time.

We observed that the tasks' service time could follow both heavy-tail or exponential-tail behaviours, depending on the job. The task compute time CCDF is plotted in Fig 9. A linear decay in log-log scale means heavy-tail behaviour and exponential decay means exponential-tail. Accordingly, jobs 1 through 4 show exponential decay in tail probability, whereas jobs 5 through 10 have almost linear and thus heavy-tail decay.

To show the effect of redundancy level on the average compute time, we sampled tasks within a job. With the sampled tasks, we formed batches and assigned each batch to a given number of workers, which is fixed across the batches. In Fig 10, we plot the average job compute time, normalized by the average compute time with no redundancy, versus the number of batches B. Jobs 1 – 4 have exponential decay in the tail probability. For these jobs, the shift parameter for the shifted-exponential distribution is large (10 for jobs 1 – 3 and

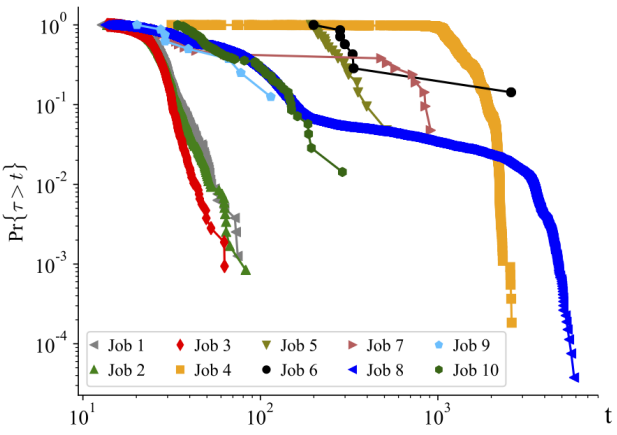

Fig. 9. CCDF of the task compute time for 10 jobs. The run times of the tasks are extracted from Google cluster traces dataset.

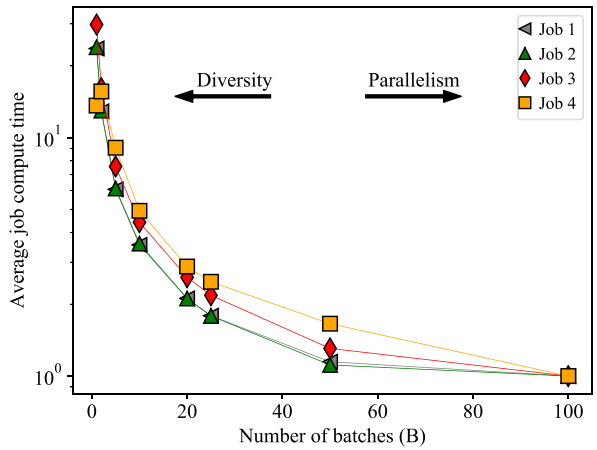

Fig. 10. The effect of redundancy on the average job compute time, when the service time of tasks within the job has **exponential-tail** distribution.

1000 for job 4). As predicted by our analysis, full parallelism minimizes the average job compute time.

In Fig. 11 we plot the normalized average job compute time versus the number of batches, where task service times are heavy-tailed. The minimum average job compute times occur somewhere between full parallelism and full diversity. This observation is inline with our analysis of Pareto distribution for tasks' service time. The optimum level of redundancy, however, depends on the job type. For instance, jobs 6, 8, 9 and 10 have their minimum average compute time at $B = 20$ whereas for job 5 and 7 the optimum value is $B = 50$ and $B = 10$, respectively. This difference in the optimum level of redundancy is a result of the shape parameter of the distribution. For instance, job 7 decays faster than the other heavy-tail jobs and thus its task service time distribution has a larger shape parameter. Therefore, it would require lower levels of redundancy. Note that the heavy-tail behavior of task service times of some jobs in Fig. 9 may not exactly fit into Pareto distribution. However, they are heavy-tailed and our general results still apply.

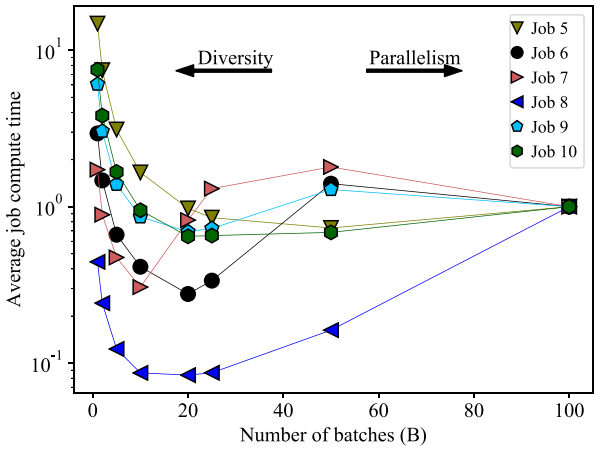

Fig. 11. The effect of redundancy on the normalized average job compute time, when the task service time is **heavy tailed**.

VII. Conclusion

We studied the efficient task assignment problem in master-worker distributed computing system. For a given level of redundancy, we showed that if the batch compute times are i.i.d stochastically decreasing and convex in in the number of workers, a balanced assignment of non-overlapping batches achieves the minimum average job compute time. We then studied the optimum level of redundancy for minimizing the average job compute time and maximizing compute time predictability. With both exponential-tail and heavy-tail distribution of workers' slow down, we showed that the redundancy level that minimizes the average job compute time is not necessarily the same as the redundancy level the maximizes the compute time predictability of jobs. Therefore, when optimizing for the optimum redundancy level, there exists an inevitable trade-off between the average and the predictability of job compute time. Finally, we evaluate the redundant task assignment with the data from Google cluster traces. We showed that a careful assignment of redundant tasks can reduce the average job compute time by an order of magnitude.

References

[1] A. Behrouzi-Far and E. Soljanin, "On the effect of task-to-worker assignment in distributed computing systems with stragglers," in *Proc. 56th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2018, pp. 560–566.

[2] A. Behrouzi-Far and E. Soljanin, "Data replication for reducing computing time in distributed systems with stragglers," 2019, *arXiv:1912.03349*. [Online]. Available: http://arxiv.org/abs/1912.03349

[3] T. Kraska, A. Talwalkar, J. Duchi, R. Griffith, M. J. Franklin, and M. Jordan, "MLbase: A distributed machine-learning system," *CIDR*, vol. 1, pp. 1–2, Jan. 2013.

[4] P. Buchlovsky *et al.*, "TF-replicator: Distributed machine learning for researchers," 2019, *arXiv:1902.00465*. [Online]. Available: http://arxiv.org/abs/1902.00465

[5] A. R. Benson and G. Ballard, "A framework for practical parallel fast matrix multiplication," *ACM SIGPLAN Notices*, vol. 50, no. 8, pp. 42–53, Dec. 2015.

[6] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231.

[7] S. Boyd, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2010.

[8] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.
[9] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," in *Proc. ACM Int. Conf. Meas. Modelinf Comput. Syst.*, 2014, pp. 599–600.
[10] K. Gardner *et al.*, "Reducing latency via redundant requests: Exact analysis," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 1, pp. 347–360, 2015.
[11] G. Joshi, E. Soljanin, and G. Wornell, "Efficient redundancy techniques for latency reduction in cloud systems," *ACM Trans. Modeling Perform. Eval. Comput. Syst.*, vol. 2, no. 2, pp. 1–30, May 2017.
[12] G. Joshi, E. Soljanin, and G. Wornell, "Efficient replication of queued tasks for latency reduction in cloud systems," in *Proc. 53rd Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2015, pp. 107–114.
[13] M. Fatih Aktas, P. Peng, and E. Soljanin, "Effective straggler mitigation: Which clones should attack and when?" 2017, *arXiv:1710.00748*. [Online]. Available: http://arxiv.org/abs/1710.00748
[14] M. F. Aktas and E. Soljanin, "Straggler mitigation at scale," *IEEE/ACM Trans. Netw.*, vol. 27, no. 6, pp. 2266–2279, Dec. 2019.
[15] M. F. Aktas, P. Peng, and E. Soljanin, "Straggler mitigation by delayed relaunch of tasks," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 3, pp. 224–231, Mar. 2018.
[16] E. Ozfatura, D. Gunduz, and S. Ulukus, "Speeding up distributed gradient descent by utilizing non-persistent stragglers," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 2729–2733.
[17] N. Ferdinand and S. C. Draper, "Anytime stochastic gradient descent: A time to hear from all the workers," in *Proc. 56th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2018, pp. 552–559.
[18] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding," 2016, *arXiv:1612.03301*. [Online]. Available: http://arxiv.org/abs/1612.03301
[19] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic MDS codes and expander graphs," 2017, *arXiv:1707.03858*. [Online]. Available: http://arxiv.org/abs/1707.03858
[20] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proc. 22nd Int. Conf. Artif. Intell. Statist.*, 2019, pp. 1215–1225.
[21] K. R. Jackson *et al.*, "Performance analysis of high performance computing applications on the Amazon Web services cloud," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci.*, Nov. 2010, pp. 159–168.
[22] K. Hazelwood *et al.*, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 620–629.
[23] P. Peng, E. Soljanin, and P. Whiting, "Diversity/parallelism trade-off in distributed systems with redundancy," 2020, *arXiv:2010.02147*. [Online]. Available: http://arxiv.org/abs/2010.02147
[24] A. Buluç and J. R. Gilbert, "Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments," *SIAM J. Sci. Comput.*, vol. 34, no. 4, pp. C170–C191, Jan. 2012.
[25] S. Li, S. Mohammadreza Mousavi Kalan, A. Salman Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," 2017, *arXiv:1710.09990*. [Online]. Available: http://arxiv.org/abs/1710.09990
[26] M. Harchol-Balter, N. Bansal, and B. Schroeder, "Implementation of SRPT scheduling in Web servers," CMU-CS, Pittsburgh, PA, USA, Tech. Rep., 2000.
[27] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2418–2422.
[28] L. Liyanage and J. G. Shanthikumar, "Allocation through stochastic Schur convexity and stochastic transposition increascingness," in *Proc. Lect. Notes-Monograph Ser.*, 1992, pp. 253–273.
[29] M. M. Amiri and D. Gunduz, "Computation scheduling for distributed machine learning with straggling workers," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 8177–8181.
[30] K. Gardner, M. Harchol-Balter, and A. Scheller-Wolf, "A better model for job redundancy: Decoupling server slowdown and job size," in *Proc. IEEE 24th Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst. (MASCOTS)*, Sep. 2016, pp. 1–10.
[31] M. Zubeldia, "Delay-optimal policies in partial fork-join systems with redundancy and random slowdowns," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 1, pp. 1–49, May 2020.
[32] P. Peng, E. Soljanin, and P. Whiting, "Diversity vs. Parallelism in distributed computing with redundancy," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2020, pp. 257–262.
[33] J. Wilkes. (Nov. 2011). *More Google Cluster Data*. [Online]. Available: http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html
[34] E. W. Weisstein, "Stirling number of the second kind," *Triangle*, vol. 7, p. 8, Dec. 2002.
[35] A. N. Myers and H. S. Wilf, "Some new aspects of the coupon Collector's problem," *SIAM Rev.*, vol. 48, no. 3, pp. 549–565, Jan. 2006.
[36] B. C. Arnold, *Pareto Distribution*. Hoboken, NJ, USA: Wiley, 2014, pp. 1–10.

Amir Behrouzi-Far received the B.S. degree in electrical engineering from the Iran University of Science and Technology, Tehran, Iran, in 2013, and the M.S. degree in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in 2016. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Rutgers University, New Brunswick, NJ, USA. His research interests include performance evaluation in distributed systems, timeliness in real time systems, scheduling in computing/storage systems, wireless communications, and reinforcement learning.

Emina Soljanin (Fellow, IEEE) was a (Distinguished) Member of Technical Staff for 21 years in the Mathematical Sciences Research, Bell Labs. In January 2016, she joined Rutgers University, where she is currently a Professor. Her interests and expertise are wide. Over the past quarter of the century, she has participated in numerous research and business projects, as diverse as power system optimization, magnetic recording, color space quantization, hybrid ARQ, network coding, data and network security, distributed systems performance analysis, and quantum information theory.

Prof. Soljanin is an Outstanding Alumnus of the Texas A&M School of Engineering, the 2011 Padovani Lecturer, a Distinguished Lecturer from 2016 to 2017, and the 2019 President of the IEEE Information Theory Society. She served as an Associate Editor for *Coding Techniques* and the IEEE TRANSACTIONS ON INFORMATION THEORY, on the Information Theory Society Board of Governors, and in various roles on other journal editorial boards and conference program committees.