Coded Elastic Computing

Yaoqing Yang, Pulkit Grover, Soummya Kar
Carnegie Mellon University
{yyaoqing, pgrover, soummyak}@andrew.cmu.edu

Matteo Interlandi, Saeed Amizadeh, Markus Weimer
Microsoft
{mainterl, saamizad, mweimer}@microsoft.com

Abstract

Cloud providers have recently introduced low-priority machines to reduce the cost of computations. Exploiting such opportunity for machine learning tasks is challenging inasmuch as low-priority machines can *elastically* leave (through *preemption*) and join the computation at any time. In this paper, we design a new technique called *coded elastic computing* enabling distributed machine learning computations over elastic resources. The proposed technique allows machines to transparently leave the computation without sacrificing the algorithm-level performance, and, at the same time, flexibly reduce the workload at existing machines when new machines join the computation. Thanks to the redundancy provided by encoding, our approach is able to achieve similar computational cost as the original (uncoded) method when all machines are present; the cost gracefully increases when machines are preempted and reduces when machines join.

1 Introduction

New offerings from cloud-service providers allow exploiting under-utilized Virtual Machines (VMs) at a fraction of the original cost [3, 1]. For example, Azure Batch provides low-priority machines at about one-fifth of the cost of ordinary virtual machines [4]. Similarly, Amazon Spot Instances provide machines at market price with a discount which can reach up to 90% with respect to regular on-demand prices [2]. Such offerings, however, have the drawback that machines can be preempted at any time if a high-priority job appears. This, in turn, will surface as a computation failure at the application level. While common distributed machine learning frameworks are already built with fault-tolerance [19, 5], they often assume that failures are transient and rare. Due to this assumption, machine failures are often recovered by a "stop-the-world" scheme whereby the entire system is forced to wait until regular execution on the failure machines are restored from previous state (eventually on new machines). The above assumptions, however, do not necessarily hold for failures due to machines being preempted because (1) these failures are permanent and local data may not be accessible anymore; (2) several machines can be preempted altogether (up to 90% [18]); (3) these failures add up to transient failures, therefore leading to more frequent disruptions during computation; and (4) the computational framework may need to acquire additional machines to compensate, meaning that data has to be copied on the new machines which will likely become stragglers for the running computation. In practice, we observed situations at scale where the stop-the-world scheme results in zero computation progress because, by the time a failure is recovered, a new failure occurs. This results in the necessity to build an elastic run-time framework [20, 6] and related failure-aware algorithms which can continue the computation and flexibly adapt in the presence of failures. Another possible technique to deal with preemption is to view the preempted machines as erasure-type faults and ignore them. Although machine learning algorithms are robust to small transient faults, simply ignoring the computational results in these permanently-failed preempted machines may result in algorithmic-level performance loss [20]. The influence of ignoring the computation results for the preemption type of faults is also more severe than usual computation faults because the number of failures can be really large [18]. Similarly, even if the data are redundant

in some applications, ignoring partial results may still lead to reduced confidence levels on the prediction accuracy. It may also not be desirable from a customer's perspective who often requires the full dataset to be present during the entire training process in order to achieve the highest accuracy.

In order to deal with the aforementioned problems, in this paper we propose *coded elastic computing*: a novel distributed learning framework allowing users to train their machine learning models over preemptable machines. In our coded elastic computing framework machines are allowed to arbitrarily join or leave during a distributed iterative learning task thanks to the introduction of redundancy in the computation. Although coded elastic computing introduces redundancy using ideas from error-correcting codes [15, 21, 9, 17, 26, 14, 24, 11, 10], and can let the computation continue when a preemption failure happens like ordinary error-correcting codes, the way it utilizes the coded data to deal with preemption is fundamentally different from existing works. More specifically, coded elastic computing can flexibly change the workload of each machine at runtime based on the number of available machines by selecting to use only a subset of the encoded data in a cyclic fashion. Apart from providing fault-tolerance when machines are preempted, coded elastic computing is also *positively-elastic* in that it can utilize the properties of the coded data to reduce the workload at existing machines flexibly when new machines join the computation. We will show that the coded elastic computing framework can make the computational cost at each machine scale inversely with the number of machines, which leads to linear scaling of theoretical computational cost. The proposed technique is also useful in other applications besides elastic computation when the number of machines needs to be dynamically adjusted during a learning task, such as when the number of machines is tuned as a hyper-parameter, or when the machines have to be reallocated to achieve fairness [13] between users or the specific need of some users at runtime.

In this short paper, we first present a coded elastic matrix-vector multiplication algorithm to illustrate the main idea. Then, we discuss the applicability of the proposed technique over broader learning algorithms. Finally, we validate the approach with a set of experiments. More generalizations will be provided online [22].

2 Resource-Elastic Coded Distributed Computing

We initially focus on the problem of matrix-vector multiplication for having a better theoretical understanding of coded elastic computing. Before presenting the algorithm in Section 2.3, we introduce the main idea of the paper in Section 2.1 and Section 2.2.

We consider the case when a data matrix $\mathbf{X}$ is stored distributedly at several machines. Suppose the number of machines is P and, in the worst-case of preemption failures, there are at least L machines that remain[1]. We will show that the parameter L is also equal to the *recovery threshold*, the meaning of which will be clear in Section 2.1. In this paper, we consider repeatedly using the same data but with different input vectors. For matrix multiplications, it means that we compute $\mathbf{X}\mathbf{w}_t$ for $t = 1, 2, \ldots$ for the same $\mathbf{X}$. This computation primitive is applicable in training linear models [25, 23, 12], PageRank [24], model-parallel deep neural networks [8] and many other machine learning algorithms at the inference stage.

Suppose there are *elastic events* thereby existing machines can be preempted and new machines can be added to the computation. To characterize the key components of these elastic events, we state the following three properties:

Property 1. When a preemption failure happens, which machine(s) to be preempted is decided by the resource allocator and is not known in advance.

Property 2. The preemption is permanent, meaning that the preempted machines are going to be removed forever. However, new machines may join after an unknown time.

Property 3. After an elastic event happens, the entire system gets the information. That is, if some machines leave or join, the other machines know immediately about which machines leave or join.

The second and the third properties differentiate the elastic failures from common faults and stragglers because (1) new machines can join the computation, and (2) one may adapt the computation scheme instantly after an elastic event and possibly utilize the newly available resources. In Section 2.1 and Section 2.2, we provide the elastic data partitioning scheme to utilize resource elasticity actively.

[1]The parameter L, or a lower bound of L, is needed for exact computation. However, in many machine learning tasks, one can often optimize with a subset of data due to data redundancy. In that case, knowledge of L is not necessary.

2.1 Coded Data Partitioning in the Presence of Preempted Machines

We partition the data matrix $\mathbf{X}$ into L subsets $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_L$ of equal size. If the total number of data points is not divisible by L, we can use zero-padding. Next, we will use $\mathbf{X}_k$ to denote the matrix composed by the k-th subset of data, $k = 1, 2, \ldots, L$. Finally, we generate P *coded* data matrices $\mathbf{X}_s^{\text{coded}}, s = 1, 2, \ldots, P$, $(P > L)$, in which each matrix is a linear combination of the form:

$$\mathbf{X}_s^{\text{coded}} = \sum_{k=1}^{L} g_{s,k} \mathbf{X}_k \tag{1}$$

where each $g_{s,k}$ is a random coefficient. The P coded data matrices are distributed to P workers. In this way, the coded data $\mathbf{X}_s^{\text{coded}}, s = 1, 2, \ldots, P$ satisfy the following property with probability one for a variety of choices of the linear coefficients $g_{s,k}$, e.g., if $g_{s,k}$'s are i.i.d. Gaussian random variables:

Lemma 2.1. Suppose we want to compute the matrix-vector product $\mathbf{X}\mathbf{w}$. Then, any L out of P coded computation results $\mathbf{X}_s^{\text{coded}}\mathbf{w}, s = 1, 2, \ldots, P$ are sufficient to recover the original (uncoded) computation results $\mathbf{X}_k\mathbf{w}, k = 1, 2, \ldots, L$, which are equivalent to $\mathbf{X}\mathbf{w}$.

The recovery of the results is through solving L linear systems of the form $\mathbf{X}_s^{\text{coded}}\mathbf{w} = \sum_{k=1}^{L} g_{s,k}\mathbf{X}_k\mathbf{w}$. Lemma 2.1 is critical for the failure recovery. It essentially shows that no matter which machines are preempted, as long as the number of remaining machines is not smaller than L, the *whole information* of the original data is preserved in the remaining machines. This is why we call the parameter L the *recovery threshold*. The parameter L is limited by the storage constraint at each machine. The more redundancy we can add to the data, the lower recovery threshold we need, and hence more failures we can tolerate. In our experiments, we use a redundancy factor of P/L = 2, and hence we can at maximum tolerate failures when half of the machines are preempted. An often-used coding technique is called *systematic code*, in which the linear coefficients satisfy

$$g_{s,k} = \mathbf{1}_{\{s=k\}}, \text{ if } s \leq L. \tag{2}$$

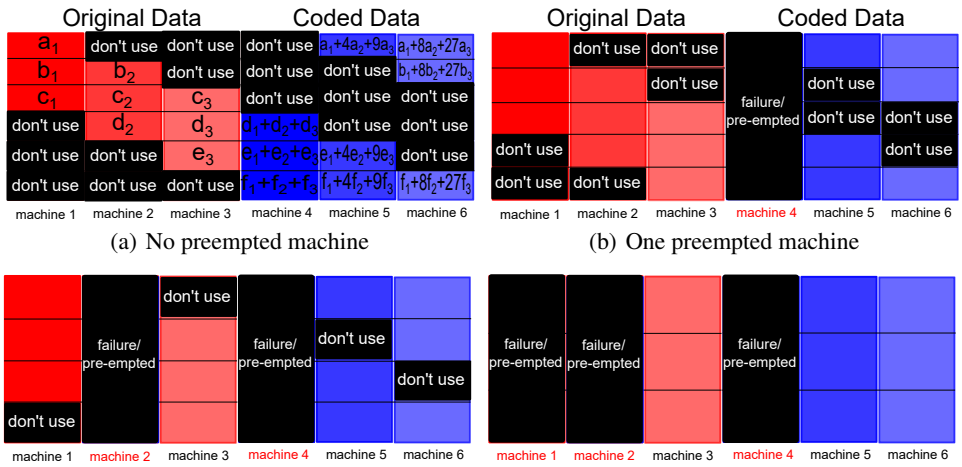
In this case, the coded data at the first L machines $\mathbf{X}_s^{\text{coded}}, s = 1, 2, \ldots, L$ are the original data $\mathbf{X}_k, k = 1, 2, \ldots, L$. This can provide backward-compatibility to switch between coded computing and uncoded ordinary computing, and at the same time significantly reduce the cost of encoding the data at the preprocessing stage (1).

2.2 Elastic Data Partitioning for Elastic Computation by Using Data In a Cyclic Way

According to Lemma 2.1, as long as the number of machines that are not preempted is greater or equal to L, the remaining data using the coded data partitioning can preserve the whole information of the original data. However, when the number of machines is strictly larger than L, it becomes redundant to use all the coded data because the data at L machines already preserve the whole information. To positively utilize all the remaining machines and achieve the parallel computing capabilities of the extra machines, we select to use data in a cyclic fashion as shown in Figure 1. We use a systematic code (see (2)) by which the first L of the P coded blocks $\mathbf{X}_s^{\text{coded}}, s = 1, 2, \ldots, P$ are the original data $\mathbf{X}_k, k = 1, 2, \ldots, L$. In Figure 1 we use red to denote original data and blue to denote the remaining coded data. In this example, the initial number of machines is $P = 6$ and the recover threshold is $L = 3$. From figure (a) to (d), we show how to continue the computation when machines are gradually preempted from 6 to 3 (machines correspond to the columns). Each machine is initially allocated a single subset of coded data $\mathbf{X}_s^{\text{coded}}, s = 1, 2, \ldots, P$, among which L subsets are the original data. Each subset of data is represented as a column in each subfigure of Figure 1.

If no failures occur (see Figure 1(a)), to remove redundancy from the data, we partition each data block (column) into P *sub-blocks*, and let each machine only use L out of P sub-blocks. By a sub-block of data, we mean one small rectangle in Figure 1. If $M \geq 1$ machines are preempted (see Figure 1(b)-(d)), we partition each data block into $P - M$ sub-blocks, and still let each node only use L out of $P - M$ sub-blocks. If new machines join, they download the coded data previously used in some failed machines, and all the machines, including the ones that just join, use elastic data partitioning based on the current number of available machines.

There are two advantages of this type of data usage (1) the overall selected data to use is of the same size as the original data and the selected data across all remaining machines have the same size; and (2) the selected data preserve the whole information of the original data, according to Lemma 2.1. Thus, we can exactly recover the results while removing the redundancy in the way of using data. These two properties will be formally introduced and proved in Theorem 2.1.



(a) No preempted machine (b) One preempted machine

(c) Two preempted machines (d) Three preempted machines

Figure 1: The main idea of elastic data partitioning is to use the data in a cyclic way. Each column of data is stored at one machine. For each group (i.e., row block) of data at different machines, there are enough number of sub-blocks that contain all the information. This cyclic way of using data leads to linear scaling of the computational cost in the number of machines.

2.3 Coded Elastic Computing for Matrix-vector Multiplications

We provide the details of the coded elastic computing algorithm for the repeated matrix-vector multiplication problem $\mathbf{X}\mathbf{w}_t, t = 1, 2, \ldots$ in Algorithm 1. We use $\mathbf{X}_{k,j}^{\text{coded}}$ to represent the j-th sub-block of the data at the k-th machine. We will call $\mathbf{X}_{k,j}^{\text{coded}}$'s with the same j "the j-th group" of sub-blocks which correspond to the j-th row block in any subfigure of Figure 1. We use $\mathbf{G}_j$ to represent the collection of linear combination coefficients for the j-th group (row block) that are selected to use. For example, for the first group (row block) in Figure 1(a), we have $\mathbf{G}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 4 & 9 \\ 1 & 8 & 27 \end{bmatrix}$ because the three selected sub-blocks are $\mathbf{a}_1$, $\mathbf{a}_1 + 4\mathbf{a}_2 + 9\mathbf{a}_3$ and $\mathbf{a}_1 + 8\mathbf{a}_2 + 27\mathbf{a}_3$.

Algorithm 1 Coded Elastic Computing for Matrix-Vector Multiplication

Input: The data matrix $\mathbf{X}$, the number of machines P, the recovery threshold L, the linear combination coefficients $g_{s,k}$'s in equation (1) and the sequence of input vectors $\mathbf{w}_t$ $t = 1, 2, \ldots$.
Preprocessing: Partition the data $\mathbf{X}$ into L subsets and compute the coded subsets as in (1).
Online computation:
FOR each computation with input $\mathbf{w}_t$:
  Broadcast: The master node sends $\mathbf{w}_t$ to each worker.
  FOR each group index j:
    Gather: The k-th worker computes $\mathbf{u}_{t,k,j} = \mathbf{X}_{k,j}^{\text{coded}}\mathbf{w}_t$ and sends $\mathbf{u}_{t,k,j}$ to the master. The master gathers vectors $\mathbf{u}_{t,k,j}$ for all workers that use the j-th sub-block and obtains the matrix $\mathbf{u}_{t,j}$ which contains the results for the j-th group (row block).
    Decode: The master node computes $\mathbf{u}_{t,j}\mathbf{G}_j^{-1}$ to obtain the results for the j-th group.
  Output: The master node outputs $\mathbf{X}\mathbf{w}_t$.

Theorem 2.1. The coded elastic computing algorithm achieves the exact computation result of $\mathbf{X}\mathbf{w}_t$ for all t. Moreover, the size of the selected data to use at each machine is the same, and the overall size of the selected data is the same as the size of the original data.

Proof. Recall that we call the sub-blocks on the same row block (in Figure 1) of the P different blocks of data a group, and we use $\mathbf{X}_{k,j}^{\text{coded}}$ to represent the coded sub-block of data that is at the k-th machine and belongs to the j-th group. Let $\mathbf{X}^j$ be the collections of original data that belongs to the j-th group. For example, in Figure 1(a), $\mathbf{X}^6$ represents the collection of original data $[\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3]^\top$.

Then, from the cyclic way of using data, we can see that when each machine uses L sub-blocks, the overall number of used sub-blocks in each group is L. From Lemma 2.1, the results $\mathbf{u}_{t,k,j} = \mathbf{X}_{k,j}^{\text{coded}}\mathbf{w}_t$ for all the workers that use the coded data in the j-th group can be collected together to decode $\mathbf{X}^j\mathbf{w}_t$.

The other claim can be seen from the figure, i.e., the area of the used data is always equal to the area of the original data, and the area of the used data is the same across all remaining machines. We prove this claim as follows. The size of the data in each of the L subsets is N/L. Thus, the used data at each machine is the same number $\frac{N}{L}\frac{L}{P-M} = N/(P-M)$. There are $P-M$ machines left, so the overall size of the used data is $\frac{N}{P-M}\cdot(P-M) = N$, which is the same as the original data. □

Theorem 2.1 shows that our technique uses the same size of data as the original (uncoded) case. This is desirable for memory-bound applications.

Remark 1. (Cost analysis) The encoding (preprocessing step) is a one-time cost. The decoding done at the master has computational cost $\mathcal{O}(PN)$, while the matrix-multiplication step at each worker has cost $\mathcal{O}(dN/P)$, where P is the number of workers and $N \times d$ is the size of the matrix $\mathbf{X}$. Thus, the decoding cost is smaller than the computational cost at each worker as long as $d = \Omega(P^2)$. Even if $d < \Omega(P^2)$, we can partition the machines into smaller groups and respectively code each group. The decoding complexity can be further reduced to $\mathcal{O}(N\log^2 P)$ if Vandermonde systems are used [16].

3 Coded Elastic Computing for Linear Models

The cyclic way of elastic data partitioning applies to general coded computing techniques proposed thus far and is not limited to matrix-vector multiplications. Due to the space limitation, we focus on one application of coded computing for linear regression [15]. Consider the linear objective function:

$$f(\mathbf{w};\mathbf{X},\mathbf{y}) = \sum_{i=1}^{n}(\mathbf{w}^\top\mathbf{x}_i - \mathbf{y}_i)^2 + h(\mathbf{w}). \tag{3}$$

The vanilla gradient descent has the form $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\mathbf{g}_t$ and $\mathbf{g}_t = \mathbf{X}^\top(\mathbf{X}\mathbf{w}_t - \mathbf{y}) + \partial_{\mathbf{w}}h(\mathbf{w}_t)$. When the data matrix $\mathbf{X}$ is large, the most time-consuming part is the computation of $\mathbf{X}^\top(\mathbf{X}\mathbf{w}_t - \mathbf{y})$. We thus extend Algorithm 1 in the following way to compute $\mathbf{X}^\top(\mathbf{X}\mathbf{w}_t - \mathbf{y})$:

- Compute $\mathbf{X}^j\mathbf{w}_t$ (where $\mathbf{X}^j$ is the data in the j-th group, or the j-th row block in Figure 1 across different machines) for all group-index j in an elastic way using Algorithm 1;
- The master computes $\mathbf{z}_t^j = \mathbf{X}^j\mathbf{w}_t - \mathbf{y}^j$ for all group-index j, where $\mathbf{y}^j$ are the labels corresponding to the data points in the j-th group;
- The master re-encodes $\mathbf{z}_t^j$'s using the (pre-computed) inverse generator matrix $(\mathbf{G}_j^{-1})^\top$ to obtain $(\mathbf{G}_j^{-1})^\top\mathbf{z}_t^j$, and scatters the results to the workers that use the j-th group of data;
- Since data at workers are encoded using $\mathbf{G}_j$, the reduced results from all the workers are

$$\sum_j(\mathbf{X}^j)^\top\mathbf{G}_j^\top(\mathbf{G}_j^{-1})^\top\mathbf{z}_t^j = \sum_j(\mathbf{X}^j)^\top(\mathbf{X}^j\mathbf{w}_t - \mathbf{y}^j) = \mathbf{X}^\top(\mathbf{X}\mathbf{w}_t - \mathbf{y}). \tag{4}$$

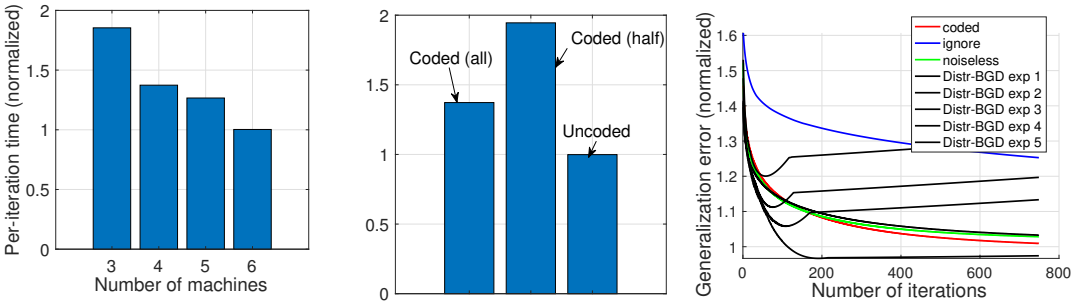
Note that in the above extension of Algorithm 1, the workers also utilize the data as in Figure 1.

4 Implementation and Experimental Evaluation

The proposed coded elastic computing technique has been implemented on top of Apache REEF [7] Elastic Group Communication (EGC) framework [2]. REEF EGC provides an API allowing to implement elastic computations by chaining fault-tolerant MPI-like primitives. In this short paper we assess the performance of our elastic code computing approach through 2 mini-benchmarks.

Matrix-vector mini-benchmark. In this mini-benchmark, we test that indeed the time cost decreases linearly with the increase in the number of machines available. We mimic an elastic computing environment on Amazon EC2 by using different numbers of t2.medium instances to compute the same matrix-vector product $\mathbf{X}\mathbf{w}$. The matrix is randomly generated and with size 30000×10000, and it is partitioned initially into 3 submatrices of size 10000×10000. Then, they are encoded

[2] `https://github.com/interesaaat/reef/tree/elastic-sync`



(a) Matrix-vector mini-benchmark (b) Linear model overhead (c) Linear model error

Figure 2: Mini-benchmarks experiments (results normalized due to confidentiality).

into 6 submatrices of the same size, and each submatrix is stored at one machine (for a total of 6 machines). To mimic the elastic events, we change the number of available machines by injecting artificial failures. The maximum number of failures is 3. The per-iteration overall time (including both communication and computation) is shown in Figure 2(a). The result is averaged using 20 independent trials. As we can see, the coded elastic computing technique can utilize the extra machines when the number of machines increases.

Linear model mini-benchmark. In this experiment, we test a coded implementation of linear regression using line-search-based batch gradient descent (the same setting as the baseline [20]). We run the test over 20 machines on a Microsoft internal multi-tenancy cluster. Each data point in the dataset has 3352 features, and we sample 10000 data for training and 10000 data for testing. We generate random failures and allow REEF EGC to reschedule new machines when failures occur. We start with Figure 2(b) where we plot the time for each iteration. In theory, when all the workers are present, the computational cost per iteration should be the same as the uncoded case. However, the coded method (all) has slight overhead due to decoding cost. The coded method (half) shows the cost when only half of the workers are running, which is, as expected, twice the cost of the uncoded method. In Figure 2(c) we report the generalization error and we compare our coded elastic computing technique with three baselines, namely noiseless (no failure), ignore the failure and continue, and an existing algorithm called Elastic Distr-BGD [20]. The coded method can achieve the same convergence behavior as the noiseless case, while the ignore method achieves worst generalization error. In the figure, we show 5 different experiments on Distr-BGD using the same failure probability but different realizations. The convergence of Distr-BGD depends on when a failure occurs and can lead to different algorithm performance. This is because the Distr-BGD keeps using previous gradient vectors at the failed machines, and this can (1) lead to overfitting, and (2) make the optimization miss the minimum point. In the plot of Distr-BGD, the *valley* part is due to overfitting, and the sudden change to near flat loss growth is because when the gradient descent has missed the optimal point of empirical training loss, the fixed gradient at the failure nodes makes the line search choose the smallest step size. In some cases, the Distr-BGD works extremely well because the fixed gradients act like momentum and can improve the speed of convergence.

From the experiment results, we can see that the coded elastic computing technique can obtain the same convergence behavior as ordinary gradient-descent-based algorithms but can elastically allocate the workload based on the number of available machines without moving data around.

5 Conclusions

The coded elastic computing framework presented in this paper can deal with new cloud offerings where machines can leave and join during the computation. Our framework handles the elastic events in a positive way, meaning that when machines leave, it shifts the computation to the remaining workers, and when new machines join the computation, it actively reduces the workload of existing machines without the reallocation of data. We prove that the coded elastic computing technique can achieve the same memory-access cost as the noiseless case, and hence is optimal for memory-bound applications. Using experiments in both Amazon EC2 and on a Microsoft multi-tenancy cluster, we show that the coded elastic computing technique can achieve the same convergence behavior as if no failure occurs, and can dynamically adjust working loads respect to the number of remaining workers.

References

[1] AWS Spot Instances. `https://aws.amazon.com/ec2/spot/`, 2018.

[2] AWS Spot Instances Prices. `https://aws.amazon.com/ec2/spot/pricing/`, 2018.

[3] Azure Batch. `https://docs.microsoft.com/en-us/azure/batch/batch-low-pri-vms`, 2018.

[4] Azure Batch Pricing. `https://azure.microsoft.com/en-us/pricing/details/batch/`, 2018.

[5] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[6] B.-G. Chun, T. Condie, Y. Chen, B. Cho, A. Chung, C. Curino, C. Douglas, M. Interlandi, B. Jeon, J. S. Jeong, et al. Apache REEF: Retainable evaluator execution framework. *ACM Transactions on Computer Systems (TOCS)*, 35(2):5, 2017.

[7] B.-G. Chun, T. Condie, Y. Chen, B. Cho, A. Chung, C. Curino, C. Douglas, M. Interlandi, B. Jeon, J. S. Jeong, G. Lee, Y. Lee, T. Majestro, D. Malkhi, S. Matusevych, B. Myers, M. Mykhailova, S. Narayanamurthy, J. Noor, R. Ramakrishnan, S. Rao, R. Sears, B. Sezgin, T. Um, J. Wang, M. Weimer, and Y. Yang. Apache reef: Retainable evaluator execution framework. *ACM Trans. Comput. Syst.*, 35(2):5:1–5:31, Oct. 2017.

[8] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover. A unified coded deep neural network training strategy based on generalized polydot codes. In *IEEE International Symposium on Information Theory (ISIT)*, pages 1585–1589. IEEE, 2018.

[9] S. Dutta, V. Cadambe, and P. Grover. Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Advances In Neural Information Processing Systems*, pages 2100–2108, 2016.

[10] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover. On the optimal recovery threshold of coded matrix multiplication. *arXiv preprint arXiv:1801.10292*, 2018.

[11] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover. On the optimal recovery threshold of coded matrix multiplication. In *Communication, Control, and Computing (Allerton), 2017 55th Annual Allerton Conference on*, pages 1264–1270. IEEE, 2017.

[12] F. Haddadpour, Y. Yang, M. Chaudhari, V. R. Cadambe, and P. Grover. Straggler-resilient and communication-efficient distributed iterative linear solver. *arXiv preprint arXiv:1806.06140*, 2018.

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

[14] C. Karakus, Y. Sun, S. Diggavi, and W. Yin. Straggler mitigation in distributed optimization through data encoding. In *Advances in Neural Information Processing Systems*, pages 5434–5442, 2017.

[15] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2018.

[16] L. Li. On the arithmetic operational complexity for solving vandermonde linear equations. *Japan journal of industrial and applied mathematics*, 17(1):15, 2000.

[17] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr. A fundamental tradeoff between computation and communication in distributed computing. *IEEE Transactions on Information Theory*, 64(1):109–128, 2018.

[18] K. Mahajan, M. Chowdhury, A. Akella, and S. Chawla. Dynamic query re-planning using QOOP. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 253–267, Carlsbad, CA, 2018. USENIX Association.

[19] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. Mllib: Machine learning in apache spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

[20] S. Narayanamurthy, M. Weimer, D. Mahajan, T. Condie, S. Sellamanickam, and S. S. Keerthi. Towards resource-elastic machine learning. In *NIPS 2013 BigLearn Workshop*, 2013.

[21] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pages 3368–3376, 2017.

[22] P. G. S. K. S. A. M. W. Y. Yang, M. Interlandi. Coded elastic computing. *to appear*, 2018.

[23] Y. Yang, M. Chaudhari, P. Grover, and S. Kar. Coded iterative computing using substitute decoding. *arXiv preprint arXiv:1805.06046*, 2018.

[24] Y. Yang, P. Grover, and S. Kar. Coded distributed computing for inverse problems. In *Advances in Neural Information Processing Systems*, pages 709–719, 2017.

[25] Y. Yang, P. Grover, and S. Kar. Coding for a single sparse inverse problem. In *IEEE International Symposium on Information Theory (ISIT)*, pages 1575–1579, 2018.

[26] 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*, pages 4403–4413, 2017.