2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)
Allerton Park and Retreat Center
Monticello, IL, USA, October 2-5, 2018

On the Effect of Task-to-Worker Assignment in Distributed Computing Systems with Stragglers

Amir Behrouzi-Far[1] and Emina Soljanin[2]

***Abstract*— We study the expected completion time of some recently proposed algorithms for distributed computing which redundantly assign computing tasks to multiple machines in order to tolerate a certain number of machine failures. We analytically show that not only the amount of redundancy but also the task-to-machine assignments affect the latency in a distributed system. We study systems with a fixed number of computing tasks that are split in possibly overlapping batches, and independent exponentially distributed machine service times. We show that, for such systems, the uniform replication of non-overlapping (disjoint) batches of computing tasks achieves the minimum expected computing time.**

I. Introduction

Distributed computing has gained great attention in the time of big data [1]. By enabling parallel task execution, distributed computing systems can bring considerable speed ups to e.g. matrix multiplication [2], model training in machine learning [3] and convex optimization [4]. However, implementing computing algorithms in a distributed system introduces new challenges that have to be addressed in order to benefit from paralleziation. In particular, since the failure rate and/or slow down of the system increase with the number of computing nodes, robustness is an essential part of any reliable distributed computing/storage algorithm [5]. For achieving robustness, redundant computing/storage is introduced in the literature [5], [6]. Redundancy in a distributed computing system enables the system to generate the overall result form the computations of a subset of all computing nodes, which provides robustness to failed and/or slow nodes, known as stragglers.

A number of algorithms for distributed computing which tolerate the failure of some computing nodes have recently appeared in the literature. For example, in [7] used error correcting codes and, for distributed gradient descent, authors established lower bound on the degree of redundancy for tolerating a specific number of node failures, and introduced some data-to-node assignment methods that achieve the lower bound. In [8], Reed-Solomon codes are studied for distributed gradient descent, and the code construction, decoding algorithm and an asymptotic bound on the computation time are provided. In [9], a random data assignment to computing nodes is proposed, and claimed to outperform the deterministic method in [7] in terms of average per-iteration computing time. Fault-tolerant distributed matrix multiplication is addressed in e.g. [10]. In this line of work, authors focus on failure tolerance capability of their proposed algorithms without considering the resulting computing time.

The effect of the replicated redundancy on the distributed computing latency was studied in [11], where it is shown that delayed replication of the straggling tasks could reduce both cost and latency in the system. Moreover, authors in [12] analyzed the latency of the system with coded redundancy, where instead of simple replication of the straggling tasks a coded combination of them would be introduced. The analysis of the same work shows that, coded redundancy could achieve the same latency with less cost of the computing. In [13] authors show that relaunching straggling tasks could significantly reduce the cost and latency in a distributed computing system. In these works, the delay of computing is considered without looking into the effect of task assignment in the timing performance.

In this paper, we study the computing time of the failure-tolerant distributed computing algorithms. We specifically focus on the algorithms proposed by [7] and [9], which used error correcting codes and random task assignment respectively, and show that how the task assignment could improve the expected computing time of the same methods. We consider a distributed computing system consisting of multiple worker nodes, which perform a specific computation over a sebset of a possibly huge data set, and a master node, which collects the local computations from worker nodes and generates the overall result. The original data set is (redundantly) distributed among worker nodes and the subset of data at each worker is called a *data batch*. Through analysis, we show that, with a fixed number of computing tasks and exponentially distributed service time of worker nodes, the lower bound of the expected computing time among different data distribution policies is achieved by the balanced assignment of non-overlapping batches. We also derived this lower bound and verified it by simulations.

The organization of this paper is as follows. In section II, we introduce the model for system architecture, computing task, data distribution and service time of the worker nodes. The data distribution policies, non-overlapping and overlapping batches, are described in section III. In section IV we provide analysis of the computing time for each data distribution policies. The numerical results and the concluding remarks are provided in section V and section VI, respectively.

[1]Amir Behrouzi-Far is with the Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ 08854, USA amir.behrouzifar@rutgers.edu

[2]Emina Soljanin is with the Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ 08854, USA emina.soljanin@rutgers.edu

II. System Model

A. Distributed Computing Architecture

We study the system given in Fig. 1, which will be referred to as System 1 in the rest of the paper. The system consists of a data set with S data blocks, a batching unit which partitions the data blocks into B batches, a batch assignment unit which allocates data batches among N worker nodes, each performing a specific computing task on its data batch, and a master node which collects local (coded) computations from worker nodes and computes the overall result accordingly. For a given data set, a fixed number of identical worker nodes and a master node, the average computing time of the system depends on how the data is distributed among workers, what the service time model of the worker nodes is, and how each worker codes its computations before sending it to the master node.

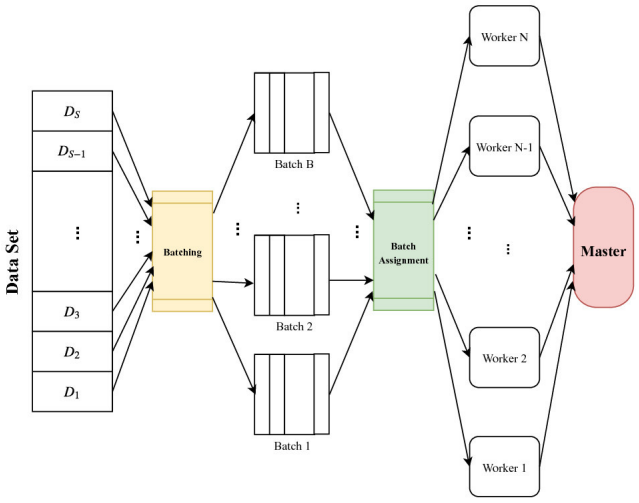


Fig. 1. System model.

B. Computing Task Model

In this work, we consider data-intensive jobs that could be divided into smaller tasks to be executed in parallel. Each task is assigned to a worker node which performs computations over a subset of the entire data and sends its local result to the master node. After receiving the local results from a large enough number of workers, the master node generates the overall result. This is a well applicable and widely used model for different algorithms, e.g. matrix multiplication [2] and gradient based optimizers [4], [7]. As a simple example of this computing model, consider a set of natural numbers $D = \{1, 2, 3, \dots, z\}$. Suppose we are interested in the sum of ℓ-th power of the elements of this set, i.e.,

$$S_z^{(\ell)} = \sum_{k=1}^{z} k^\ell.$$

This summation could be decomposed into N sums (one for each worker):

$$S_z^{(\ell)} = \sum_{k=1}^{z_1} k^\ell + \sum_{k=z_1+1}^{z_2} k^\ell + \sum_{k=z_2+1}^{z_3} k^\ell + \cdots + \sum_{k=z_{N-1}+1}^{z_N} k^\ell, \tag{1}$$

where $z_1 < z_2 < z_3 < \cdots < z_{N-1} < z_N = z$. Now, if we split the set D into subsets of size $z_1, z_2 - z_1, z_3 - z_2, \dots, z_N - z_{N-1}$ and assign each subset to one worker node, then a worker would have to compute the ℓ-th power for each number in the data subset (batch) assigned to it and add them up. Then the master node would be able to generate $S_z^{(\ell)}$ by summing the local computation results.

C. Data Distribution

To ensure reliable computations, each block of the data set is stored redundantly among workers. The redundant distribution of data among the worker nodes is abstracted into a two-stage process. In the first stage, the data set is stored in equal-sized batches and in the second stage batches get assigned to the workers. Note that, the batches could be non-overlapping, when the data set is simply chopped into smaller parts, or they could overlap, if each data block is available in more than one batch. Nevertheless, the batch sizes will be the same in both cases. Therefore, there would be more batches if they overlap. Assuming the batch size is S/B, where B divides S, the number of batches is an integer in the range of $[B, N]$.

D. Service Time Model

We define $T_{i,j}$ as the time that worker j takes to perform computing over data batch i and communicate the result to the master node. We assume that $T_{i,j}$ are all independent and exponentially, identically distributed:

$$T_{i,j} \sim \exp(\lambda), \tag{2}$$

where λ is the service rate of the worker nodes and is identical for all of the workers.

III. Data Distribution

The data distribution policies could be categorized as: 1) non-overlapping batches, or 2) overlapping batches. We will discuss each policy in detail in the following.

A. Non-overlapping Batches

Under this batching policy, the data set of S blocks is split into B equal-size batches, giving batch size S/B. When $B < N$, batches can be assigned redundantly to the N worker nodes, in order to enable failure and/or straggler tolerance in the system. Since the intersection of the batches are empty, the data at each worker either completely overlap or do not overlap at all with the data at any other worker. Therefore, for acquiring the computations over a specific batch, the master node should receive the local results from at least one of the workers hosting that batch.

The random batch assignment in which each worker draws a batch, uniformly at random, with replacement from the pool

of batches was studied in [9], and can be naturally modeled as the Coupon Collection (CC) problem. It was shown in [9] that the random assignment reduces the computation time per iteration compared to deterministic assignment in [7]. We will show in the following section that the imbalance data distribution among workers, which results from the random assignment, adversely affects the computation time. On the other hand, since some of the batches will be drawn only once, there will be no failure tolerance for a worker if its data is not replicated at any other worker. Furthermore, by random assignment, there is always a non-zero probability that some batches not get selected, leading to an inaccurate computation result.

Let n be the number of workers for covering all the batches in random assignment policy. The following proposition provides the data coverage probability with the random batch-to-worker assignment.

Proposition 1. *The probability of covering B batches with N workers with random batch-to-worker assignment is given by,*

$$P(n \leq N) = \frac{B!}{B^N} \left\{ {N \atop B} \right\}, \tag{3}$$

where $\left\{ {n \atop B} \right\}$ is the Stirling number of second kind [14], given by,

$$\left\{ {n \atop B} \right\} = \frac{1}{B!} \sum_{i=0}^{B} (-1)^{B-i} \binom{B}{i} i^n.$$

Proof. The probability of covering B batches with exactly N workers is given in [15], as

$$P(n = N) = \frac{B!}{B^N} \left\{ {N-1 \atop B-1} \right\}. \tag{4}$$

Therefore,

$$\begin{aligned} P(n \leq N) &= \sum_{n=B}^{N} \frac{B!}{B^n} \left\{ {n-1 \atop B-1} \right\} \\ &= B! \sum_{n=B-1}^{N-1} \frac{1}{B^{n+1}} \left\{ {n \atop B-1} \right\} \\ &= \frac{B!}{B^N} \sum_{n=B-1}^{N-1} B^{N-n-1} \left\{ {n \atop B-1} \right\} \\ &= \frac{B!}{B^N} \left\{ {N \atop B} \right\}, \end{aligned}$$

where the last equation holds according to [16]. □

The probability of covering all batches (3) vs. the number of batches is plotted for four different values of N in Fig. 2. We see that in order to cover all the B batches with N workers, the number of workers should be much larger than the number of batches. On the other hand, as the number of batches increases, the probability of covering them with a given number of workers decreases. Note that, the number of batches is of great interest to us, since file systems, [17] and [18], usually store data in batches of fixed size. Therefore, the larger the data the larger the number of data batches. For example, the batch size in Hadoop distributed file system (HDFS) is 64MB, [18].

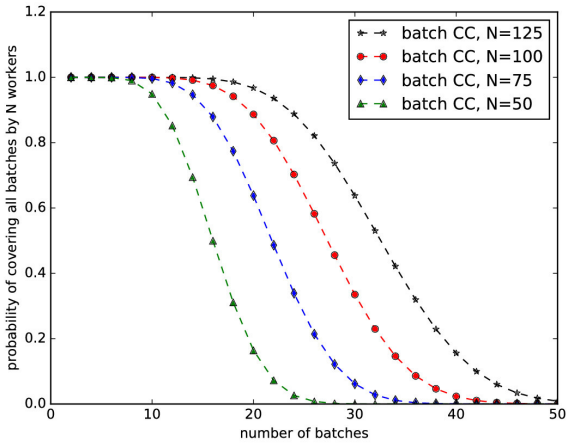


Fig. 2. Coverage probability of batch CC with N workers.

B. Overlapping Batches

Under this batching policy, the data set is stored in N overlapping batches, each assigned to a worker node. To be able to compare different data distributions' computing time, we keep the same batch sizes in both overlapping and non-overlapping policies. Thus, the batch size with overlapping batches is S/B, where B is the number of batches in non-overlapping policy. Note that with overlapping batches, the number of batches is equal to the number of workers and, therefore, we need not decide about the number of workers assigned to each batch, which was the problem with non-overlapping batches. Besides, the challenge here is that which batches and how much should they overlap for faster computing. This question is in general very hard to answer (due to the huge size of the problem) and we will study it under the following assumption. We assume that the entire set of batches could be divided into non-overlapping *group* of batches, such that each block of data set appears one and only one time in each group. In other words, each group hosts the entire data set, divided into batches of size S/B. Now the question is how we should arrange the elements of the data set in each group to have faster computations, which will be answered in section IV-B.

IV. Computing Time Analysis

In this section we analyse the computing time of System 1, under the two data distribution policies described in section III.

A. Computing Time with Non-overlapping Batches

Let N_i be the number of workers that are assigned with the data batch i, and define $\bar{N} = (N_1, N_2, \ldots, N_B)$ as the assignment vector. Recall that the service times of the worker nodes (the computation time and the time it takes to communicate the results) are assumed to be exponentially distributed with rate λ.

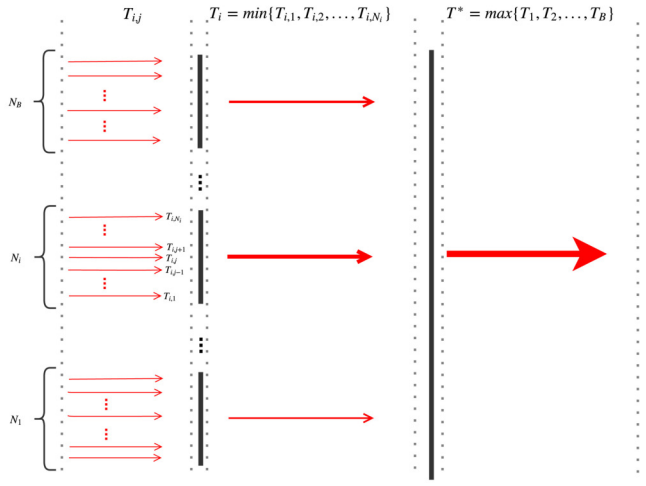


Fig. 3. Time diagram of the system in Fig. 1.

As illustrated in Fig. 3, N_i workers start computations over batch i simultaneously. Thus, the result from the fastest worker, out of N_i, is sufficient for the master to recover the computation over batch i. Therefore, the i'th batch recovery time is the first order statistics of N_i i.i.d exponential random variables, given by

$$T_i = \min\left(T_{i,1}, T_{i,2}, \ldots, T_{i,N_i}\right), \quad \forall i \in \{1, 2, \ldots, B\}, \quad (5)$$

where $T_{i,j}$s are i.i.d exponential random variables with rate λ, i.e. $T_{i,j} \sim \exp(\lambda)$. According to the exponential distribution properties [19], the recovery time of batch i is also exponentially distributed with rate $N_i\lambda$,

$$T_i \sim \exp\left(N_i\lambda\right), \quad \forall i \in \{1, 2, \ldots, B\}. \quad (6)$$

For generating the overall result, the master node has to wait for the results of computing over 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 overall result generating time, T, could be written as,

$$T = \max\left(T_1, T_2, \ldots, T_B\right). \quad (7)$$

Here T is the maximum order statistics of B exponential random variables.

The question we address next is how should one redundantly assign B non-overlapping batches of data among $N > B$ workers to achieve the shortest expected service time in System 1. To this end, we next introduce several concepts.

Definition 1. *The real valued random variable X is greater than or equal to the real valued random variable Y in the sense of **usual stochastic ordering**, shown by $X \underset{st}{\geq} Y$, if their tail distributions satisfy*

$$P\{X > \beta\} \geq P\{Y > \beta\}, \quad \forall \beta \in \mathcal{R}, \quad (8)$$

or equivalently,

$$E[\phi(X)] \geq E[\phi(Y)],$$

for any non-decreasing function ϕ.

Definition 2. *The random variable $X(\theta)$ is **stochastically decreasing and convex** if its tail distribution, $\bar{F}_X(x) = P\{X > x\}$, is a pointwise decreasing and convex function of θ.*

Definition 3. *For any $V_p = (v_{p1}, v_{p2}, \ldots, v_{pM})$ in $\mathcal{R}^M$, 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, $v_{[p1]} > v_{[p2]} > \cdots > v_{[pM]}$.*

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 $\mathcal{R}^M$. Then V_p **majorizes** V_q, denoted by $V_p \succeq V_q$, if*

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

Definition 5. *A real valued function $\phi : \mathcal{R}^M \to \mathcal{R}$ is **Schur convex** if for every V and W in $\mathcal{R}^M$, $V \succeq W$ implies $\phi(V) \geq \phi(W)$.*

Definition 6. *A real valued random variable $Z(\bar{x})$, $\bar{x} \in \mathcal{R}^M$, is **stochastically schur convex** , in the sense of usual stochastic ordering, if for any $\bar{x}$ and $\bar{y}$ in $\mathcal{R}^M$, $\bar{x} \succeq \bar{y}$ implies $Z(\bar{x}) \geq Z(\bar{y})$.*

Lemma 1. *If the batch assignment $\bar{\mathcal{N}}_1 = (N_{11}, N_{12}, \ldots, N_{1B})$ majorizes the batch assignment $\bar{\mathcal{N}}_2 = (N_{21}, N_{22}, \ldots, N_{2B})$, that is, $\bar{\mathcal{N}}_1 \succeq \bar{\mathcal{N}}_2$, then the corresponding service times $T(\bar{\mathcal{N}}_1)$ and $T(\bar{\mathcal{N}}_2)$ satisfy*

$$E[T(\bar{\mathcal{N}}_1)] \geq E[T(\bar{\mathcal{N}}_2)],$$

when the service time of the workers are independent and exponentially, identically distributed.

Proof. The service time for batch assignment policy $\bar{\mathcal{N}}_k$, $\forall k \in \{1, 2\}$, is given by

$$T(\bar{\mathcal{N}}_k) = \max\left(T_{k1}, T_{k2}, \ldots, T_{kB}\right),$$

where T_{ki}s are exponentially distributed with rate $N_{ki}\lambda$. Form Definition 2, T_{ki} $\forall i \in \{1, 2, \ldots, B\}$, is stochastically decreasing and convex function of N_{ki}. Therefore, $T(\bar{\mathcal{N}}_i)$, which is a maximum of stochastically decreasing and convex functions, is stochastically decreasing and schur convex function of $\bar{\mathcal{N}}_i$, [20]. Hence, by definition, $\bar{\mathcal{N}}_1 \succeq \bar{\mathcal{N}}_2$ implies $T(\bar{\mathcal{N}}_1) \geq T(\bar{\mathcal{N}}_2)$ in the sense of usual stochastic ordering. Hence, for any non-decreasing function ϕ,

$$E[\phi(T(\bar{\mathcal{N}}_1))] \geq E[\phi(T(\bar{\mathcal{N}}_2))].$$

Substituting ϕ by unit ramp function completes the proof. □

Proposition 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 batchs, is majorized by any other batch assignment policy.

Proof. See [21]. □

Then Theorem (1) follows immediately from Lemma 1.

Theorem 1. *With exponentially distributed service time of workers, among all (non-overlapping) batch assignment policies, the balanced assignment achieves the minimum expected service time for the overall result generation.*

Proof. From Proposition 2, $\bar{\mathcal{N}}_a \succeq \bar{\mathcal{N}}_b$ for any arbitrary batch assignment strategy $\bar{\mathcal{N}}_a$ and from Lemma 1, and by choosing ϕ to be the constant function, the following inequality holds:

$$E[T(\bar{\mathcal{N}}_a)] \geq E[T(\bar{\mathcal{N}}_b)],$$

which means that among all non-overlapping batch assignment policies, the balanced assignment achieves the minimum expected computing time. □

Now, in the following proposition we precisely quantify the minimum expected computing time, which could be achieved by the balanced assignment.

Proposition 3. *The lower bound of the expected time for overall result generation in System 1, with non-overlapping batches, when the service time of the workers are exponentially distributed with rate λ, is $\frac{B}{N\lambda}H_B$.*

Proof. From (7), the overall result generation time is the maximum order statistics of B i.i.d random variables. With balanced assignment of non-overlapping batches and exponentially distributed service time of workers, T_is are also exponential with rate $\frac{N\lambda}{B}$. On the other hand, the expected value of the maximum order statistics of B i.i.d exponential random variables with rate μ is $\frac{1}{\mu}H_B$, [22]. Substituting μ by $\frac{N\lambda}{B}$ completes the proof. □

B. Computing Time with Overlapping Batches

Recall the original problem of assigning a data set of size S redundantly among N workers. We assumed $S = N$. Each worker is assigned N/B data blocks, for a given parameter B. Furthermore, the number of copies of each data block is identical and each worker hosts at most one copy of a block. Therefore, each data block will be available at exactly N/B workers.

Suppose that the data set is divided into N overlapping batches, each with size N/B, in a cyclic order, such that the first batch consists of blocks 1 through N/B, the second batch is comprised of blocks 2 through $N/B+1$, and so on. An example of this batching is given in Fig. 4(a). With this data batching policy, which we call it *cyclic batching*, no two batches share the exact same data blocks but, on the other hand, the number of batches which share at least one data block with a specific batch is maximum. Specifically, with cyclic batching, each bach shares at least one common data block with $2\,(N/B-1)$ other batches. On the other hand, with non-overlapping batches, as it is shown in Fig. 4(c), each batch shares exactly N/B data blocks with $N/B-1$ other batches. With any other batching policy each batch shares same data blocks with more than $N/B-1$ and less than $2\,(N/B-1)$ other batches, an example of which is given in Fig. 4(b). In what follows, we will provide analytic comparison of the computing time of System 1, with three different batching policies, provided in Fig. 4.

Consider System 1, with $N = S = 6$, $B = 3$, and three different batching policies, as shown in Fig. 4. In each policy, there are two groups, shown by different shapes. Each group comprises the entire data set, in all three policies. However, the placement of blocks in groups is different across batching policies. Let $X_i\ \forall i \in \{1, 2, \ldots, 6\}$ be the i.i.d random vector of workers service time. Let us assume, without loss of generality, that W_1 is the fastest worker, delivering its local results before the rest of the workers. Then the overall result generating time for policy 4(a) is

$$T^*_{(a)} = \min\left(\max\left(X_3, X_5\right), \max\left(X_2, X_4, X_6\right)\right). \tag{9}$$

For policy 4(b), the overall result generating time could be written as,

$$T^*_{(b)} = \min(\max(X_3, \min(X_5, X_6)), \tag{10}$$

$$\max\left(\max\left(X_2, X_4\right), \min\left(X_5, X_6\right)\right). \tag{11}$$

Comparing (9) and (11), it is easy to see that $E[T^*_{(b)}] < E[T^*_{(a)}]$, since,

$$E[\max\left(X_3, \min\left(X_5, X_6\right)\right)] < E[\max\left(X_3, X_5\right)],$$

$$E[\max\left(\min\left(X_5, X_6\right), \max\left(X_2, X_4\right)\right)] < E[\max\left(X_2, X_4, X_6\right)].$$

On the other hand, for policy 4(c),

$$T^*_{(c)} = \max\left(\min\left(X_3, X_4\right), \min\left(X_5, X_6\right)\right). \tag{12}$$

In order to be able to compare the computing time of 4(c), we rewrite (11) as follows:

$$T^*_{(b)} = \max\left(\min\left(X_3, \max\left(X_2, X_4\right)\right), \min\left(X_5, X_6\right)\right). \tag{13}$$

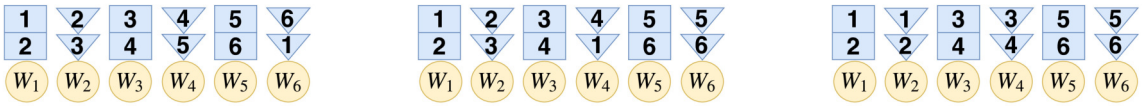
The first argument of the outmost max functions in (12) and (13) are compared as,

$$E[\min\left(X_3, X_4\right)] < E[\min\left(X_3, \max\left(X_2, X_4\right)\right)].$$

Therefore, $E[T^*_{(c)}] < E[T^*_{(b)}]$. Accordingly, the expected computing times of three batching policies in Fig. 4 are compared as follows:

$$E[T^*_{(c)}] < E[T^*_{(b)}] < E[T^*_{(a)}], \tag{14}$$

which essentially means that, the balanced assignment of non-overlapping batches achieves the minimum expected computing time when compared to overlapping batch assignment.



(a) Cyclic batching. Groups does not share same subset of the data set.

(b) Groups share a same subset of data set at W_5 and W_6.

(c) Groups share same subsets of data set (non-overlapping batches).

Fig. 4. Overlapping batches with $N = S = 6$ and $B = 3$.

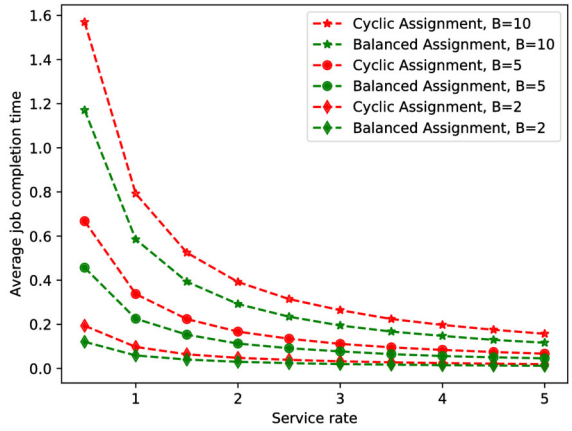


Fig. 5. The comparison of the average job completion time for cyclic assignment 4(a) and balanced assignment 4(c), with $N = 50$ worker nodes and three different B parameters.

V. Numerical Results

In this section we numerically evaluate the analytical results obtained in Sec. IV. In particular, we compare the computing time of balanced non-overlapping-batch assignment with cyclic overlapping-batch assignment.

In Fig. 5, the average computing time of balanced and cyclic assignments are plotted versus the service rate of the worker nodes for three different values of parameter B and $N = S = 50$. It can be seen that balanced assignment of the non-overlapping batches achieves lower computing time compared to cyclic assignment, for all three Bs and service rates. This observation is in line with our earlier analytic result in (14). Moreover, the gap between the two assignment polices gets larger in the low service rate region, which could be explained as follows. With exponentially distributed service times of workers, the time between each consequent local result deliveries to the master node is also exponentially distributed (due to the memoryless property of the exponential distribution), the average of which increases as the service rate of the workers decrease. On the other hand, from our analytic results we know that on average with the cyclic assignment, the master node would have to wait for more worker nodes to respond, compared to balanced assignment. Therefore, with lower service rate, the difference between computing times of the two assignment policies should increase. Besides, the performance gap decreases as the parameter B decrease. In order to explain this behaviour, recall that the number of data blocks at each machine is N/B. Hence, for smaller Bs, each worker does a larger portion of the computation and the master node should wait for a smaller number of worker nodes, which results in smaller performance gap between the two policies.

VI. Conclusion

The computing time of fault-tolerant distributed computing algorithms was analyzed. Specifically, the two recently proposed algorithms with focus on using error correcting codes and random task assignment were studied. It was analytically showed that, algorithms with the same amount of redundancy and failure tolerance would have different expected computing time. It was proved that, with a fixed number of computing tasks and exponentially distributed service time of the worker nodes, balanced assignment of non-overlapping data batches achieves minimum expected computing time, which was also validated by the simulation results.

References

[1] A. D. Kshemkalyani and M. Singhal, *Distributed computing: principles, algorithms, and systems*. Cambridge University Press, 2011.

[2] J. Choi, "A new parallel matrix multiplication algorithm on distributed-memory concurrent computers," *Concurrency: practice and experience*, vol. 10, no. 8, pp. 655–670, 1998.

[3] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[5] A. Elser, *Reliable distributed systems: technologies, web services, and applications*. Springer Science & Business Media, 2005.

[6] Y. Brun, G. Edwards, J. Y. Bang, and N. Medvidovic, "Smart redundancy for distributed computation," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011, pp. 665–676.

[7] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding," *arXiv preprint arXiv:1612.03301*, 2016.

[8] W. Halbawi, N. Azizan-Ruhi, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," *arXiv preprint arXiv:1706.05436*, 2017.

[9] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," *arXiv preprint arXiv:1710.09990*, 2017.

[10] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems*, 2017, pp. 4406–4416.

[11] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 3, pp. 7–11, 2015.

[12] M. F. Aktas, P. Peng, and E. Soljanin, "Effective straggler mitigation: Which clones should attack and when?" *arXiv preprint arXiv:1710.00748*, 2017.
[13] ——, "Straggler mitigation by delayed relaunch of tasks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 2, pp. 224–231, 2018.
[14] E. W. Weisstein, "Stirling number of the second kind," *triangle*, vol. 7, p. 8, 2002.
[15] A. N. Myers and H. S. Wilf, "Some new aspects of the coupon collector's problem," *SIAM review*, vol. 48, no. 3, pp. 549–565, 2006.
[16] L. Comtet, *Advanced Combinatorics: The art of finite and infinite expansions*. Springer Science & Business Media, 2012, ch. 5.
[17] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.
[18] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*. IEEE, 2010, pp. 1–10.
[19] M. Desu *et al.*, "A characterization of the exponential distribution by order statistics," *The Annals of Mathematical Statistics*, vol. 42, no. 2, pp. 837–838, 1971.
[20] L. Liyanage and J. G. Shanthikumar, "Allocation through stochastic schur convexity and stochastic transposition increasingness," *Lecture Notes-Monograph Series*, pp. 253–273, 1992.
[21] A. W. Marshall, I. Olkin, and B. C. Arnold, *Inequalities: theory of majorization and its applications*. Springer, 1979, vol. 143.
[22] C. S. Crow IV, D. Goldberg, and W. Whitt, "Two-moment approximations for maxima," *Operations research*, vol. 55, no. 3, pp. 532–548, 2007.