Timely Coded Computing

Chien-Sheng Yang[†], Ramtin Pedarsani[*], and A. Salman Avestimehr[†]
[†] University of Southern California [*] University of California, Santa Barbara

***Abstract*—In modern distributed computing systems, unpredictable and unreliable infrastructures result in high variability of computing resources. Meanwhile, there is significantly increasing demand for timely and event-driven services with deadline constraints. Motivated by measurements over Amazon EC2 clusters, we consider a two-state Markov model for variability of computing speed in cloud networks. In this model, each worker can be either in a good state or a bad state in terms of the computation speed, and the transition between these states is modeled as a Markov chain which is unknown to the scheduler. We then consider a *Coded Computing* framework, in which the data is possibly encoded and stored at the worker nodes in order to provide robustness against nodes that may be in a bad state. Our goal is to design the optimal computation-load allocation strategy that maximizes the timely computation throughput (i.e, the average number of computation tasks accomplished before their deadline). Our main result is the development of a dynamic computation strategy called *Estimate-and-Allocate (EA)* strategy, which achieves the optimal timely computation throughput. Compared with the static allocation strategy, EA improves the timely computation throughput by $1.44\times \sim 4.6\times$ in experiments over Amazon EC2 clusters.**

I. Introduction

Large-scale distributed computing systems can substantially suffer from unpredictable and unreliable computing infrastructure which can result in high variability of computing resources, i.e., speed of the computing resources vary over time. The speed variation has several causes including hardware failure, co-location of computation tasks, communication bottlenecks, etc. [1] This variability is further amplified in computing clusters, such as Amazon EC2, due to the utilization of credit-based computing policy, in which the most commonly used T2 and T3 instances can operate significantly above a baseline level of CPU performance (approximately 10 times faster as shown in Fig. 1) by consuming CPU credits that are allocated periodically to the nodes. At the same time, there is a significant increase in utilizing the cloud for event-driven and time-sensitive computations (e.g., IoT applications and cognitive services), in which the users increasingly demand timely services with deadline constraints, i.e., computations of requests have to be finished within specified deadlines.

Our goal in this paper is to study the problem of computation allocation over cloud networks with particular focus on variability of computing resources and timely computation tasks. From the measurements of nodes' computation speeds over Amazon EC2 clusters, shown in Fig. 1, we observe that when a node is slow (fast), it is more likely that it continues to be slow (fast) in the following rounds of computation, which implies temporal correlation of computation speeds. Thus, to capture this phenomenon, we consider a two-state Markov model for variability of computing speed in cloud networks. In this model, each worker can be either in a good state or a bad state in terms of the computation speed, and the transition between these states is modeled as a Markov chain which is unknown to the scheduler.

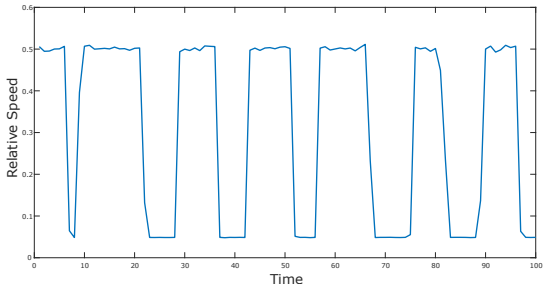

Fig. 1: Empirical measurement of speed variation of a credit-based `t2.micro` instance in Amazon EC2: A two-state Markov model.

Furthermore, we consider a *Coded Computing* framework, in which the data is possibly encoded and stored at the workers in order to provide robustness against workers that may be in a bad state. The key idea of coded computing is to encode the data and design each worker's computation task such that the fastest responses of any k workers out of total of n workers suffice to complete the distributed computation, similar to classical coding theory where receiving any k symbols out of n transmitted symbols enables the receiver to decode the sent message.

We consider a dynamic computation model, where a sequence of matrix multiplications needs to be computed over the (encoded) data that is distributedly stored at the nodes. More precisely, in an online manner, timely computation requests with given deadlines are submitted to the system, i.e., each computation has to be finished within the given deadline. Our goal is to design the optimal computation-load allocation strategy that maximizes the timely computation throughput (i.e, the average number of computation tasks that are accomplished before their deadline).[1]

One major challenge in this problem is the design of an adaptive computation load allocation strategy for the workers based on the history of previous computation times. In particular, the state of the nodes and the transition probabilities of the Markov model are unknown to the scheduler. Note that to find the optimal computation strategy, one has to solve a complex optimization which in general requires searching over all possible load allocations, even if the transition probabilities of the Markov model are known to the master. Thus, it's not clear how one allocates the computation loads efficiently and what computation strategy is optimal, especially for the network with unknown Markov model.

[1]Our metric of timely computation throughput is motivated by timely throughput metric, introduced in [2], which measures the average number of packets that are delivered by their deadline in a communication network.

As the main contributions of the paper, we propose a dynamic computation strategy called *Estimate-and-Allocate (EA)* strategy, and show that it achieves the optimal timely computation throughput. The EA strategy estimates the transition probabilities by observing the past events at each time step, and then assigns computation loads based on the estimated probabilities. Moreover, we show that finding the optimal load assignment using EA can be done efficiently instead of searching over all possible load allocations which is computationally infeasible to implement.

To prove the optimality of EA, we first provide an upper bound for the timely computation throughput by maximizing the success probability of each round when the transition probabilities are known to master. Then, we show that the success probability using EA converges to the optimal success probability. By the Strong Law of Large Numbers (SLLN), thr Ergodic theorem and a coupling argument, we finally prove that timely computation throughput achieved by the EA strategy matches with the derived upper bound, i.e., EA is optimal. Finally, we show that compared to the static allocation strategy, EA increases the timely computation throughput by $1.44\times \sim 4.6\times$ in experiments over Amazon EC2 clusters.

Beyond matrix multiplications, we note that the proposed dynamic computation strategy can be generalized to the computation model of multivariate polynomial functions [3].

Related Work: There are two lines of work that are most related to this paper: coded computing and task scheduling.

Coded computing broadly refers to a family of techniques that utilize coding to inject computation redundancy in order to alleviate the various issues that arise in large-scale distributed computing. In the past few years, coded computing has a tremendous success in various problems, such as straggler mitigation and bandwidth reduction (e.g., [4]–[9]). Coded computing has also been expanded in various directions, such as heterogeneous networks (e.g., [10]), partial stragglers (e.g., [11]), secure and private computing (e.g., [12]) and distributed optimization (e.g., [13]). However, none of the prior works in this area have considered deadline in computations as well as a dynamic network/computation setting, which are the focuses of this paper.

Dynamic task scheduling problem has been widely studied in the literature, where jobs arrive to the network according to a stochastic process, and get scheduled dynamically over time. The primary goal of dynamic scheduling is to find a throughput-optimal scheduling policy, i.e. a policy that stabilizes the network, whenever it can be stabilized. For example, Max-Weight scheduling, first proposed in [14], is known to be throughput-optimal for wireless networks, flexible queueing networks [15], [16], data centers networks [17] and dispersed computing networks [18]. Moreover, there have been many works which focus on task scheduling problem with deadline constraints over cloud networks (e.g. [19]).

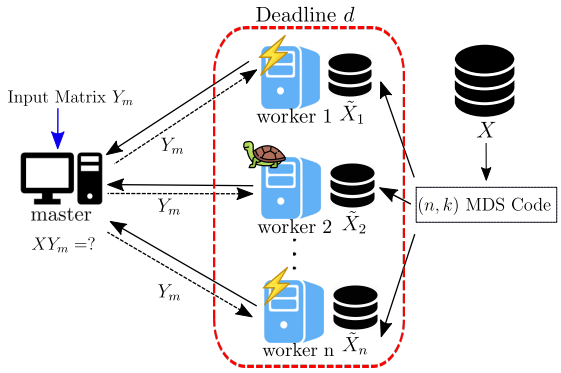

Fig. 2: Overview of timely coded computing. In each round m, the goal is to compute the matrix multiplication of data matrix X and the input matrix Y_m by the deadline d using n workers.

II. System Model

A. Computation Model

We consider a distributed computing problem, in which computation requests are submitted to a distributed computing system in an online manner, and the computation is carried out in the system. In particular, there is a fixed deadline for each computation round, i.e., each computation has to be finished within the given deadline.

As shown in Fig. 2, the considered system is composed of a master and n workers. There is a data matrix $X \in \mathbb{R}^{w\times q}$ which is divided to $X_1, X_2, \ldots, X_k \in \mathbb{R}^{\frac{w}{k}\times q}$. In each round m (or time slot in a discrete-time system), a computation request with a matrix $Y_m \in \mathbb{R}^{q\times t}$ is submitted to the system. We denote by d the deadline of each computation request which is smaller than or equal to the duration of each round. In such distributed computing system, we are interested in computing the matrix multiplications $X_1Y_m, X_2Y_m, \ldots, X_kY_m$ in each round m by the deadline d.

Prior to the computation, the master first encodes the dataset $X_1, X_2, \ldots, X_k$ to $\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_n$ using an (n, k) MDS code. Each worker i stores encoded data $\tilde{X}_i$ locally. In each round m, each worker i computes matrix multiplication of certain rows of $\tilde{X}_i$ and Y_m, which is determined by the master.

Given a matrix Y_m in round m, the master assigns the computations to each worker. More specifically, we denote $\vec{r}_m = (r_{m,1}, r_{m,2}, \ldots, r_{m,n})$ as the starting point vector, in which $r_{m,i}$ denotes the starting row of computation for worker i in round m. Also, we denote $\vec{\ell}_m = (\ell_{m,1}, \ell_{m,2}, \ldots, \ell_{m,n})$ as the load allocation vector, in which $\ell_{m,i}$ denotes the number of rows to be assigned to worker i in round m. Each worker i computes matrix multiplication of sub-matrix $[\tilde{X}_i]_{[r_{m,i}:r_{m,i}+\ell_{m,i}-1]_c}$ and Y_m, where $[A]_{[i:j]_c}$ denotes the sub-matrix of A which consists of row vectors with indices $\ell \pmod{\frac{w}{k}}$, $\forall i \le \ell \le j$. Each worker returns the results back to the master upon the completion of all assigned computations. Note that we index the rows of matrix $\tilde{X}_i$ from 0.

The master aggregates the results from the workers until it receives a *decodable* set of local computations. We say a set of computations is decodable if $X_1Y_m, X_2Y_m, \ldots, X_kY_m$ can be obtained by computing decoding functions over the received

results. Due to utilizing an MDS code, the master can recover $X_1Y_m, X_2Y_m, \ldots, X_kY_m$ if and only if the master receives at least k out of total of n results for each corresponding row of $\tilde{X}_1Y_m, \tilde{X}_2Y_m, \ldots, \tilde{X}_nY_m$. In each round, the goal of the master is to receive a decodable set of computations within the given deadline d.

B. Network Model

Motivated by the measurements over Amazon EC2 clusters, shown in Fig. 1, we assume that each worker has two different states for computing, *good state* and *bad state*. We denote μ_g as the computing speed (vector-matrix multiplications per second) in the good state, and denote μ_b as the computing speed in the bad state. Note that given a worker's state, its computation time (per vector-matrix multiplication) is deterministic. We denote $\mu_{m,i}$ as computing speed of worker i in round m. And, we denote $\vec{\mu}_m = (\mu_{m,1}, \mu_{m,2}, \ldots, \mu_{m,n})$ as the computing speed vector in round m. For each worker i, we model the state transitions as a stationary Markov process $S_i[1], S_i[2], \ldots$ with the transition matrix defined as follows:

$$P_i = \begin{bmatrix} p_{g\to g,i} & 1-p_{g\to g,i} \\ 1-p_{b\to b,i} & p_{b\to b,i} \end{bmatrix} \quad (1)$$

where $p_{g\to g,i}$ is the transition probability of worker i going to the good state from the good state, and $p_{b\to b,i}$ is the transition probability of worker i going to the bad state from the bad state. We assume that the Markov processes of different workers are mutually independent. Prior to the computation, we assume the initial state of worker i is given by the stationary distribution of Markov chain $(S_i[1], S_i[2], \ldots)$. We assume that the transition probabilities and current state of each worker are unknown to the master before the master assigns the computations to each worker.

C. Problem Formulation

Given the computation deadline d, we denote $N_m(d)$ as an indicator representing whether the computation is finished by deadline d, i.e., $N_m(d) = 1$ if the computation is finished by time d in round m, and $N_m(d) = 0$ otherwise. We denote $\eta = (\{(\vec{r}_m, \vec{\ell}_m)\}_{m=1}^{\infty})$ as the computation strategy.

Definition 1 (Timely Computation Throughput)**.** Given the computation deadline d, using computation strategy η, the timely computation throughput, denoted by $R(d, \eta)$, is defined as follows:

$$R(d,\eta) = \lim_{M\to\infty} \frac{\sum_{m=1}^{M} N_m(d)}{M}. \quad (2)$$

Based on the above definitions, our problem is now formulated as the following.

Problem. *Consider a distributed computing system consisting of computation and network models as defined in Subsections II-A and II-B. Our goal is to find an optimal computation strategy achieving optimal timely computation throughput, denoted by $R^*(d)$ which is defined as follows:*

$$R^*(d) = \sup_{\eta} R(d,\eta) \quad (3)$$

As the main contributions of this paper, the problem is completely solved by a computation strategy proposed in Section III. We provide the proof of optimality of the proposed computation strategy in Section IV.

III. Estimate-and-Allocate (EA) Strategy

In this section, we propose a dynamic computation strategy called *Estimate-and-Allocate (EA)* strategy, which allocates loads to the workers adaptively by observing the history of computation times. In each round, the EA strategy first assigns computation loads by maximizing the estimated success probability based on the estimated transition probabilities of the underlying Markov chain (and based on that the previous state of the workers). After receiving the results, the EA strategy updates the estimated transition probabilities by observing the computation times in the past events.

Before introducing EA, we first define the following terms. For each worker i, we denote $C_{g\to g,i}(m)$ as the number of times that event "good state to good state" happened up to round m. Similarly, $C_{g\to b,i}(m)$, $C_{b\to g,i}(m)$ and $C_{b\to b,i}(m)$ are for events "good state to bad state", "bad state to good state" and "bad state to bad state" respectively. For worker i, we denote $\hat{p}_{g\to g,i}(m)$ and $\hat{p}_{b\to b,i}(m)$ as the estimated transition probabilities after the first $m-1$ rounds of computations. Also, we denote $\hat{p}_{g,i}(m)$ and $\hat{p}_{b,i}(m)$ as the estimated probabilities being in the good state and the bad state in round m respectively. Without loss of generality, we assume that $\hat{p}_{g,1}(m) \geq \hat{p}_{g,2}(m) \geq \cdots \geq \hat{p}_{g,n}(m)$. We also define

$$\ell_b \triangleq \mu_b d,\ \ell_g \triangleq \min(\mu_g d, \frac{w}{k}). \quad (4)$$

Note that we only consider the case: $w \geq n\mu_b d$, otherwise the computation can be always finished in time d which is trivial.

In each round m, EA has the following 4 phases:

(1) Load Assignment Phase: The master maximizes the estimated success probability in round m based on the the estimated probabilities $\hat{p}_{g,i}(m)$ and $\hat{p}_{b,i}(m)$. To do so, the master finds i_m^* $(1 \leq i_m^* \leq n)$ maximizing the estimated success probability function $\hat{\mathbb{P}}_m(\tilde{i})$ defined as follows:

$$\hat{\mathbb{P}}_m(\tilde{i}) = 0 \text{ if } w > \tilde{i}\ell_g + (n-\tilde{i})\ell_b, \quad (5)$$

otherwise

$$\hat{\mathbb{P}}_m(\tilde{i}) = \sum_{l=v(\tilde{i})}^{\tilde{i}} \sum_{\mathcal{G}:\mathcal{G}\subseteq[\tilde{i}],|\mathcal{G}|=l} \prod_{i\in\mathcal{G}} \hat{p}_{g,i}(m) \prod_{i\in[\tilde{i}]\setminus\mathcal{G}} \hat{p}_{b,i}(m) \quad (6)$$

where $v(\tilde{i}) \triangleq \tilde{i} + k - \lfloor \frac{k}{w}\{\tilde{i}\ell_g + (n-\tilde{i})\ell_b\} \rfloor$ is the minimum number of workers which compute ℓ_g rows have to be in the good state such that the computation is finished by deadline d. Thus, $i_m^* = \arg\max \hat{\mathbb{P}}_m(\tilde{i})$. The master does assignment by using the load allocation vector $\vec{\ell}_m$ defined as:

$$\ell_{m,i} = \begin{cases} \ell_g, & \text{if } 1 \leq i \leq i_m^* \\ \ell_b, & \text{otherwise;} \end{cases} \quad (7)$$

and the starting point vector $\vec{r}_m$ defined as:

$$r_{m,i} = \begin{cases} 0 & \text{if } i = 1 \\ r_{m,i-1} + \ell_{m,i-1} \pmod{\frac{w}{k}} & \text{if } 1 < i \leq n. \end{cases} \quad (8)$$

In load assignment phase, the idea is to select workers in the order of the estimated probability being in the good state, and assign more loads accordingly. Also, it is just a linear search in load assignment phase which is computationally efficient.

(2) Local Computation Phase: Within each round m, each worker i receives a matrix Y_m and the computation strategy $(r_{m,i}, \ell_{m,i})$ from the master. Then, each worker i computes matrix multiplication of sub-matrix $[\tilde{X}_i]_{[r_{m,i}:r_{m,i}+\ell_{m,i}-1]_c}$ and Y_m. After the computation, each worker sends all the computation results back to the master upon its completion.

(3) Aggregation and Observation Phase: Having received results from the fastest $n - i_m^* + v(i_m^*)$ workers, the master recovers X_1Y_m, X_2Y_m, ..., X_kY_m. By observing whether the results are sent back or not, the master checks which one of events "good state to good state", "good state to bad state", "bad state to good state" and "bad state to bad state" has happened in round m for each worker i. Then, the master obtains $C_{g\to g,i}(m)$, $C_{g\to b,i}(m)$, $C_{b\to g,i}(m)$ and $C_{b\to b,i}(m)$. Note that the time that it takes for one worker's result to be completed and sent back to the master actually indicates the (previous) state of that worker, since the speeds are deterministic and the computation time in a good state is less than the computation time in a bad state.

(4) Update Phase: After aggregation and observation phase, the master updates the estimated transition probabilities $\hat{p}_{g\to g,i}(m+1)$ and $\hat{p}_{b\to b,i}(m+1)$ for the round $m+1$: $\hat{p}_{g\to g,i}(m+1) = \frac{C_{g\to g,i}(m)}{C_{g\to g,i}(m)+C_{g\to b,i}(m)}$ and $\hat{p}_{b\to b,i}(m+1) = \frac{C_{b\to b,i}(m)}{C_{b\to g,i}(m)+C_{b\to b,i}(m)}$. The master updates the estimated probabilities $\hat{p}_{g,i}(m+1)$ and $\hat{p}_{b,i}(m+1)$: If worker i was in good state in round m, $\hat{p}_{g,i}(m+1) = \hat{p}_{g\to g,i}(m+1)$, and $\hat{p}_{g,i}(m+1) = 1 - \hat{p}_{b\to b,i}(m+1)$ otherwise. Then, the computation goes to round $m+1$.

IV. Optimality of EA

The following theorem shows the optimality of EA.

Theorem 1. *The proposed Estimate-and-Allocate (EA) strategy is optimal, i.e.,*

$$R_{EA}(d) = R^*(d) \quad \textit{almost surely}, \tag{9}$$

where $R_{EA}(d)$ denotes the timely computation throughput using the EA strategy.

The proof of Theorem 1 consists of 2 steps. In Step 1, we give an upper bound for the timely computation throughput by maximizing the success probability of each round when the Markov model is known to the master. Then in Step 2, we first show that the success probability using EA converges to the optimal success probability. By the Strong Law of Large Numbers (SLLN), the Ergodic theorem and a coupling argument, we prove that timely computation throughput achieved by the EA strategy matches with the derived upper bound.

For Step 1, we consider one round of computation using the computation strategy $(\vec{r}, \vec{\ell})$. Without knowing $\vec{\mu}$, we denote $T^{(\vec{r},\vec{\ell})}(\vec{\mu})$ as the random variable of finish time using the computation strategy $(\vec{r}, \vec{\ell})$. We define the success probability as the probability that the computation is finished in time d, i.e., $\mathbb{P}(T^{(\vec{r},\vec{\ell})} \le d)$ according to the distribution of $\vec{\mu}$.

Given an arbitrary load allocation vector $\vec{\ell}$, we now propose a starting point vector $\vec{r^*}(\vec{\ell})$ which is defined as follows:

$$r_i^* = \begin{cases} 0 & \text{if } i = 1 \\ r_{i-1}^* + \ell_{i-1} \pmod{\frac{w}{k}} & \text{if } 1 < i \le n. \end{cases} \tag{10}$$

To maximize the success probability by using the computation strategy $(\vec{r^*}(\vec{\ell}), \vec{\ell})$, we now introduce an optimization problem called *Load Allocation Problem* defined as follows:

Load Allocation Problem:

$$\text{Maximize } \mathbb{P}(T^{(\vec{r^*}(\vec{\ell}),\vec{\ell})} \le d) \tag{11}$$

$$\text{subject to } 0 \le \ell_i \le \frac{w}{k},\ \ell_i \in \mathbb{Z}, \forall 1 \le i \le n. \tag{12}$$

Note that the load allocation problem can be solved efficiently by a linear search instead of doing combinatorial search over all possible allocations [20].

The following lemma shows that the computation strategy composed of the proposed starting point vector and the load allocation vector that is the solution of load allocation problem achieves the optimal timely computation throughput.

Lemma 1. *Assume the Markov model of the network is know to the master. Let $\eta^* = \{(\vec{r^*}(\vec{\ell^*}_m), \vec{\ell^*}_m)\}_{m=1}^{\infty}$ be the computation strategy where $\vec{r^*}(\vec{\ell})$ is defined in (10) and $\{\vec{\ell^*}_m\}_{m=1}^{\infty}$ is given by solving load allocation problem. Then, η^* achieves the optimal timely computation throughput.*

We prove Lemma 1 in [20]. Since the Markov model is unknown to the master in the original problem, the timely computation throughput achieved by η^* gives us an upper bound. For Step 2, we first state the following lemma which is proved in [20].

Lemma 2. *$\mathbb{P}_{EA}(m)$ converges to $\mathbb{P}^*(m)$ as m goes to infinity, where $\mathbb{P}^*(m)$ denotes the optimal success probability in round m and $\mathbb{P}_{EA}(m)$ denotes the success probability in round m using the EA strategy.*

We denote $N_m^*(d)$ as the indicator representing whether the computation is finished by time d in round m using η^*. Then, $N_m^*(d)$ is a Bernoulli process with parameter $\mathbb{P}^*(m)$. We denote $N_{\text{EA},m}(d)$ as the indicator representing whether the computation is finished by time d in round m using EA. Then, $N_{\text{EA},m}(d)$ is a Bernoulli process with parameter $\mathbb{P}_{\text{EA}}(m)$.

Now, we model the state of the whole system which includes all n workers as a Markov chain. Since each worker has 2 states (good or bad), there are a total of 2^n different states of the system. Without loss of generality, we index the states of the system as $\{1, 2, \ldots, 2^n\}$. Since the transition matrix of this Markov chain has all the entries larger than 0, this Markov chain is irreducible. We denote $s(m)$ as the state of the system in round m. Also, p_s^* is denoted as the success probability of state s using the optimal computation strategy, i.e., $\mathbb{P}^*(m) = p_s^*$ if $s(m) = s$. By SLLN and the Ergodic theorem, we have

$$R^*(d) = \lim_{M\to\infty} \frac{\sum_{m=1}^{M} N_m^*(d)}{M} \tag{13}$$

$$= \lim_{M\to\infty} \sum_{s=1}^{2^n} \frac{\sum_{m\ge 1: s(m)=s} N_m^*(d)}{V_s(M)} \frac{V_s(M)}{M} = \sum_{s=1}^{2^n} p_s^* \frac{1}{\mathbb{E}_s[T_s]} \ a.s.,$$

where $V_s(m)$ is the number of visits to state s up to round m and $\mathbb{E}_s[T_s]$ is the expected return time to state s.

By Lemma 2, for all $\epsilon > 0$, there exits $m(\epsilon)$ such that $\mathbb{P}_{\text{EA}}(m) > \mathbb{P}^*(m) - \epsilon$ for all $m > m(\epsilon)$. Let $\tilde{N}_m(d)$ be the independent Bernoulli process with parameter $\mathbb{P}^*(m) - \epsilon$. We couple $N_{\text{EA},m}(d)$ and $\tilde{N}_m(d)$ as follows. If $N_{\text{EA},m}(d) = 0$,

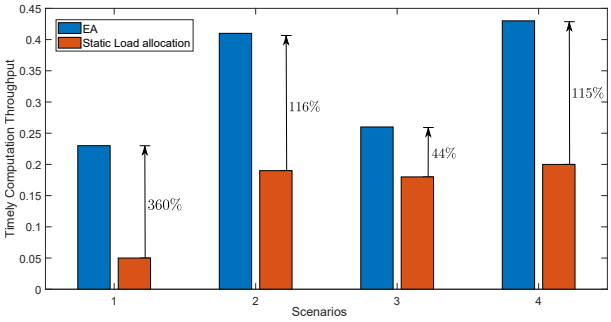

Fig. 3: Experimental evaluations over 15 `t2.micro` instances in Amazon EC2. Compared with the static load allocation strategy, EA improves the timely computation throughput by $1.44\times \sim 4.6\times$.

then $\tilde{N}_m(d) = 0$. If $N_{\text{EA},m}(d) = 1$, then $\tilde{N}_m(d) = 1$ with probability $\frac{\mathbb{P}^*(m)-\epsilon}{\mathbb{P}_{\text{EA}}(m)}$, and $\tilde{N}_m(d) = 0$ with probability $1 - \frac{\mathbb{P}^*(m)-\epsilon}{\mathbb{P}_{\text{EA}}(m)}$. Note that $\tilde{N}_m(d)$ is still marginally independent Bernoulli process of parameter $\mathbb{P}^*(m) - \epsilon$. Then, we have

$$R_{\text{EA}}(d) \geq R^*(d) - \sum_{s=1}^{2^n} \epsilon \frac{1}{\mathbb{E}_s[T_s]} \quad a.s. \tag{14}$$

using SLLN and the Ergodic theorem. The complete derivation of (14) can be found in [20]. By the fact that $R_{\text{EA}}(d) \leq R^*(d)$ and letting $\epsilon \to 0$, we have $R_{\text{EA}}(d) = R^*(d)$.

V. Experiments

We now demonstrate the impact of EA by experimental evaluations over Amazon EC2 clusters. We consider a sequence of matrix multiplications with a data matrix $X \in \mathbb{R}^{3000\times 3000}$ and an input matrix $Y_m \in \mathbb{R}^{3000\times 3000}$ in each round m with deadline d. Furthermore, the inter-arrival times between each round is set to a shifted-exponential distribution, with a shift of $T_c = 30$ seconds and parameter λ.

We implemented EA over an Amazon EC2 cluster with 16 nodes: one master node that is a `m4.xlarge` instance and 15 workers that are `t2.micro` instances. We compare EA with a static load allocation strategy, in which each worker is assigned to ℓ_g or ℓ_b number of vector-matrix multiplications with equal probability in each round, where ℓ_g and ℓ_b are defined in (4). We estimate the speed of workers in the good and bad states (i.e., μ_g and μ_b) by measuring the average runtime of a vector-matrix multiplication for both two states.

We implemented EA and the static load allocation strategy in python, and used MPI4py [21] for message passing between the instances. Before starting the computations, each worker stores an encoded data matrix in its local memory. In each round, having received a matrix from the master, each worker computes the assigned computation and sends it back to the master asynchronously using `Isend()`. As soon as the master gathers enough results from the workers, it recovers the computations. For experiments, we consider the following four parameter settings.

Scenario 1: $(15, 12)$ MDS code, $\lambda = 10$ and $d = 2.5$.
Scenario 2: $(15, 12)$ MDS code, $\lambda = 30$ and $d = 2.5$.
Scenario 3: $(15, 10)$ MDS code, $\lambda = 10$ and $d = 3$.
Scenario 4: $(15, 10)$ MDS code, $\lambda = 30$ and $d = 3$.

Fig. 3 provides a performance comparison of EA with the static load allocation strategy. Over the four scenarios, EA increases the timely computation throughput by $1.44\times \sim 4.6\times$.

VI. Acknowledgment

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053, ARO award W911NF1810400, NSF grants CCF-1703575, CCF-1763673, NeTS-1419632, ONR Award No. N00014-16-1-2189, and the UC Office of President under grant No. LFR-18-548175. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones.," in *NSDI*, 2013.
[2] I. . Hou, V. Borkar, and P. R. Kumar, "A theory of qos for wireless," in *IEEE INFOCOM 2009*, pp. 486–494, April 2009.
[3] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely-throughput optimal coded computing over cloud networks," in *ACM MobiHoc*, 2019.
[4] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.
[5] 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*, 2018.
[6] 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*, pp. 2100–2108, 2016.
[7] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *2017 IEEE ISIT*, pp. 2418–2422, IEEE, 2017.
[8] 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.
[9] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *International Conference on Machine Learning*, pp. 3368–3376, 2017.
[10] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded computation over heterogeneous clusters," in *Information Theory (ISIT), 2017 IEEE International Symposium on*, pp. 2408–2412, IEEE, 2017.
[11] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *2018 IEEE ISIT*, pp. 1620–1624, IEEE, 2018.
[12] Q. Yu, S. Li, N. Raviv, S. M. Mousavi, M. Soltanolkotabi, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," in *Artificial Intelligence and Statistics*, 2019.
[13] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Advances in Neural Information Processing Systems*, pp. 5434–5442, 2017.
[14] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE transactions on automatic control*, vol. 37, no. 12, pp. 1936–1948, 1992.
[15] J. G. Dai and W. Lin, "Maximum pressure policies in stochastic processing networks," *Operations Research*, vol. 53, no. 2, 2005.
[16] R. Pedarsani, J. Walrand, and Y. Zhong, "Robust scheduling for flexible processing networks," *Advances in Applied Probability*, vol. 49, 2017.
[17] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *INFOCOM, 2012 Proceedings IEEE*, pp. 702–710, IEEE, 2012.
[18] C.-S. Yang, A. S. Avestimehr, and R. Pedarsani, "Communication-aware scheduling of serial tasks for dispersed computing," in *ISIT*, IEEE, 2018.
[19] M. Hoseinnejhad and N. J. Navimipour, "Deadline constrained task scheduling in the cloud computing using a discrete firefly algorithm," *INTERNATIONAL JOURNAL OF NEXT-GENERATION COMPUTING*, vol. 8, no. 3, 2017.
[20] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely coded computing," http://www-bcf.usc.edu/~avestime/papers/EA_ISIT2019.pdf.
[21] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, "Parallel distributed computing using python," *Advances in Water Resources*, 2011.