Analyzing irregular Tree Gradient Coding

BHARGAV KULKARNI

Abstract

Distributing the task of gradient computation among multiple machines helps scale up training of machine learning models. Synchronous algorithms based on master-worker topology have been proposed for the same. But such systems suffer from straggling, where the overall completion time is determined by the slowest worker. Gradient Coding was proposed to mitigate this issue. The technique allows the master to recover complete gradient from responses of only some and not all workers. It is able to achieve this by incorporating coding theoretic techniques for data distribution among workers and for aggregating gradient responses from the faster workers. Although the issue of stragglers is resolved by Gradient Coding, it suffers from bandwidth contention in reception of simultaneous responses from workers. Tree Gradient Coding parallelizes communications by arranging workers in a regular tree.

It is not always possible to distribute a set of nodes into a regular tree over multiple levels. The realm of irregular tree topology for distributed gradient computation is unexplored, where a greater number of nodes can be incorporated within the topology. In this work we provide a theoretical analysis for completion time of one epoch in Tree Gradient Coding for irregular topology over two levels. We derive the computation load that is obtained on each node. We obtain asymptotic bounds on the expected completion time to measure the impact of different parameters. We also provide exact and approximation analysis for different cases for a more detailed study. Implementation of Tree Gradient Coding is performed on Amazon EC2 nodes to motivate the analysis. The analysis is verified by theoretical simulations.

Contents

Chapter 1

Introduction

Machine learning is being used in a wide range of applications: from prediction tasks to natural language generation to recommendation systems, among many others. To improve the accuracy of the ML models used, they are usually trained on huge datasets. Some of these datasets serve as benchmarks to compare ML models. For instance the ImageNet [**5**] dataset contains over 15 million images with labellings intended for image classication tasks. Various models for recognizing hand-written digits have been trained on the MNIST dataset, which consists of 60,000 images. Training models on such datasets can take a long period of time. Training a CNN-based model on the MNIST dataset consumes over 40 minutes for 30 epochs [**6**]. Processing such volumes of data also involves excessive I/O delays [**7**].
It becomes desirable to scale out the machine learning tasks to multiple systems. Various solutions have been proposed to perform machine learning at scale. Distributing ML algorithms can be achieved in broadly two ways: data-parallelism and model-parallelism [**7**]. In data-parallelism approach the dataset is split and assigned to worker nodes. The worker nodes independently train on the local data-partition and send the updates to a master node. The master aggregates the responses to update the model. Model-parallel algorithms on the other hand partitions the model parameters across workers, while operating on the entire dataset. Many ML algorithms are known to exhibit data-parallel nature as opposed to model-parallelism.

Gradient descent is a commonly used optimization technique to train machine learning models. It iteratively updates model parameters in opposite direction of their gradient

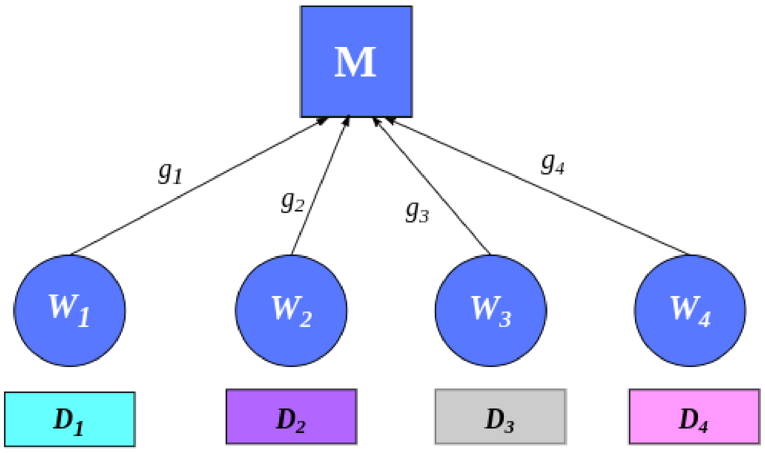

Figure 1.1: Synchronous Gradient Descent

by small steps. Gradient of a parameter is often a function of existing parameters and the data points. Distributing the computation of gradient becomes essential in scaling up the training phase of ML tasks. Gradient computation is data-parallel in nature, allowing it to be distributed among multiple nodes. Synchronous stochastic gradient descent was proposed in [**8**], where the dataset is equally partitioned among worker nodes. Master/parameter server initializes the model parameters and communicates it to all workers. Each worker independently computes gradient on the local data partition and communicates the partial gradient to the master. Master aggregates the partial gradients to arrive at (approximate) total gradient.

While the approach scales linearly with the number of workers, it suffers from shortcomings prevalent in master-worker topology:

1. **Straggler effect**: Workers that compute relatively slowl are termed stragglers. As the master waits for responses from all workers, the slowest worker dictates the overall computation time.

2. **Communication bottleneck**: If multiple workers simultaneously communicate to master, a response has to wait for previously queued responses to be processed. Congestion in responses scales up with worker nodes. This is also found to be the case with Amazon EC2 instances, as observed from Fig.1.2.

To address these issues, mainly that of stragglers, Gradient Coding technique was proposed.

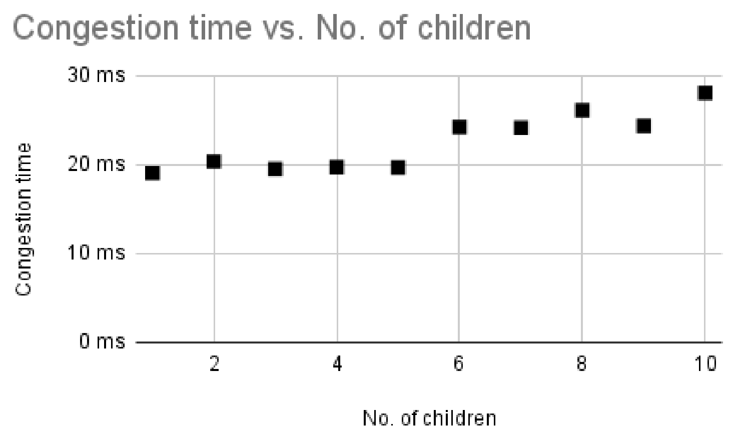

Figure 1.2: Congestion increases with number of children nodes

Gradient Coding [**2**] enables the master to reconstruct the total gradient by only requiring partial gradient responses from non-straggling workers. To achieve this, coding theoretic methods are used in distributing data among workers and in the linear combination of the partial gradients to retrieve the complete gradient. This benefit comes at the cost of redundant distribution of data partitions among workers, resulting in increased computation at each worker.

Gradient Coding technique does not adequately address the issue of congestion of responses at the master. To parallelize the communications involved, Tree Gradient Coding [**3**] was proposed. Tree Gradient Coding arranges the worker nodes in a regular tree topology. Data distribution among children and gradient reconstruction at each (non-leaf) node is similar to that of Gradient Coding. Each node upon receiving data partitions from its master assigns a fraction of data to itself. Rest of the data is recursively distributed among its children. This technique was found to perform better than GC in having a lower epoch time.

1.1 Problem gap and Contributions

In a distributed setting, the completion time of the gradient descent algorithm is a crucial factor that directly impacts the performance of the system. Completion time is the duration taken to complete one epoch. A shorter completion time can lead to faster convergence and better accuracy of the model. Therefore, researchers have focused on developing efficient strategies to reduce the completion time of distributed gradient descent, such as parallelizing the computation, optimizing communication overhead, and minimizing synchronization delay. There is a dearth of theoretical analysis on completion time, which can be used to provide guarantees of efficacy of one topology over another. The realm of irregular tree topology for gradient computation is unexplored by TGC, where a greater number of nodes can be incorporated within the topology.

Motivated by these reasons, we aim to provide theoretical analysis over completion time of TGC on irregular tree toplogy consisting of two levels. We make the following contributions in this work:

1. Derive computation load of TGC on each node for irregular topology
2. Obtain asymptotic bounds on completion time and verify the expressions obtained with simulations
3. Provide an exact analysis of TGC for irregular topology. Suitable approximations for completion times are also obtained and the expressions are verified against simulations.

Chapter 2

Background

2.1 Gradient Descent

Most of the machine learning algorithms try to optimize the value of its loss function using the well known gradient descent algorithm. For a training dataset $D = \{(x_j, y_j) \in \mathbb{R}^p \times \mathbb{R}\}$, where p is the number or parameters, the goal is to minimize the loss function:

$$\sum_{x \in D} \ell(\theta; x) + \lambda R(\theta),$$

where $\ell(\cdot)$ is the underlying loss function, $R(\cdot)$ denotes the regularization function, θ denotes parametric weights, and λ denotes the regularization parameter. The gradient descent algorithm tries to estimate θ to minimize the total loss value. Gradient descent is a popular optimization algorithm used to minimize the cost function in various machine learning and deep learning models. Gradient descent algorithm works by iteratively adjusting the parameters of the model in the direction of the negative gradient of the cost function.

The gradient descent algorithm starts with random initialization of the model parameters and then repeatedly updates them until the convergence criteria are met. The convergence is achieved when the gradient of the cost function is close to zero, which indicates that the minimum of the function has been reached. The learning rate is a crucial hyperparameter that determines the size of the step taken in each iteration. A small learning rate results in a slow convergence, whereas a large learning rate can cause the algorithm to diverge. Gradient descent has several variations, such as batch, stochastic, and mini-batch, depending on the number of samples used to compute the gradients in

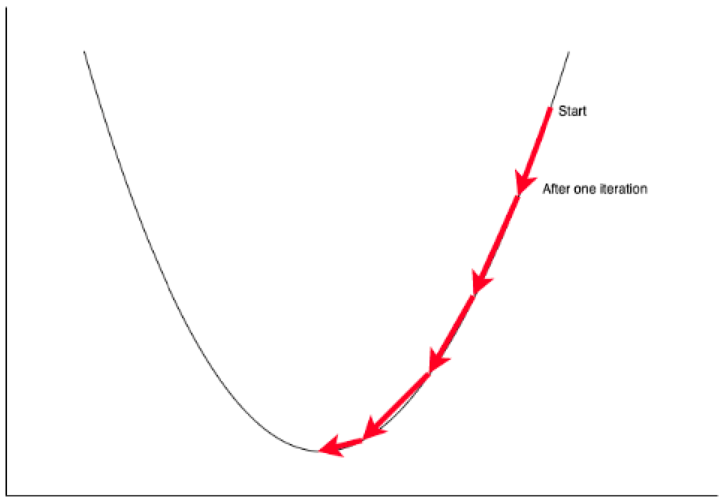

Figure 2.1: Gradient descent

each iteration. Despite its simplicity, gradient descent is an essential optimization algorithm that underpins many machine learning and deep learning models and continues to be an active area of research.

In gradient descent algorithm, the gradient of parameters $g = \sum_{x \in D} \nabla \ell(\theta^{(t)}; x)$ is calculated at each epoch. The parameters are updated at each epoch t as $\theta^{(t+1)} = \theta^{(t)} - \mu(g + \lambda \nabla R(\theta^{(t)}))$, where μ is the learning rate.

In distributed gradient descent, we consider one master and N worker nodes $W_1, W_2, ...W_N$. The training dataset needs to be distributed over N workers. These workers will communicate with the master in each epoch, and the master will aggregate the gradients received from the workers to obtain the full gradient. For resiliency to stragglers, computation redundancy needs to be introduced. In the remaining part of this section, Gradient Coding [**2**] and Tree Gradient Coding [**3**] are briefly discussed.

2.2 Gradient Coding

Gradient coding introduces redundancy in computation to provide resilience to s stragglers. Consider an instance of gradient coding with $N = 4$ worker nodes and robust to $s = 2$ stragglers, which is depicted in Fig 2.2. The training dataset D is divided into $k = 4$ partitions D_1, D_2, D_3, D_4. The gradient corresponding to a dataset $\mathcal{D}$ is denoted by $g_{\mathcal{D}}$. The vector $\bar{g} = [g_{D_1}, g_{D_2}, g_{D_3}, g_{D_4}]^T$ denotes a vector of partial gradients. To achieve

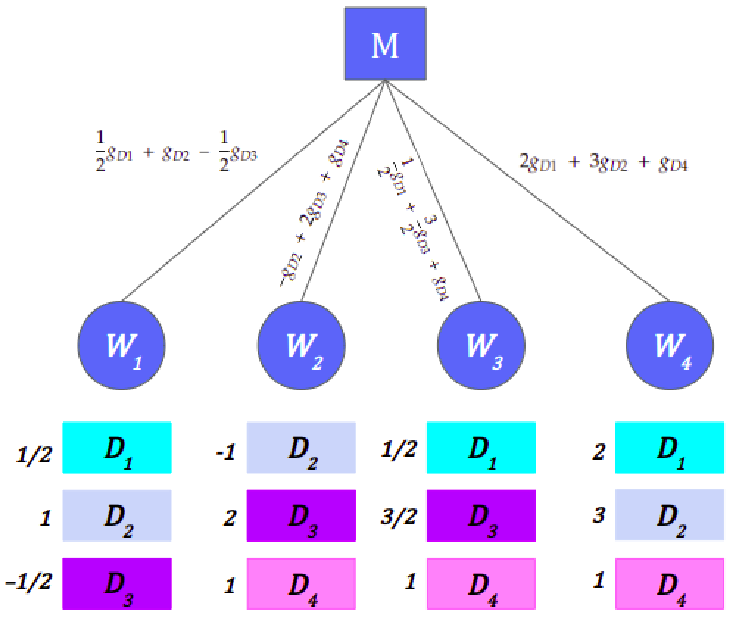
M
$\frac{1}{2}g_{D1} + g_{D2} - \frac{1}{2}g_{D3}$
$-g_{D2} + 2g_{D3} + g_{D4}$
$\frac{1}{2}g_{D1} + \frac{3}{2}g_{D3} + g_{D4}$
$2g_{D1} + 3g_{D2} + g_{D4}$
W_1
W_2
W_3
W_4
1/2 D_1
1 D_2
-1/2 D_3
-1 D_2
2 D_3
1 D_4
1/2 D_1
3/2 D_3
1 D_4
2 D_1
3 D_2
1 D_4

Figure 2.2: Gradient Coding

resiliency to $s = 2$ stragglers, we assign $s + 1 = 3$ partitions to each worker. Each worker computes the gradient on 3 partitions, and sends a linear combination of them to the master. The master would be able to recover full gradient after receiving partial gradients from $N - s = 2$ workers. The linear combination of the gradients is governed by an encoding matrix $B \in \mathbb{R}^{N \times k}$, and recovery of the full gradients from the received partial gradients is guided by a decoding matrix $A \in \mathbb{R}^{\binom{N}{s} \times N}$. The i^{th} row of B matrix denotes the coefficients for linear combination of the gradient computed on the partitions assigned to worker i. The A matrix has a row for each straggling scenarios, and could be used for recovering the full gradient. The A and B matrix are chosen such that $AB = 1_{\binom{N}{s} \times k}$. Methods to derive A and B matrix for any N and s $(s < N)$ are highlighted in [**2**].

The computation load r of a worker is the fraction of the dataset allotted to it for gradient computation. To be robust against s stragglers, a computation load $r \geq \frac{s+1}{N}$ is necessary. The lower bound is achieved by the gradient coding scheme $r_{GC} = \frac{s+1}{N}$.

In the example depicted in the Fig. **??**, the computation load is $r_{GC} = \frac{3}{4}$. After computing gradients on the partitions alloted to it, the worker W_i sends $m_i = B_i\bar{g}$ to the master. After receiving gradients from $N - s$ workers, the master uses the A matrix to recover the full gradient. If in this setup, W_2 and W_3 are the stragglers, then master can recover the full gradient g by a linear combination of m_1 and m_4, as $g = -2m_1 + m_4$. Similarly, if W_3 and W_4 are stragglers, the master could recover $g = 2m_1 + m_2$ from m_1

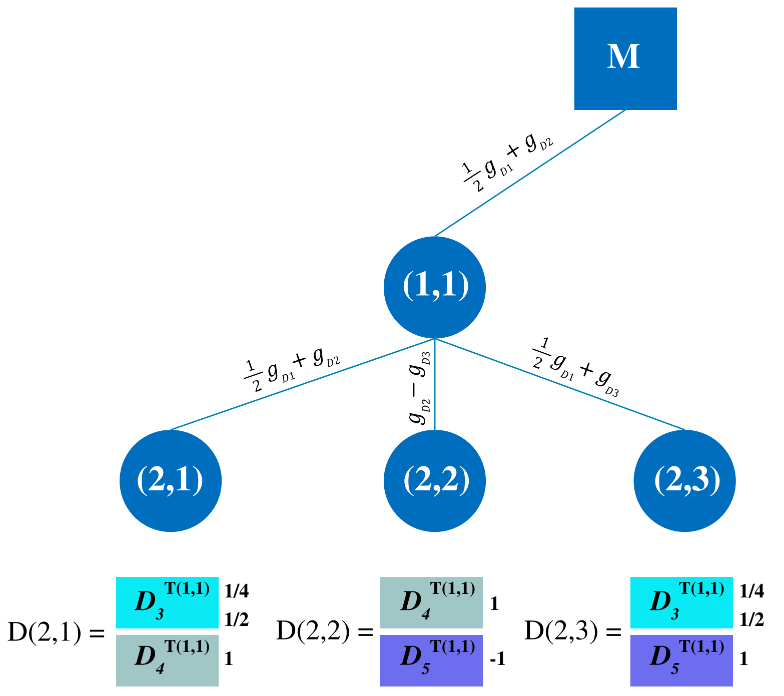

Figure 2.3: Tree Gradient Coding

and m_2.

As noted in [**2**], the system is bottlenecked by multiple workers trying to talk to the master concurrently. The bandwidth contention at the master significantly deteriorates the performance as the number of workers increases. TGC circumvents this issue by introducing a tree topology where communications are parallelized.

2.3 Tree Gradient Coding

We consider an (n, L) regular tree topology of master and workers as shown in the Fig. **??**, where there are L levels, and except the nodes in the last level, each node has n children nodes. So, the total number of worker nodes is $N = n + n^2 + ... + n^L$. Here, each node only communicates with its parent and children nodes, and hence the issue of bandwidth contention at master is alleviated. We use M to denote the master node, and (ℓ, j) to denote the worker node with index $j \in [n^\ell]$ in level $\ell \in [L]$. We represent the sub-tree rooted at node (ℓ, j) by $T(\ell, j)$. The coded gradient sent by a node (ℓ, j) to its parent is denoted by $m_{(\ell,j)}$.

As depicted in the Fig. **??**, the master M redundantly assigns $s + 1$ data partitions of the training dataset D to each of its n sub-trees following the gradient coding scheme with

n workers and s stragglers. The dataset has $|D|= d$ datapoints. The data distributed within the sub-tree $T(\ell, j)$ is denoted by $D^{T(\ell,j)}$. The node (ℓ, j) picks $|D(\ell, j)|= r_{TGC}d$ datapoints as its local dataset, and distributes the remaining $D_{T(\ell,j)} = D^{T(\ell,j)} \setminus D(\ell, j)$ datapoints redundantly to its sub-trees following the gradient coding scheme. Note that redundancy is introduced at each level to acheive straggler resiliency. In [**3**], it is shown that TGC achieves the optimal computation load for an (n, L) regular tree topology, which is

$$r_{TGC} = \frac{1}{\sum_{j=1}^{L} \left(\frac{n}{s+1}\right)^j}. \tag{2.1}$$

Once the leaf nodes compute their gradients, they send their coded gradients $m_{(L,j)}$ $(j \in [n^L])$ computed on their local datasets $D(L, j)$ to their parents. Their parents $(L - 1, i)$ recover $g_{D_{T(L-1,i)}}$ $(i \in [n^{(L-1)}])$ from the coded gradients received from $n - s$ children, and accumulate it with the gradient computed on their local dataset $D(L-1, i)$ to obtain $g_{D^{T(L-1,i)}}$. This process is carried out across the levels, and the master will eventually recover the full gradient g.

2.3.1 Other literature on Gradient Coding

Since the introduction of Gradient Coding, other research works have come up that have attempted to improve on different aspects of the technique. Partial recovery of gradient using GC was proposed in [**9**], where the requirement for the master was relaxed to recover only a certain fraction α of the total gradient. Modified data distribution and partial gradient aggregation based on combinatorial methods were employed to achieve this. On a similar line, work in [**10**] approximately computed gradient by using sparse random graphs. To handle for worker nodes with different computational capabilities, an adaptive load distribution scheme for GC was proposed in [**11**]. This scheme results in different computation load for different nodes in proportion to their speed of computation. Performance of TGC was improved by the work in [**12**], which allocated higher computation load to nodes at higher levels (nearer to master). This accounted for the idle time that nodes experience in TGC in upper levels due to time taken for results to be communicated through the levels.

Chapter 3

Work

3.1 Model

We consider a machine learning task which has p model parameters to be trained on a dataset $\mathcal{D}$ of size d. On a convex loss function $\mathcal{L}$, the gradient descent algorithm aims to arrive at a parameter set $\mathcal{P}^*$ which minimizes $\mathcal{L}$. Tree Gradient Coding is employed to distribute the computation of gradient. The topology of TGC is a complete tree consisting of two levels. The number of nodes in level 1 (which is the number of children of the master node) is denoted by n_1 and the number of children assigned to each node in level 1 is denoted by n_2. The total number of worker nodes N in the topology would be:

$$N = n_1 + n_1 \cdot n_2$$

The number of stragglers among nodes in level 1 is taken to be s_1 and that among siblings (nodes having a common parent) in level 2 is s_2. Straggler ratio refers to the fraction of

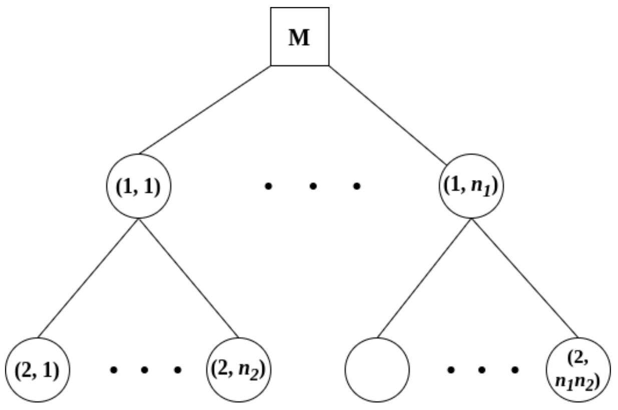

Figure 3.1: Topology for TGC

siblings (nodes having a common parent) that can be stragglers. It is denoted by α and we have $s_1 = n_1{\cdot}\alpha$ and $s_2 = n_2{\cdot}\alpha$. The variation in computation performance among nodes characterizes the straggler ratio. Setting the number of stragglers is a choice orthogonal to our analysis.

We assume the computational time modelling explained in [**4**], where the computation time of each worker is modelled as an i.i.d. random variable which is exponential in nature. The exponential r.v. consists of two parameters a and μ, which control the computation time on a dataset for each worker. If T_i denotes the random variable for the time taken by a worker i in computing gradient for dataset having d_i data points, the probability distribution of the computation time is given by:

$$\mathbb{P}[T_{\mathrm{i}} \leq t] = 1 - e^{\frac{\mu}{d_i}(t - a \cdot d_i)},\ t \geq a \cdot d_i$$

We assume t_c to be the time required in communicating the p partial gradients from a worker to its parent.

We begin by arriving at computation load for the irregular tree topology.

3.2 Computation load

The computational load r_{irr} on each worker turns out to be

$$r_{\mathrm{irr}} = \frac{1}{\dfrac{n_1}{1+s_1} + \dfrac{n_1{\cdot}n_2}{(1+s_1)\cdot(1+s_2)}} \tag{3.1}$$

We derive the more general expression for computational load for an irregular tree consisting of L levels, where number of children of node at level $i-1$ is n_i and number of stragglers among the siblings at level i is s_i:

$$r_{\mathrm{irr},\ L} = \frac{1}{\sum_{i=1}^{L}\prod_{j=1}^{i}\dfrac{n_j}{1+s_j}}$$

The data allocation and gradient computation scheme is the same as that in [**3**]. The master node initially holds all the data and divides it into n_1 equal partitions and each partition is duplicated $(s_1 + 1)$ times. Hence,

$$|\mathcal{D}^{T(1,1)}| = \left(\tfrac{1+s_1}{n_1}\right) d$$

As each worker assigns itself equal amounts of data, which is equal to computational load times the total data, we have

$$|\mathcal{D}(1,1)| = r_{\text{irr},\ L} \cdot d$$

The data that remains to be assigned to children of node $T(1,1)$ is the data left after it assigns the portion of data to itself. So

$$\mathcal{D}_{T(1,1)} = \mathcal{D}^{T(1,1)} / \mathcal{D}(1,1)$$

and

$$|\mathcal{D}_{T(1,1)}| = \left(\frac{1+s_1}{n_1}\right) d - r_{\text{irr},\ L} \cdot d \tag{3.2}$$

The data set $D_{T(1,1)}$ is again distributed to children of $T(1,1)$ in a manner similar to that in gradient coding. Hence,

$$|D^{T(2,1)}| = \left(\frac{1+s_2}{n_2}\right) |D_{T(1,1)}| \tag{3.3}$$

After node $T(2,1)$ assigns data to itself, we have:

$$\mathcal{D}_{T(2,1)} = \mathcal{D}^{T(2,1)} / \mathcal{D}(2,1)$$

and

$$|\mathcal{D}_{T(2,1)}| = |\mathcal{D}^{T(2,1)}| - |\mathcal{D}(2,1)| \tag{3.4}$$

Substituting equations (4) and (5) into (6), we have:

$$|\mathcal{D}_{T(2,1)}| = \left(\frac{1+s_1}{n_1}\right)\left(\frac{1+s_2}{n_2}\right) d - r_{\text{irr},\ L}\left(1 + \left(\frac{1+s_2}{n_2}\right)\right) d$$

Since

$$|\mathcal{D}^{T(2,1)}| = \left(\frac{1+s_2}{n_2}\right) |\mathcal{D}_{T(1,1)}| \tag{3.5}$$

we obtain:

$$|\mathcal{D}^{T(3,1)}| = \left(\frac{1+s_1}{n_1}\right)\left(\frac{1+s_2}{n_2}\right)\left(\frac{1+s_3}{n_3}\right) d - r_{\text{irr},\ L}\left(1 + \left(\frac{1+s_3}{n_3}\right) + \left(\frac{1+s_2}{n_2}\right)\left(\frac{1+s_3}{n_3}\right)\right) d$$

In a similar manner, we can continue computing incoming data volume for nodes at deeper levels and we arrive at the following expression for $|\mathcal{D}^{T(L,1)}|$:

$$\begin{aligned} |\mathcal{D}^{T(L,1)}| = &\left(\frac{1+s_1}{n_1}\right)\left(\frac{1+s_2}{n_2}\right)\cdots\left(\frac{1+s_L}{n_L}\right) d \\ &- r_{\text{irr},\ L}\left(1 + \left(\frac{1+s_L}{n_L}\right) + \left(\frac{1+s_{L-1}}{n_{L-1}}\right)\left(\frac{1+s_L}{n_L}\right) + \ldots + \left(\frac{1+s_2}{n_2}\right)\cdots\left(\frac{1+s_L}{n_L}\right)\right) d \end{aligned} \tag{3.6}$$

But $|\mathcal{D}^{T(L,1)}| = r_{\text{irr},L} \cdot d$, as $T(L,1)$ is a leaf node. Incorporating this into the above equation, we obtain the following for $r_{\text{irr},L}$:

$$r_{\text{irr},\ L} = \frac{\left(\frac{1+s_1}{n_1}\right)\left(\frac{1+s_2}{n_2}\right)\cdots\left(\frac{1+s_L}{n_L}\right)}{1 + \left(\frac{1+s_L}{n_L}\right) + \left(\frac{1+s_{L-1}}{n_{L-1}}\right)\left(\frac{1+s_L}{n_L}\right) + \ldots + \left(\frac{1+s_2}{n_2}\right)\cdots\left(\frac{1+s_L}{n_L}\right)} \tag{3.7}$$

Multiplying both numerator and denominator of the above expression by $\left(\frac{1+s_1}{n_1}\right)\left(\frac{1+s_2}{n_2}\right)\dots\left(\frac{1+s_L}{n_L}\right)$, we obtain the final expression for $r_{\text{irr},L}$:

$$r_{\text{irr},\,L} = \frac{1}{\sum_{i=1}^{L}\prod_{j=1}^{i}\frac{n_j}{1+s_j}}$$

As we require computational load for irregular tree consisting of two levels, we substitute L by 2 to obtain:

$$r_{\text{irr}} = \frac{1}{\frac{n_1}{1+s_1} + \frac{n_1 \cdot n_2}{(1+s_1)\cdot(1+s_2)}}$$

Having derived the expression for computation load, we now turn our attention to characterizing the completion time for gradient computation. Specifically, completion time refers to the period starting from workers computing partial gradient to the time when the master has aggregated partial gradients to arrive at the complete gradient. We now asymptomatically chracterize the completion time and then move on to a more accurate estimation.

3.3 Asymptotic analysis

The expected value of the r.v. for completion time T_C can be asymptotically bound as follows:

$$\begin{aligned}\mathbb{E}[T_{\text{C}}] \geq \frac{r_{\text{irr}}d}{\mu}\log\left(\frac{1}{\alpha}\right) + ar_{\text{irr}}d + \\ (\max(n_1,n_2)(1-\alpha) - o(\max(n_1,n_2)) + 1)t_{\text{c}} + o(1)\end{aligned} \tag{3.8}$$

$$\mathbb{E}[T_{\text{C}}] \leq \frac{r_{\text{irr}}d}{\mu}\log\left(\frac{1}{\alpha}\right) + ar_{\text{irr}}d + (n_1+n_2)(1-\alpha)t_{\text{c}} + o(1) \tag{3.9}$$

We first derive the expression for lower bound

$$\begin{aligned}\mathbb{E}[T_{\text{C}}] \geq \frac{r_{\text{irr}}d}{\mu}\log\left(\frac{1}{\alpha}\right) + ar_{\text{irr}}d + \\ (\max(n_1,n_2)(1-\alpha) - o(\max(n_1,n_2)) + 1)t_{\text{c}} + o(1)\end{aligned}$$

Consider all the groups of siblings residing in layer 2. Since the gradient computation scheme is robust to at most s_2 stragglers, the parent of a sibling group only expects partial gradient result from any $n_2 - s_2$ nodes. Let T_{L_2} denote the fastest time in which a parent of a group of siblings is able to recover partial gradients from the fastest $n_2 - s_2$ siblings. Let $T_{(n_2 - s_2)}$ denote the $(n_2 - s_2)^{\text{th}}$ order of statistic of computation times of sibling nodes $T_1, T_2, \ldots, T_{n_2}$. Assuming $t_c > \mathbb{E}[T_i]$, with a probability of $1 - o(1)$, at most $o(n_2)$ transmissions are completed by the time $T_{(n_2 - s_2)}$. Hence, at least $n_2 - s_2 - o(n_2)$ transmissions remain after after $T_{(n_2 - s_2)}$. As a result, we have the following, with probability $1 - o(1)$:

$$T_{L_2} \geq T_{(n_2\,-\,s_2)} + (n_2 - s_2 - o(n_2))t_{\text{c}}$$

Incorporating the probabilistic event, we compute the expected time:

$$\mathbb{E}[T_{\text{C}}] \geq (\mathbb{E}[T_{(n_2\,-\,s_2)}] + \\ (n_2 - s_2 - o(n_2))t_{\text{c}} + t_{\text{c}})(1 - o(1)) + \mathbb{E}[T_{(n_2\,-\,s_2)}]o(1)$$

$$\mathbb{E}[T_{\text{C}}] \geq \mathbb{E}[T_{(n_2\,-\,s_2)}] + (n_2 - s_2 - o(n_2) + 1)t_{\text{c}}(1 - o(1)) \tag{3.10}$$

We can make a similar analysis on T_{L_1}, where we ignore nodes at L_2 for calculating computation time and add communication time later.

$$\mathbb{E}[T_{\text{C}}] \geq \mathbb{E}[T_{(n_1\,-\,s_1)}] + (n_1 - s_1 - o(n_1) + 1)t_{\text{c}}(1 - o(1)) \tag{3.11}$$

Given the straggler ratio α, we have:
$\alpha = \dfrac{s_1}{n_1} = \dfrac{s_2}{n_2}$
For n i.i.d. exponential random variables with parameter λ, expected value of k^{th} order statistic can be approximated as:

$$\mathbb{E}[T_{(\text{k})}] = \frac{1}{\lambda}\log\left(\frac{n}{n-k}\right) + o(1)$$

Incorporating above inequalities into (10) and (11), we obtain

$$\mathbb{E}[T_{\text{C}}] \geq \frac{r_{\text{irr}}d}{\mu}\log\left(\frac{1}{\alpha}\right) + ar_{\text{irr}}d + (n_2(1-\alpha) - o(n_2) + 1)t_{\text{c}} + o(1) \tag{3.12}$$

$$\mathbb{E}[T_{\text{C}}] \geq \frac{r_{\text{irr}}d}{\mu}\log\left(\frac{1}{\alpha}\right) + ar_{\text{irr}}d + (n_1(1-\alpha) - o(n_1) + 1)t_{\text{c}} + o(1) \tag{3.13}$$

Obtaining a tighter bound from (12) and (13), we have

$$\mathbb{E}[T_{\text{C}}] \geq \frac{r_{\text{irr}}d}{\mu}\log\left(\frac{1}{\alpha}\right) + ar_{\text{irr}}d + (\max(n_1, n_2)(1-\alpha) - o(\max(n_1, n_2)) + 1)t_{\text{c}} + o(1)$$

We now derive expression for upper bound of the expected time

$$\mathbb{E}[T_{\text{C}}] \leq \frac{r_{\text{irr}}d}{\mu}\log\left(\frac{1}{\alpha}\right) + ar_{\text{irr}}d + (n_1 + n_2)(1-\alpha)t_{\text{c}} + o(1)$$

In assuming all communications take place after computation, we can upper bound the total time. Let T_{comp} denote the time by which sufficient number of nodes have computed their partial gradients from which it is possible to recover the total gradient descent.

$$T_{\mathrm{C}} \leq T_{\mathrm{comp}} + (n_2 - s_2 + n_1 - s_1)t_{\mathrm{c}}$$

So

$$\mathbb{E}[T_{\mathrm{C}}] \leq \mathbb{E}[T_{\mathrm{comp}}] + (n_1 + n_2)(1-\alpha)t_{\mathrm{c}} \tag{3.14}$$

To bound the expected time of computations, we introduce two events. Consider event ε_1:

$\varepsilon_1 := \{T^{gr}_{(n_i - s_i)} \leq \mathbb{E}[T^{gr}_{(n_i - s_i)}] + \epsilon, \forall \text{ sibling groups } gr\}$
where $\epsilon = \Theta(1/\min(n_1, n_2)^{\frac{1}{4}})$

Denote by $\hat{T}$ the time by which all non-straggling nodes of all sibling groups have finished computing their partial gradients. Let ε_2 be the event:
$\varepsilon_2 := \{\hat{T} > \Theta(\log(\min(n_1, n_2)))\}$

We have,

$$\mathbb{E}[T_{\mathrm{comp}}] \leq \mathbb{E}[T_{\mathrm{comp}}|\varepsilon_1] \ + \ \mathbb{E}[T_{\mathrm{comp}}|\varepsilon_1^c] \ \mathrm{P}[\varepsilon_1^c] \tag{3.15}$$

$$\mathbb{E}[T_{\mathrm{comp}}|\varepsilon_1] \leq \frac{r_{\mathrm{irr}} d}{\mu} \log\left(\frac{1}{\alpha}\right) + a r_{\mathrm{irr}} d + o(1) \tag{3.16}$$

Conditioning on event ε_1^c, we have

$$\mathbb{E}[T_{\mathrm{comp}}|{\varepsilon_1}^{\mathrm{c}}] \leq \mathbb{E}[T_{\mathrm{comp}}|{\varepsilon_1}^{\mathrm{c}} \cap {\varepsilon_2}^{\mathrm{c}}] + \mathbb{E}[T_{\mathrm{comp}}|{\varepsilon_1}^{\mathrm{c}} \cap \varepsilon_2]\mathbb{P}[\varepsilon_2] \tag{3.17}$$

Since $\mathbb{E}[T^{gr}_{(n_i - s_i)}] + \epsilon = \Theta(1)$,

$$\mathbb{E}[T_{\mathrm{comp}}|{\varepsilon_1}^{\mathrm{c}} \cap {\varepsilon_2}^{\mathrm{c}}] \leq \Theta(\log(\min(n_1, n_2))) \tag{3.18}$$

As $\hat{T} \geq T_{comp}$,

$$\begin{aligned} &\mathbb{E}[T_{\mathrm{comp}}|{\varepsilon_1}^{\mathrm{c}} \cap \varepsilon_2]\mathbb{P}[\varepsilon_2] \leq \mathbb{E}[\hat{T}|{\varepsilon_1}^{\mathrm{c}} \cap \varepsilon_2]\mathbb{P}[\varepsilon_2] \\ &\quad = \mathbb{E}[\hat{T}|\varepsilon_2]\mathbb{P}[\varepsilon_2] \\ &\quad \leq \mathbb{E}[\hat{T}] \leq \mathbb{E}[T_{\mathrm{max}}] \end{aligned} \tag{3.19}$$

where T_{max} is the time by which all nodes finish computing their partial gradients. T_{max} is the $(n_1 + n_1 \cdot n_2)^{th}$ order of statistic of all the random variables representing compute time of the nodes. So

$$\mathbb{E}[T_{\mathrm{max}}] = \frac{r_{\mathrm{irr}} d}{\mu}(\log n_1 + \log(1 + n_2)) + a r_{\mathrm{irr}} d + o(1) \tag{3.20}$$

From (17), (18), (19) and (20), we have

$$\begin{aligned}\mathbb{E}[T_{\text{comp}}|{\varepsilon_1}^{\text{c}}] &\leq \Theta(\log(\min(n_1, n_2))) + \Theta(\log n_1 + \log n_2)\\ &\leq \Theta(\log n_1 + \log n_2)\end{aligned} \tag{3.21}$$

From Lemma 2 in [**3**] for concentration inequality, for any group of siblings gr in layer 1 we have

$$\mathbb{P}[T^{gr}_{(n_1-s_1)} > \mathbb{E}[T^{gr}_{(n_1-s_1)}] \quad + \Theta(1/n_1^{1/4})] \leq e^{-\Theta(\sqrt{n_1})} \tag{3.22}$$

Similarly, for layer 2

$$\mathbb{P}[T^{gr}_{(n_2-s_2)} > \mathbb{E}[T^{gr}_{(n_2-s_2)}] \quad + \Theta(1/n_2^{1/4})] \leq e^{-\Theta(\sqrt{n_2})} \tag{3.23}$$

Using (22) and (23), and applying union bound of probability, we obtain probability on ε_1^c

$$\mathbb{P}[{\varepsilon_1}^{\text{c}}] \leq (1 + n_1)e^{-\Theta(\sqrt{min(n_1,n_2)})} \tag{3.24}$$

Putting together (15), (16), (21) and (24), we obtain

$$\begin{aligned}\mathbb{E}[T_{\text{comp}}] &\leq \frac{r_{\text{irr}}d}{\mu}\log\left(\frac{1}{\alpha}\right) + ar_{\text{irr}}d + o(1) + \Theta(\log n_1 + \log n_2)(1 + n_1)e^{-\Theta(\sqrt{min(n_1,n_2)})}\\ &\leq \frac{r_{\text{irr}}d}{\mu}\log\left(\frac{1}{\alpha}\right) + ar_{\text{irr}}d + o(1)\end{aligned} \tag{3.25}$$

Finally, from (14) and (25) we obtain the required upper bound:

$$\mathbb{E}[T_{\text{C}}] \leq \frac{r_{\text{irr}}d}{\mu}\log(\frac{1}{\alpha}) + ar_{\text{irr}}d + (n_1 + n_2)(1 - \alpha)t_{\text{c}} + o(1)$$

The computation model adopted to analyze TGC enables us to theoretically simulate it. Completion time for a worker node can be thought of as a random variable which is an interplay of its own computation random variable, the r.v. of completion time of its children and the communications involved with its children. The r.v. at the master node is the completion time we seek. After all the required parameters are set, in one run of the simulation, we generate the exponential computation time r.v. of each node and eventually obtain the total completion time. Averaging over sufficiently large number number of such r.v., we claim the obtained value to be the required expected value of the r.v. In verifying the expressions we obtain, we compare them against the values obtained from simulation as the values to be attained. The parameters required for simulation are appropriately set.

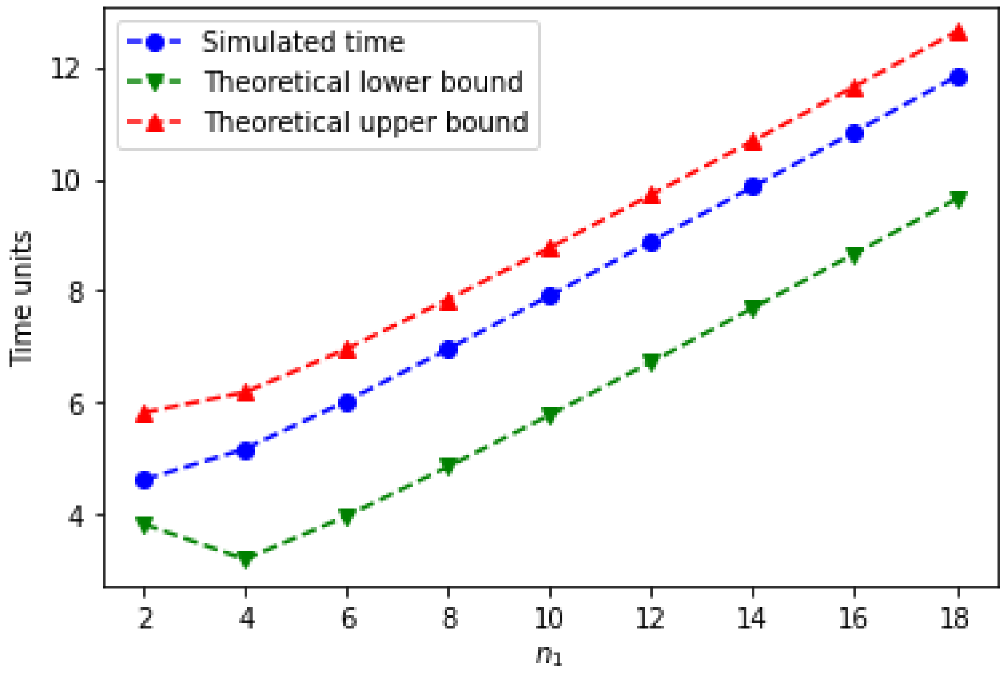

Figure 3.2: Expected time obtained from simulations, theoretical lower bound and upper bound by varying n_1

The expressions for bounds on expected time are compared with the simulated values. The number of data points in data set d is taken to be 100,000 and value of a is taken such that minimum computation time for each worker node is 1 time unit. Communication time is taken to be 1 time unit (to be comparable with the computational time). Straggler ratio is taken to be 0.5. The completion time for a simulation instance is calculated in accordance with the gradient computation scheme. Simulations are run 10,000 times, each time re-assigning computation times to worker nodes according to the exponential distribution. Average of all the times obtained in simulations are taken into consideration for comparing with the values obtained from the lower and upper bound expressions.
Plots for obtained time values, and from lower and upper bound expressions by varying values of n_1 and n_2 are shown in Fig. 3.2 and Fig. 3.3 respectively. While varying n_1, n_2 is taken to be 4 and vice versa. Upon observation, we can conclude that the lower and upper bounds rightly bound the expected time over the irregular coded reduce topology.

While the asymptotic analysis helps in deriving insights on the impact the topology (n_1 and n_2) and different parameters (such as t_c, α) have in speeding up total gradient computation, they are not adequate to make more detailed analysis. In the following

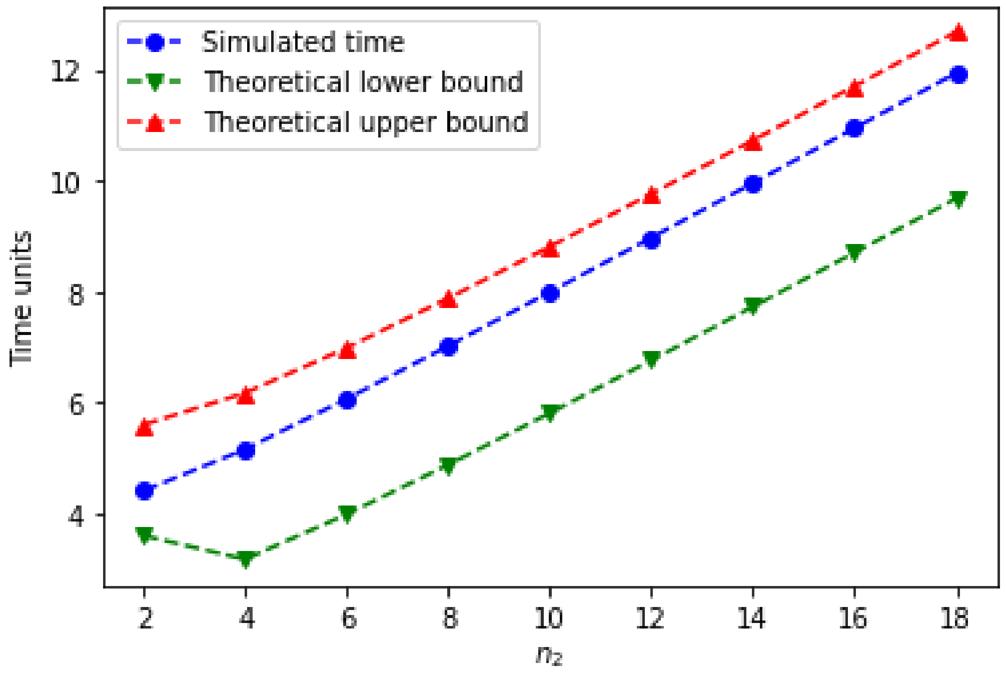

Figure 3.3: Expected time obtained from simulations, theoretical lower bound and upper bound by varying n_2

section, we demonstrate a methodology for exact analysis and approximation.

3.4 Exact and Approximate analysis

Communication time plays an important role in determining the completion time. The interplay between the events of local gradient computation and reception of partial gradients from children is a key aspect used to make the analysis. We discretely consider different cases of communication time t_c in comparison to the expected computation time of worker node $\mathbb{E}[T]$:

- Case 1: $t_c << \mathbb{E}[T]$
- Case 2: $t_c >> \mathbb{E}[T]$
- Case 3: $t_c \approx \mathbb{E}[T]$

We will analyze the first two cases. Analysis of the third case is out of the scope of our work as it highly complicates the interaction between computation of sibling nodes and their communication with their parent.

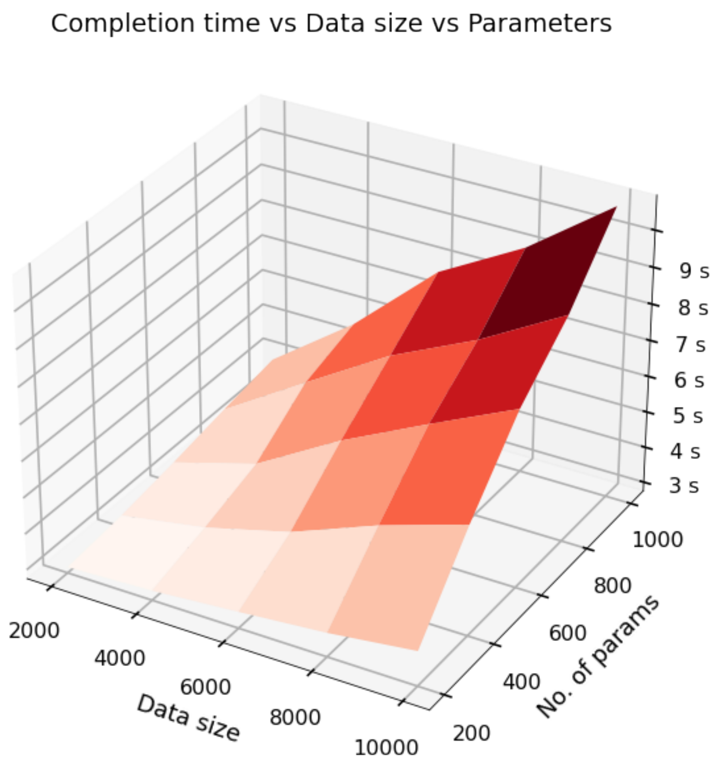

Figure 3.4: Completion times obtained over implementation on EC2 nodes

3.4.1 Case 1: $t_c << \mathbb{E}[T]$

This case is motivated to be analyzed via empirical observations on communication and computation times. TGC was deployed on 13 Amazon EC2 nodes in Python using the mpi4py[**1**] message passing library. The number of children is $n = 3$, with $s = 1$ stragglers consisting of two levels. Linear regression task was run over 1000 epochs. Dataset for the linear regression task has been randomly generated. For every 100th epoch, time to compute the local gradient is recorded. First successful communication from a node in level 1 to master is measured every 100th epoch and taken to be one instance of communication latency. Average of all observations is taken under consideration. The completion times obtained are shown in Fig.3.4. Dataset sizes are varied from 2,000 to 10,000 and number of parameters are varied from 200 to 1000.

The completion time increases with data size as well as the number of parameters involved, which is to be expected. The average computation and communication times were obtained as shown in Fig.3.5. We can observe that the communication times are considerably smaller than computation times as the size of dataset and number of pa-

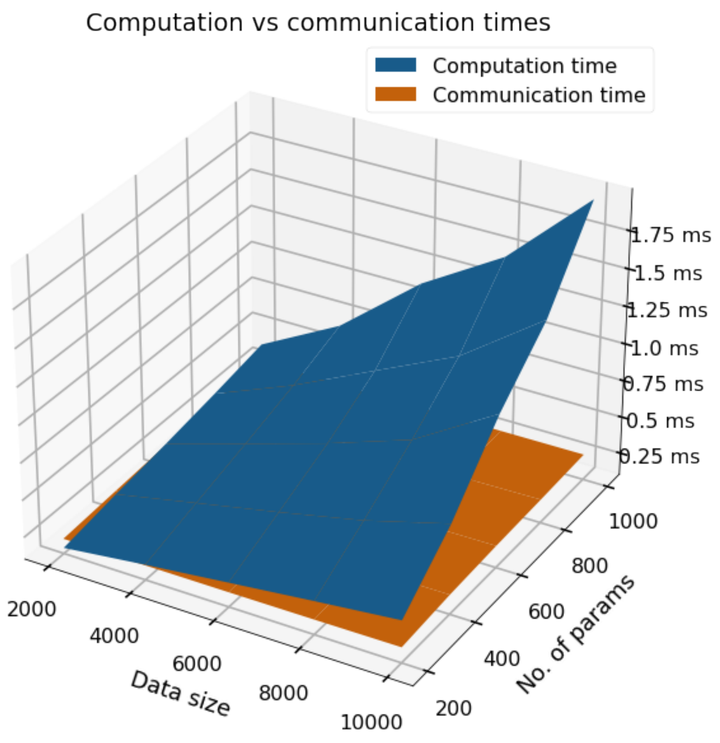

Figure 3.5: Computation and communication times

rameters increase. Even if we specifically consider the times for 1000 parameters which is shown in Fig.3.6, we can observe that the computation time significantly dominates over communication time. This motivates us to analyze the case where $t_c << \mathbb{E}[T]$.

Some notations are introduced here which will be used for our analysis. Random variables (r.v.) are denoted by capital letters such as T, X, B. If X is a r.v., then the cumulative distribution function (c.d.f.) of X is denoted by $F_X(x)$, which refers to the following probability event:

$$F_X(x) = \mathbb{P}(X \leq x)$$

Probability density function (p.d.f.) is the differential of the corresponding cumulative function and is denoted by $f_X(x)$ for r.v. X:

$$f_X(x) = \frac{dF_X(x)}{dx}$$

Conversely,

$$F_X(x) = \int_0^x f_X(y)dy$$

For a tree gradient coding setup, the time taken to compute gradient on local data for each node is an exponential random variable denoted by **T**. The c.d.f. of T is expressed

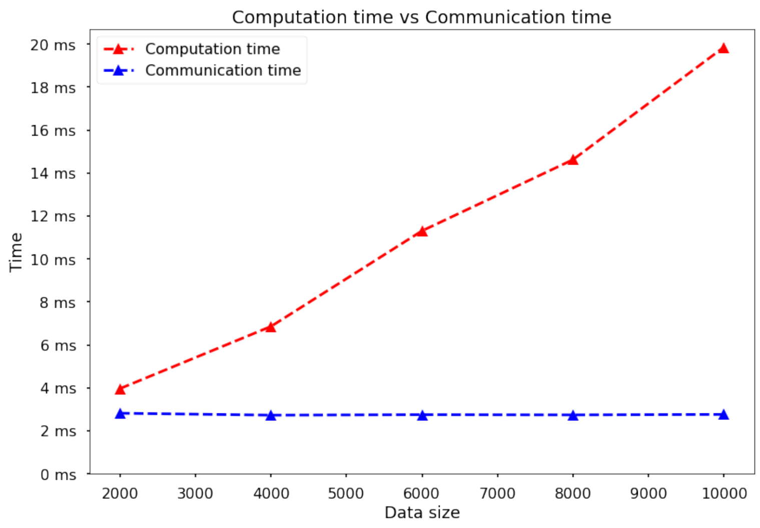

Figure 3.6: Computation and communication times over different datasizes for 1000 parameters. Computation time dominates over communication latency

as follows:

$$F_T(t) = \mathbb{P}(T \le t) = 1 - e^{-\lambda(t-t_0)}$$

if $t \ge t_0$ and 0 otherwise. The exponential parameter $\lambda = \frac{\mu}{d}$
A *branch* refers to a set of sibling leaf nodes and their parent. A branch is said to have completed its gradient computation when the parent has computed its local gradient and has received partial gradients from its non-straggling children. Random variable for the computation time for a branch is denoted by $\mathbf{B}$. Random variable $\mathbf{T\dot{}C}$ denotes the completion time of one epoch.
Among n instances of r.v. X $\{X_1, X_2, \ldots X_n\}$, the i^{th} order statistic of the group is the i^{th} smallest value among the n instances. It is denoted by $X_{(i)}$.

Firstly, the k^{th} order of statistic of T among n r.v. is analysed. The probability density function of $T_{(k)}$ would be:

$$f_{T_{(k)}}(t) = n f_T(t) \binom{n-1}{k-1} F_T(t)^{k-1} (1 - F_T(t))^{n-k} \tag{3.26}$$

This can be explained as follows: to get the probability density at t for k^{th} order statistic, we need to pick one r.v. among n r.v. and have it occur at t. There are n ways to pick a r.v. and the probability density that it occurs at t is $f_T(t)$. For this r.v. to be the k^{th} order statistic, we need to have exactly $k-1$ r.v. to occur before t. There are $\binom{n-1}{k-1}$ ways to choose $k-1$ r.v. among $n-1$ available r.v. and the probability that all of them occur before t would be the product of individual c.d.f, which is $F_T(t)^{k-1}$. The rest $n-k$ r.v must all occur after t, the probability of which will be $(1 - F_T(t))^{n-k}$. The c.d.f. of $T_{(k)}$ would be:

$$F_{T_{(k)}}(t) = \int_{t_0}^{t} n\binom{n-1}{k-1} F_T(u)^{k-1}(1 - F_T(u))^{n-k} f_T(u)du$$

which can be simplified to:

$$F_{T_{(k)}}(t) = n\binom{n-1}{k-1} \int_{0}^{F_T(t)} y^{k-1}(1-y)^{n-k}dy$$

by change of variable in integration from $F_T(u)$ to y. This integral is of the form of incomplete beta function:

$$F_{T_{(k)}}(t) = n\binom{n-1}{k-1} B(F_T(t); k, n-k+1)$$

where $B(x; a, b)$ is the incomplete beta function (not to be confused with r.v. of branch completion time). The c.d.f. turns out to be:

$$F_{T_{(k)}}(t) = n\binom{n-1}{k-1}\sum_{i=0}^{n-k}\left[\frac{(-1)^i\binom{n-k}{i}}{i+k}\left(1 - e^{-\lambda(t-t_0)}\right)^{i+k}\right] \tag{3.27}$$

This result has been verified by simulations. The c.d.f of $T_{(k)}$ was obtained from simulation by counting the fraction of instances where k^{th} smallest r.v. among n r.v. is smaller than t. The results for the case where $n = 6$ for different values of k are depicted in Fig.3.7

The c.d.f. obtained from the expression of p.d.f. overlaps with that simulated for all values of k considered, validating the result.

Branch computation time - B Having derived expression for distribution of $T_{(k)}$, analysis is made for branch computation time B. Branch computation time can be expressed as:

$$B = \max(T, T_{(n-s)})$$

Hence the c.d.f. for B would be:

$$F_B(t) = F_T(t) \cdot F_{T_{(n-s)}}(t) \tag{3.28}$$

Substituting expression of $F_{T_{(k)}}(t)$ obtained in (3.27) into the above equation, where $k = n - s$, we obtain:

$$F_B(t) = n\binom{n-1}{s}\sum_{i=0}^{s}\left[\frac{(-1)^i\binom{s}{i}}{i+n-s}\left(1 - e^{-\lambda(t-t_0)}\right)^{i+n-s+1}\right] \tag{3.29}$$

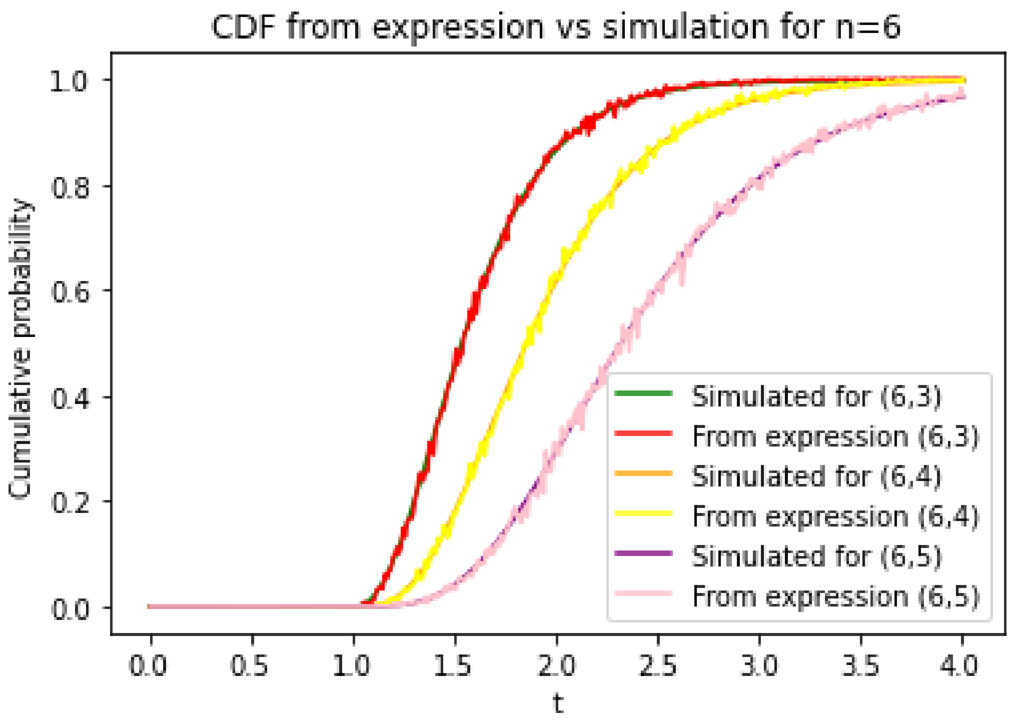

Figure 3.7: Comparing CDF obtained from expression against simulation for $n = 6$

The obtained c.d.f. is compared with the c.d.f obtained through simulation and the results are plotted in Fig.3.8.

For $n = 6$, different values of s are considered. For all the cases, the obtained c.d.f overlaps with the c.d.f obtained from simulation. This validates the above mentioned c.d.f for branch computation time B.

Using chain rule of differentiation, the p.d.f. of B can be obtained:

$$f_B(t) = f_T(t) \cdot F_{T_{(n-s)}}(t) + F_T(t) \cdot f_{T_{(n-s)}}(t) \tag{3.30}$$

Substituting equations (3.26) and (3.27) into the above equation, this turns out to be:

$$f_B(t) = n\binom{n-1}{s}\lambda e^{-\lambda(t-t_0)}\left(1-e^{-\lambda(t-t_0)}\right)^{n-s}\left(\sum_{i=0}^{s}\left[\frac{(-1)^i\binom{s}{i}}{i+n-s}\left(1-e^{-\lambda(t-t_0)}\right)^i\right]+e^{-\lambda(t-t_0)s}\right)$$

Completion time - T_C Finally, the time taken for complete gradient computation T_C is analysed. T_C is simply $B_{(n-s)}$. The p.d.f. of k^{th} order statistic for B is derived in a similar manner as that of equation (1).

$$f_{T_C}(t) = n f_B(t)\binom{n-1}{s}F_B(t)^{n-s-1}(1-F_B(t))^s$$

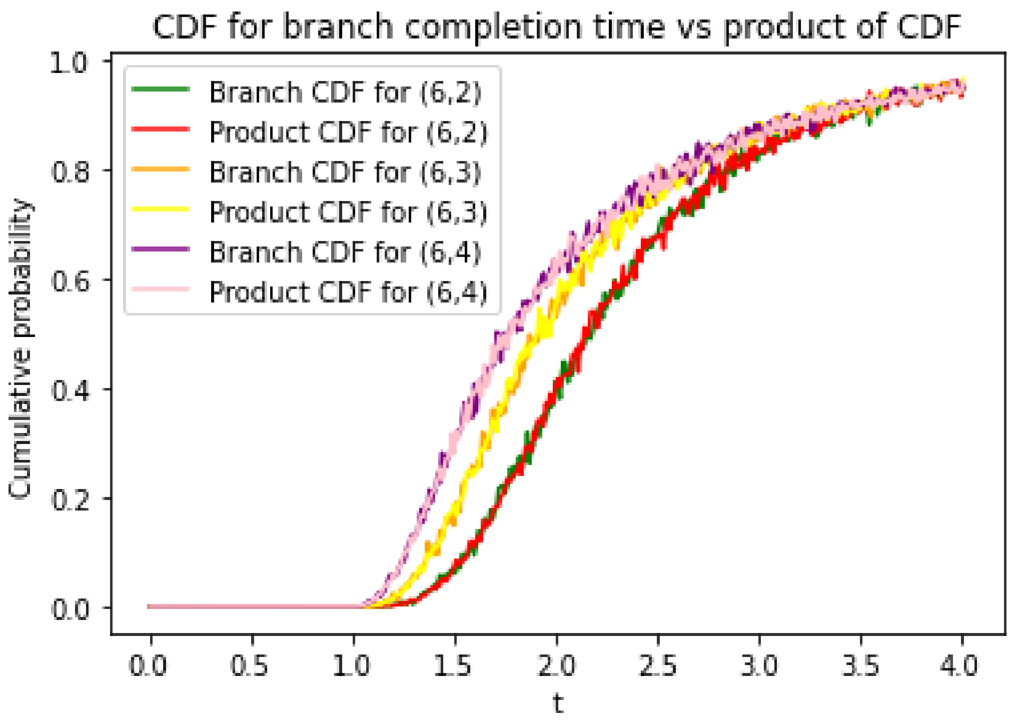

Figure 3.8: Comparing CDF for branch completion time from simulation against obtained expression for $n = 6$

The above result can be used to derive the c.d.f. for the completion time:

$$F_{T_C}(t) = \int_0^t f_{B_{(n-s)}}(u)du \tag{3.31}$$

Previously obtained results were used to compute $f_{B_{(k)}}(t)$ and this was used in arriving at $F_{T_C}(t)$. This was compared against the completion time obtained from simulation for various values of s for $n = 6$ and the results are plotted in Fig.3.9:

The plots overlap for each value of s which validates the expression obtained.

Expected completion time The expected completion time would be:

$$\mathbb{E}[T_C] = \int_{t_0}^{\infty} t{\cdot}f_{T_C}(t)dt$$

Due to the complexity of the expression f_{T_C}, it was not possible to obtain a closed-form expression for the expected completion time. An alternate perspective adopted is to look at the expected number of branch computations that finish within time t. Then, the time

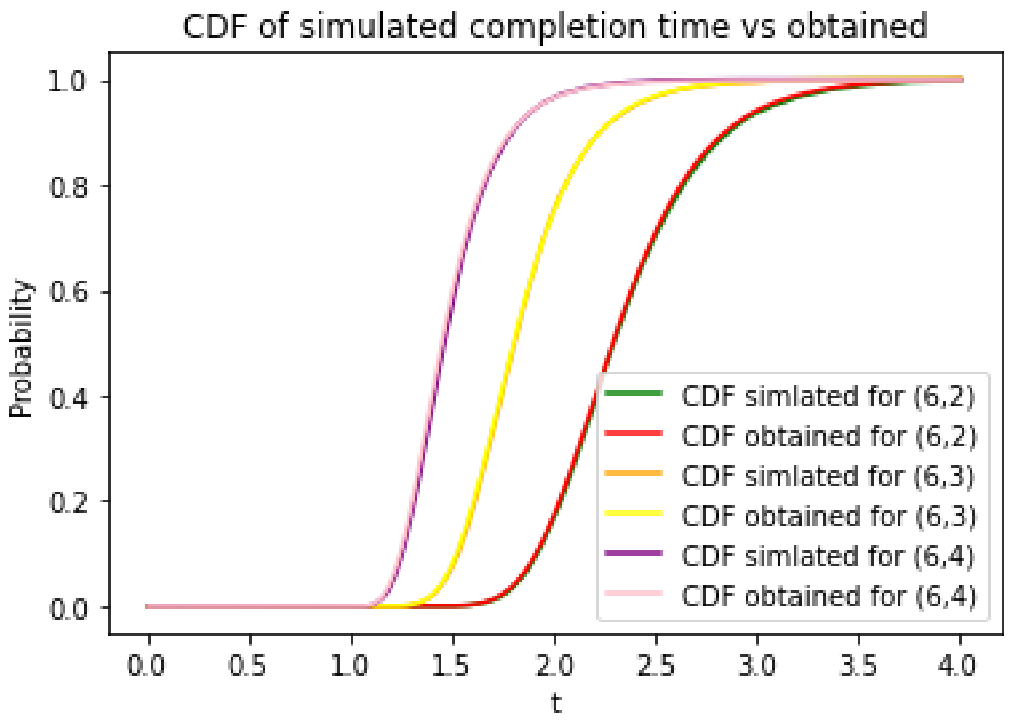

Figure 3.9: Comparing CDF for completion time from simulation against obtained expression for $n = 6$

at which the expected number of branch computations completed is $n-s$ is assumed to be the expected completion time. This obtained time is compared against actual expected time from simulations to see how accurate the results are.
Let $N_B(t)$ represent the random variable for number of instances of B out of n instances that complete within t. The expected value of $N_B(t)$ would be:

$$\mathbb{E}[N_B(t)] = n{\cdot}\mathbb{P}(B \leq t)$$

As the c.d.f for B is already obtained, this is used to compute the above expected value. This is plotted in Fig.3.10 for various t against the expected value obtained from simulation.

The values match with those of simulation for different values of s where $n = 6$. The values of t are extracted where $\mathbb{E}[N_B(t)] = n - s$ and is compared against the expected completion time from simulation. These values are tabulated in Table.3.1

The values match very nearly with those obtained from simulation. This indicates that the method used to obtain the expected completion time is fairly accurate.

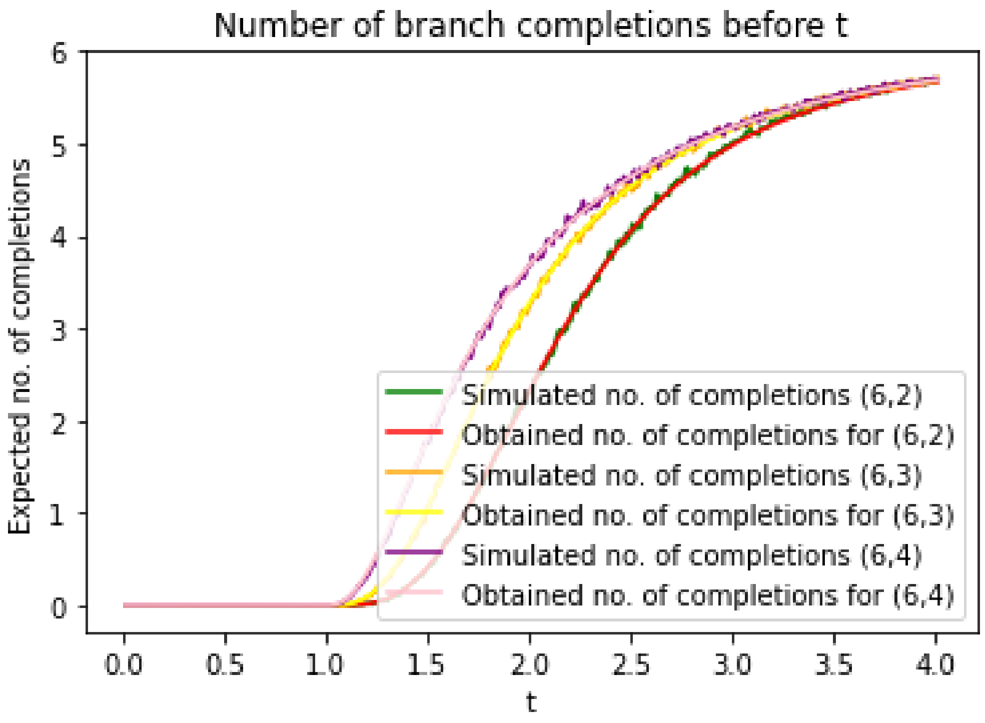

Figure 3.10: Comparing expected number of branch completions from simulation against obtained expression for $n = 6$

(n, s)	**Completion time from simulation**	**Obtained completion time**	**Percent error**
(6, 2)	2.36	2.48	4.80
(6, 3)	1.85	1.92	4.10
(6, 4)	1.50	1.54	3.15

Table 3.1: Completion time obtained from simulation against that obtained from time to complete $n - s$ branch completions

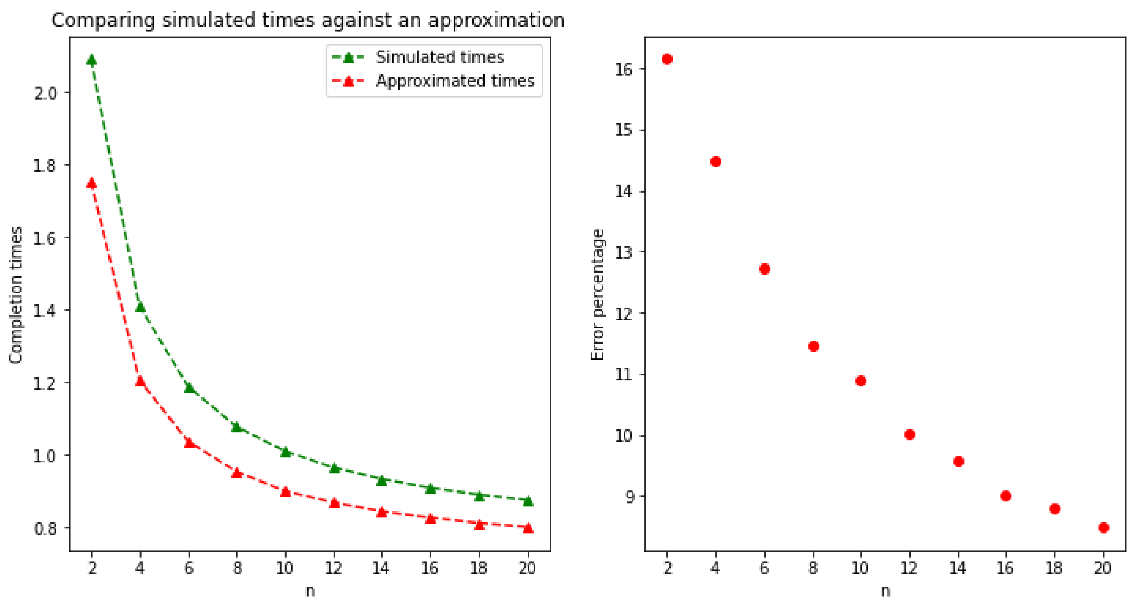

Figure 3.11: Approximation vs simulated completion time for small t_c

Approximation To simplify the analysis, we can assume that the computation time of a node at level 1 is masked by the slowest non-straggling child. This would be a fair assumption as the $(n-s)^{th}$ order statistic of T would most probably yield a higher value than a single instance of T. As a result, the branch computation time could be approximated as:

$$B = T_{(n_2-s_2)} + t_c$$

and as the completion time is a $(n_1-s_1)^{th}$ order statistic of B with an additional computation tailed at the end of completion of non-straggling branches:

$$T_C = B_{(n_1-s_1)} + t_c$$

The completion time can hence be approximated as:

$$T_C = \left(\left(T_{(n_2-s_2)}\right)_{(n_1-s_1)}\right) + L{\cdot}t_c$$

where the expected completion time would be:

$$\mathbb{E}[T_C] = \mathbb{E}\left[\left(\left(T_{(n2-s2)}\right)_{(n1-s1)}\right)\right] + L{\cdot}t_c$$

The approximation was compared against simulated completion times by setting $n_1 = n_2$ (for reducing dimension of analysis to conveniently view results). This is plotted in Fig.3.11 We observe that the relative error varies between $10-20\%$, suggesting that the approximation is reasonably good.

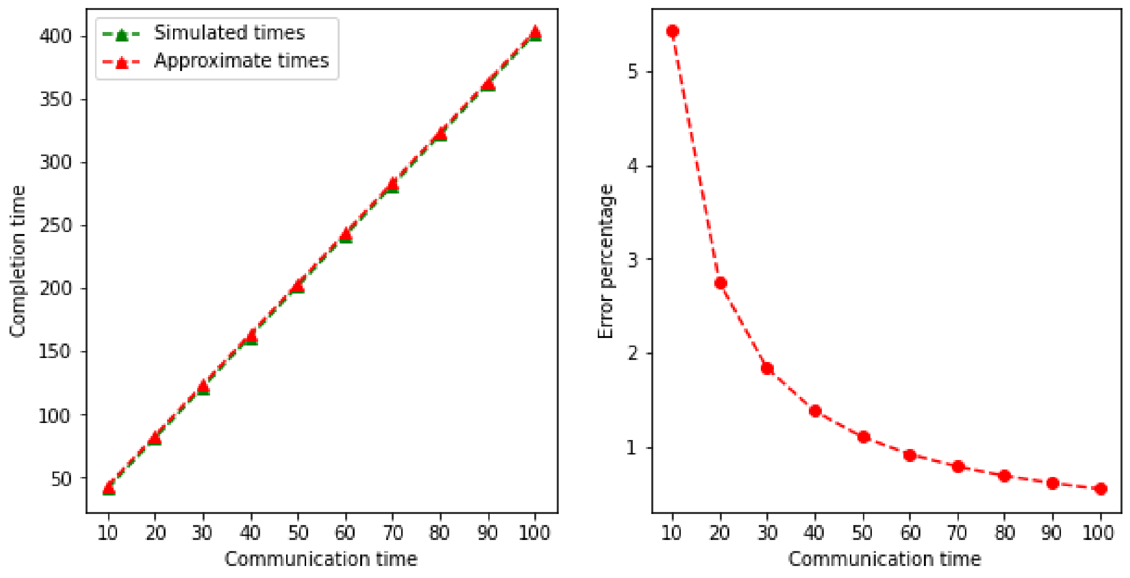

Figure 3.12: Approximation against simulated completion times for large t_c

3.4.2 Case 2: $t_c >> E[T]$

If communication time dominates over the time taken by nodes to compute local gradients, the probability that the nodes at upper levels continue computing by the time the children have communicated their responses would be very small. The completion time in this scenario can be approximated by considering the computation time by the leaf nodes and the time taken to communicate all the necessary partial gradients by the non-straggling nodes to their parents.

$$T_C = T + L{\cdot}(n - s){\cdot}t_c$$

from which the expected comes out to be:

$$\mathbb{E}[T_C] = \mathbb{E}[T] + L{\cdot}(n - s){\cdot}t_c$$

The approximation has been compared against the simulated values for $n_1 = n_2 = 4$, $\alpha = 2$ and number of levels $L = 2$. The communication time is varied by factors of 10 to 100 against individual computation time. The approximation is compared against simulated completion time and is plotted in Fig.3.12. The relative error for the approximation considered is very small, indicating the approximation closely estimates the completion time.

Chapter 4

Conclusion

This study presented a theoretical analysis of the completion time of Tree Gradient Coding which is used to compute gradient in a distributed manner. We considered irregular tree topology in our analysis. We started by deriving computation load for the general case of irregular tree topology for TGC. We then shifted our attention to topologies consisting of two levels. We characterized the completion time by deriving asymptotic upper and lower bounds, which were verified by simulations. Upon performing implementations on Amazon EC2 nodes, we found that the communication delay was insignificant in comparison to individual computation times. We then adopted an exact approach to derive completion time for the same case. Although we were unable to arrive at a closed-form expression, we believe the methodology used for the analysis can be adopted to analyze other similar distributed computing schemes. Approximations were also used in simplifying the analysis and arriving at expressions, which were validated by simulations.

The theoretical analysis presented in this study hopes to provide valuable insights into the design and optimization of TGC and similar distributed machine learning systems, and it can guide researchers and practitioners in selecting the most efficient topologies to minimize the completion time of the algorithm. Further research can build on these findings by conducting experiments to validate the theoretical analysis and explore more complex topologies and computation schemes.

Chapter 5

Future Work

The analysis provided in this work can be extended to obtain a closed-form expression for the exact analysis. This would enable us to solve very useful optimization problems. For instance, if we are given a number of nodes N, we can hope to find the topology which would yield minimum completion time. Specifically, if we consider an irregular tree topology over two levels, we would intend to minimize completion time $T_C(n_1, n_2)$ subject to the constraint on total number of worker nodes $N \geq n_1 + n_1{\cdot}n_2$. This problem can be generalized to topology of any number.

Rather than the results themselves, the methodology adopted in the analysis can prove to be useful in analyzing other variants of TGC and for master-worker based systems in general.

Other ways to model computation of systems can be explored and applied in analyzing the metrics studied in this work. Exponential model of computation is considered for our work. The parameters of computation a and μ are intrinsic to the systems that participate in computation. Concrete ways to determine the values of these parameters can be explored to empirically validate the theoretical models used for computation.

Bibliography

[1] https://github.com/mpi4py/mpi4py.

[2] Rashish Tandon et al. "Gradient Coding: Avoiding Stragglers in Distributed Learning". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 3368–3376.

[3] Amirhossein Reisizadeh et al. "CodedReduce: A Fast and Robust Framework for Gradient Aggregation in Distributed Learning". In: *IEEE/ACM Trans. Netw.* 30.1 (Sept. 2021), pp. 148–161. ISSN: 1063-6692. DOI: 10.1109/TNET.2021.3109097. URL: https://doi.org/10.1109/TNET.2021.3109097.

[4] Amirhossein Reisizadeh and Ramtin Pedarsani. "Latency analysis of coded computation schemes over wireless networks". In: *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 2017, pp. 1256–1263. DOI: 10.1109/ALLERTON.2017.8262881.

[5] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[6] Ritik Dixit, Rishika Kushwah, and Samay Pashine. "Handwritten Digit Recognition using Machine and Deep Learning Algorithms". In: *International Journal of Computer Applications* 176.42 (July 2020), pp. 27–33. DOI: 10.5120/ijca2020920550. URL: https://doi.org/10.5120%2Fijca2020920550.

[7] Joost Verbraeken et al. "A Survey on Distributed Machine Learning". In: *ACM Comput. Surv.* 53.2 (Mar. 2020). ISSN: 0360-0300. DOI: 10.1145/3377454. URL: https://doi.org/10.1145/3377454.

[8] Jianmin Chen et al. *Revisiting Distributed Synchronous SGD*. 2017. arXiv: 1604.00981 [cs.LG].

[9] Sahasrajit Sarmasarkar, V. Lalitha, and Nikhil Karamchandani. "On Gradient Coding with Partial Recovery". In: *2021 IEEE International Symposium on Information Theory (ISIT)*. 2021, pp. 2274–2279. DOI: `10.1109/ISIT45174.2021.9517849`.

[10] Zachary Charles, Dimitris Papailiopoulos, and Jordan Ellenberg. *Approximate Gradient Coding via Sparse Random Graphs*. 2017. arXiv: `1711.06771 [stat.ML]`.

[11] Haozhao Wang et al. *Heterogeneity-aware Gradient Coding for Straggler Tolerance*. 2019. arXiv: `1901.09339 [cs.DC]`.

[12] Ela Bhattacharya et al. "Improved Tree Gradient Coding with Non-uniform Computation Load". In: *ICC 2021 - IEEE International Conference on Communications*. 2021, pp. 1–6. DOI: `10.1109/ICC42927.2021.9500717`.